OMRON

V+ Keyword

Reference Manual

I672-E-04

NOTE

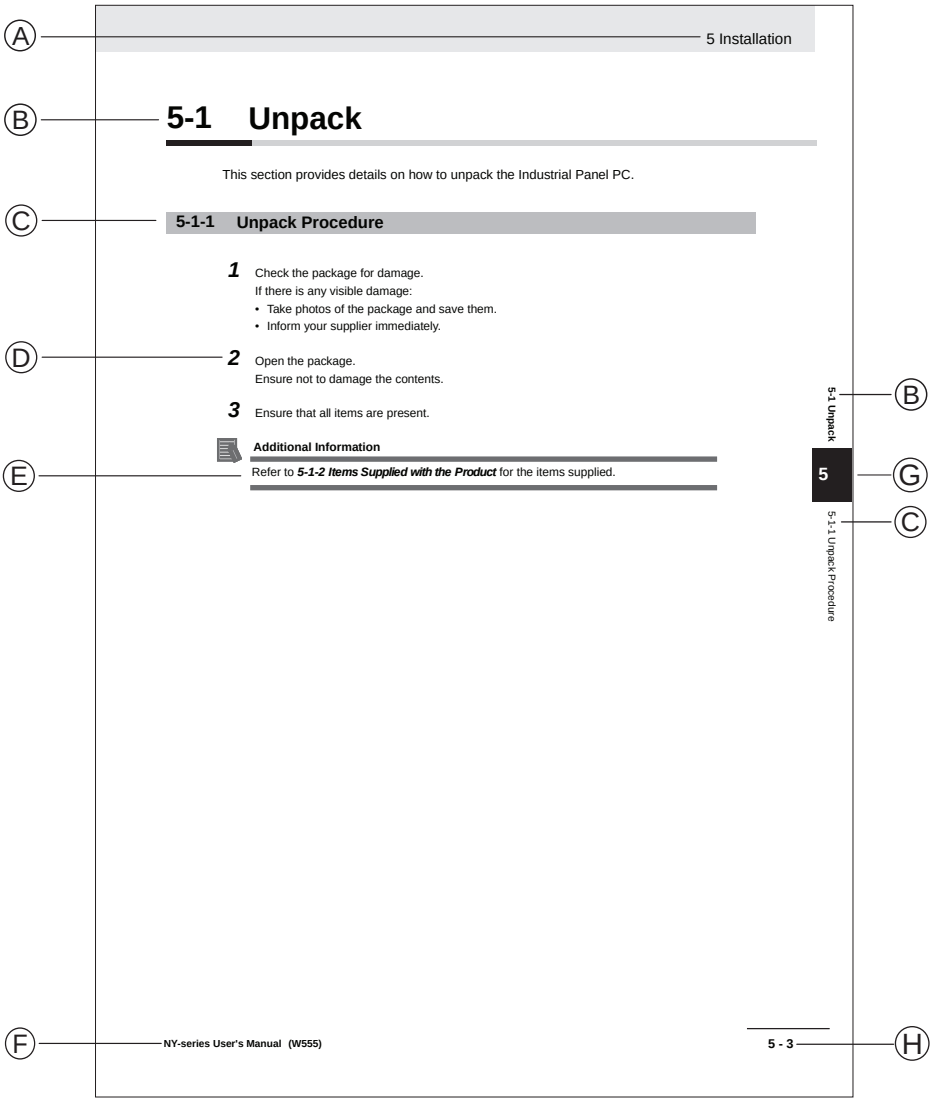
Trademarks

Copyrights

Manual Information

Page Structure

The following page structure is used in this manual.



Note: This illustration is provided as a sample. It will not literally appear in this manual.

Item	Explanation	Item	Explanation
A	Level 1 heading	E	Special Information
B	Level 2 heading	F	Manual name
C	Level 3 heading	G	Page tab with the number of the main section
D	Step in a procedure	H	Page number

Special Information

Special information in this manual is classified as follows:

Precautions for Safe Use

Precautions on what to do and what not to do to ensure safe usage of the product.

Precautions for Correct Use

Precautions on what to do and what not to do to ensure proper operation and performance.

Additional Information

Additional information to read as required.
This information is provided to increase understanding or make operation easier.

Version Information

Information on differences in specifications and functionality between different versions.

Sections in this Manual

CONTENTS

Terms and Conditions Agreement

Warranty and Limitations of Liability

Warranty

● Exclusive Warranty

Omron's exclusive warranty is that the Products will be free from defects in materials and workmanship for a period of twelve months from the date of sale by Omron (or such other period expressed in writing by Omron). Omron disclaims all other warranties, expressed or implied.

● Limitations

OMRON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, ABOUT NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OF THE PRODUCTS. BUYER ACKNOWLEDGES THAT IT ALONE HAS DETERMINED THAT THE PRODUCTS WILL SUITABLY MEET THE REQUIREMENTS OF THEIR INTENDED USE.
Omron further disclaims all warranties and responsibility of any type for claims or expenses based on infringement by the Products or otherwise of any intellectual property right.

● Buyer Remedy

Omron's sole obligation hereunder shall be, at Omron's election, to (i) replace (in the form originally shipped with Buyer responsible for labor charges for removal or replacement thereof) the non-complying Product, (ii) repair the non-complying Product, or (iii) repay or credit Buyer an amount equal to the purchase price of the non-complying Product; provided that in no event shall Omron be responsible for warranty, repair, indemnity or any other claims or expenses regarding the Products unless Omron's analysis confirms that the Products were properly handled, stored, installed and maintained and not subject to contamination, abuse, misuse or inappropriate modification. Return of any Products by Buyer must be approved in writing by Omron before shipment. Omron Companies shall not be liable for the suitability or unsuitability or the results from the use of Products in combination with any electrical or electronic components, circuits, system assemblies or any other materials or substances or environments. Any advice, recommendations or information given orally or in writing, are not to be construed as an amendment or addition to the above warranty.
See http://www.omron.com/global/ or contact your Omron representative for published information.

Limitations of Liability

OMRON COMPANIES SHALL NOT BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR PRODUCTION OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCTS, WHETHER SUCH CLAIM IS BASED IN CONTRACT, WARRANTY, NEGLIGENCE OR STRICT LIABILITY. Further, in no event shall liability of Omron Companies exceed the individual price of the Product on which liability is asserted.

Application Considerations

Suitability for Use

Omron Companies shall not be responsible for conformity with any standards, codes or regulations which apply to the combination of the Product in the Buyer's application or use of the Product. At Buyer's request, Omron will provide applicable third party certification documents identifying ratings and limitations of use which apply to the Product. This information by itself is not sufficient for a complete determination of the suitability of the Product in combination with the end product, machine, system, or other application or use. Buyer shall be solely responsible for determining appropriateness of the particular Product with respect to Buyer's application, product or system. Buyer shall take application responsibility in all cases.

NEVER USE THE PRODUCT FOR AN APPLICATION INVOLVING SERIOUS RISK TO LIFE OR PROPERTY WITHOUT ENSURING THAT THE SYSTEM AS A WHOLE HAS BEEN DESIGNED TO ADDRESS THE RISKS, AND THAT THE OMRON PRODUCT(S) IS PROPERLY RATED AND INSTALLED FOR THE INTENDED USE WITHIN THE OVERALL EQUIPMENT OR SYSTEM.

Programmable Products

- Omron Companies shall not be responsible for the user's programming of a programmable Product, or any consequence thereof.
- Omron Companies shall not be responsible for the operation of the user accessible operating system (e.g. Windows, Linux), or any consequence thereof.

Disclaimers

Performance Data

Data presented in Omron Company websites, catalogs and other materials is provided as a guide for the user in determining suitability and does not constitute a warranty. It may represent the result of Omron's test conditions, and the user must correlate it to actual application requirements. Actual performance is subject to the Omron's Warranty and Limitations of Liability.

Change in Specifications

Product specifications and accessories may be changed at any time based on improvements and other reasons. It is our practice to change part numbers when published ratings or features are changed, or when significant construction changes are made. However, some specifications of the Product may be changed without any notice. When in doubt, special part numbers may be assigned to fix or establish key specifications for your application. Please consult with your Omron's representative at any time to confirm actual specifications of purchased Product.

Errors and Omissions

Information presented by Omron Companies has been checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical or proofreading errors or omissions.

Introduction

This manual is OMRON's original instructions describing how to use V+ keywords.
V+ uses a special programming language and command set to send and request information to and from the robot control system. The keywords detailed in this manual are used with Sysmac Studio or ACE Software for creating V+ programs and issuing commands from the Monitor Window.
This manual contains IMPORTANT SAFETY INSTRUCTIONS. Please read this manual and make sure you understand V+ keyword features, functions, and concepts before attempting to use them.
SAVE THESE INSTRUCTIONS. Keep this manual in a safe place where it will be available for reference during operation.

Intended Audience

This manual is intended for the following personnel, who must also have knowledge of common programming practices and robotic control methods.

- Personnel in charge of introducing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of installing and maintaining FA systems.
- Personnel in charge of managing FA systems and facilities.

Version Information

The details in this document apply to products operating with the following hardware and software versions.

- Integrated Control Robot Firmware Version: 5.0C
- Standard Control Robot Firmware Version: 6.102C

Safety Precautions

Definition of Precautionary Information

The following notation is used in this manual to provide precautions required to ensure safe usage of the AMR. The safety precautions that are provided are extremely important to safety.
Always read and heed the information provided in all safety precautions.
The following notation is used.

DANGER	Identifies an imminently hazardous situation which, if not avoided, is likely to result in serious injury, and might result in fatality or severe property damage.
WARNING	Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury. Additionally, there may be severe property damage.
CAUTION	Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury, or property damage.

Symbols

	The circle and slash symbol indicates operations that you must not do. The specific operation is shown in the circle and explained in text. This example indicates prohibiting disassembly.
	The triangle symbol indicates precautions (including warnings). The specific operation is shown in the triangle and explained in text. This example indicates a precaution for electric shock.
	The triangle symbol indicates precautions (including warnings). The specific operation is shown in the triangle and explained in text. This example indicates a general precaution.
	The filled circle symbol indicates operations that you must do. The specific operation is shown in the circle and explained in text. This example shows a general precaution for something that you must do.
	The triangle symbol indicates precautions (including warnings). The specific operation is shown in the triangle and explained in text. This example indicates a precaution for high temperatures.
	The triangle symbol indicates precautions (including warnings). The specific operation is shown in the triangle and explained in text. This example indicates a precaution for laser radiation.

Dangers

⚠ DANGER

Before issuing the RESET program command, ensure all devices connected to the output signals can safely be turned OFF.	
Do not use the POWER switch to enable power from within a program unless your system is subject to European certification. With European certification, special safety features are built-in to the system to prevent the robot from being activated without warning	
Using the POWER 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 you do not have a clear view of the robot workspace or do not have immediate access to an emergency stop button.	

Warnings

⚠ WARNING

General

Digital and analog I/O are not affected by DRY.RUN, so external devices driven by analog or digital output commands still operate.	
It is important to execute the SETBELT program command each time the robot is going to track the belt to make sure the difference between the current belt position as returned by the BELT function and the belt position of the specified belt variable does not exceed 8,388,607 (^H7FFFFF) during active belt tracking. Unpredictable robot motion may result if the difference does exceed this value while tracking the belt.	
Typing a DO command with no program instruction specified can result in unexpected motion of the robot, because the previous DO operation is executed again.	
Entering an EXECUTE command with no program specified could result in unexpected motion of the robot, since the previous program is executed again.	
Do not issue the RESET command unless you are sure all output signals can be safely turned OFF. Be particularly careful of devices that are activated when a signal is turned OFF.	

Cybersecurity

To maintain the security and reliability of the system, a robust cybersecurity defense program should be implemented, which may include some or all of the following:

Anti-virus protection

- Install the latest commercial-quality anti-virus software on the computer connected to the control system and keep the software and virus definitions up-to-date.
- Scan USB drives or other external storage devices before connecting them to control systems and equipment.

Security measures to prevent unauthorized network access

- Install physical controls so that only authorized personnel can access control systems and equipment.
- Reduce connections to control systems and equipment via networks to prevent access from untrusted devices.
- Install firewalls to block unused communications ports and limit communication between systems. Limit access between control systems and systems from the IT network.
- Control remote access and adopt multifactor authentication to devices with remote access to control systems and equipment.
- Set strong password policies and monitor for compliance frequently.

Data input and output protection

- Backup data and keep the data up-to-date periodically to prepare for data loss.
- Validate backups and retention policies to cope with unintentional modification of input/output data to control systems and equipment.
- Validate the scope of data protection regularly to accommodate changes.
- Check validity of backups by scheduling test restores to ensure successful recovery from incidents.
- Safety design, such as emergency shutdown and fail-soft operations in case of data tampering and incidents.

Additional recommendations

- When using an external network environment to connect to an unauthorized terminal such as a SCADA, HMI or to an unauthorized server may result in network security issues such as spoofing and tampering.
- You must take sufficient measures such as restricting access to the terminal, using a terminal equipped with a secure function, and locking the installation area by yourself.
- When constructing network infrastructure, communication failure may occur due to cable disconnection or the influence of unauthorized network equipment.
- Take adequate measures, such as restricting physical access to network devices, by means such as locking the installation area.
- When using devices equipped with an SD Memory Card, there is a security risk that a third party may acquire, alter, or replace the files and data in the removable media by removing or unmounting the media.
- Please take sufficient measures, such as restricting physical access to the Controller or taking appropriate management measures for removable media, by means of locking and controlling access to the installation area.
- Educate employees to help them identify phishing scams received via email on systems that will connect to the control network.

Cautions

⚠ CAUTION

If a file is loaded with the /S switch value when using the LOAD monitor command, you should also use the /R switch value to prevent editing of the squeezed version of a file. If the squeezed version of the file is edited the unsqueezed version may be overwritten when the edits are saved.	
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.	

Precautions for Correct Use

- IOSTAT should be used after each I/O operation. If one I/O operation fails, a subsequent operation may have unexpected behavior. For example, if opening a file fails, a write operation may not be issued. I/O operation errors do not stop program execution and therefore cannot be handled asynchronously with REACTE.
- The statement "FDELETE *.*" can be used to delete all files in a subdirectory. Use the DEFAULT command to make sure you are in the correct subdirectory before issuing this command.
- The system switches are shared by all the program tasks. 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.
- Modern, structured programming considers GOTO statements to be poor programming practice. It is recommended to use one of the other control structures in place of GOTO statements.
- Do not allow multiple tasks to change DRY.RUN simultaneously, since the DRY.RUN state can then be different from that expected. Your programs should use a software interlock in this case.
- Disconnecting EtherCAT communications while the robot is transitioning between control states may cause an invalid fieldbus state. Cycle power to the robot to recover from this condition

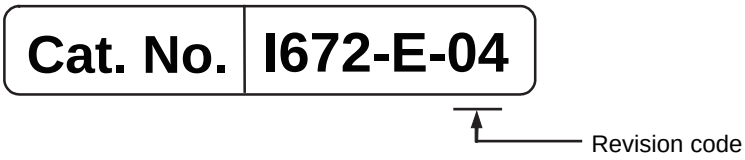
Related Manuals

Use the following related manuals for reference.

Manual	Description
V+ User's Manual (Cat. No. I671)	Provides information that is necessary to use V+.
Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)	Instruction for the use of the ACE Version 4 software.
Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)	Learning about the operating procedures and functions of the Sysmac Studio to configure Robot Integrated System using Robot Integrated CPU Unit.
Robot User Manuals	User Manuals for the robot in use.
Industrial Robot Fieldbus Configuration User's Guide (Cat. No. M130)	Provides information about fieldbus configurations for OMRON Industrial Robots.
NJ-series Robot Integrated CPU Unit User's Manual (Cat. No. O037)	Describes the settings and operation of the CPU Unit and programming concepts for OMRON robot control.
Robot Safety Guide (Cat. No. I590)	Contains safety information for OMRON industrial robots.
T20 Pendant User's Manual (Cat. No. I601)	Describes the settings and operation of the CPU Unit and programming concepts for OMRON robot control.

Revision History

A manual revision code appears as a suffix to the catalog number on the front and back covers of the manual.



Revision code	Date	Revised content
04	August 2025	Updates for EtherCAT SubDevice functionality and other improvements.
03	November 2023	Updates to OVERLAP/NOOVERLAP Keywords.
02	May 2023	PROFINET support updates and other improvements.
01	January 2022	Original production

1

Overview

The following sections provide an overview of the V+ Keywords syntax and parameters.

1-1 Keyword Syntax

Use the following guidelines with the keyword syntax provided in this manual.

- Keywords are presented with uppercase text and parameters are presented with lower- case text.
- Any parentheses, brackets, and commas must be used exactly as shown.
- Bold text represents required syntax and non-bold text represents optional syntax.
- Commas must be present between consecutive parameters.
- Blank spaces are not evaluated by the system and extra spaces or missing spaces do not cause syntax problems.

Use the following example to understand how keyword syntax is presented in this manual.

```
KEYWORD req_param1 = req_param2 SECONDARY.KEYWORD opt_param1, opt_param2
```

Item	Details
KEYWORD	Required and must be entered exactly as shown.
req_param1	Required and must be replaced with a value, variable, or expression.
=	Assignment operator. If required, follows the keyword and / or parameters.
req_param2	Required when an equal sign is present and must be replaced with a value, variable, or expression.
SECONDARY.KEYWORD	If present, required and must be entered exactly as shown.
opt_param1,opt_param2	Optional and if used, must be replaced with a value, variable, or expression.

1-2 Keyword Parameters

This section describes parameters and how they are used with keyword syntax.
This section does not describe System Parameter Keywords. For more information about System Parameter Keywords refer to the *2-5 System Parameter Keyword Summary* on page 2-20.
Parameters are considered arguments to keywords. Keywords may have multiple required and optional parameters. Some keywords do not use any parameters such as BREAK or FALSE .
When an optional parameter is omitted, the system will assume a default. Omitting an optional parameter that is in the middle of the keyword's syntax must respect the use of any necessary commas, as shown below.

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

If the optional parameter is trailing, it can be removed if unused. Consider the CALL keyword shown with the syntax structure CALL program(arg_list). If this keyword is used to simply execute a new subroutine program without passing arguments, the syntax would be as follows.

```
CALL program_name
```

1-2-1 Parameter Data Type Designations

The data type of the constant or variable must be the same type that is required by the keyword. String and numeric parameters can be constant values or any valid variable names. Use the following rules to designate parameter data types used in keyword syntax.

- String variables must be preceded with the $ symbol and string constants must be enclosed in quotes.
- Precision Point variables must be preceded with a # symbol.
- Belt variables must be preceded with a % symbol.
- Real and integer constants can be used without modification.

Additional Information

Keywords cannot be used as variable names or program names.

1-2-2 Numeric Parameters

Several types of numeric parameters can appear in keyword syntax. For each type of parameter, the value can generally be specified by a numeric constant, a variable name, or a mathematical expression. There are some restrictions on the numeric values that are accepted by V+. The following rules determine how a value will be interpreted in the various situations.

- Distances are used to define locations to where a robot is to move. The unit of measure for distances is in millimeters, although units are never explicitly entered for any value. Values entered for distances can be positive or negative. Refer to the IPS keyword for a special case of specifying robot speed in inches.
- Joint numbers are integers from 1 up to the number of joints in the robot. Refer to the supporting robot user manual for information about robot joint numbering arrangements.
- Signal numbers are used to identify digital signals. They are always considered as integer values. A negative signal number indicates an OFF state. Refer to the supporting robot user manual and *V+ User's Manual (Cat. No. I671)* for more information about signal numbers for your particular robot.

- Integer parameters can be satisfied with real values (values with integer and fractional parts). When an integer is required, the value is rounded and the resulting integer is used.
- Parameters indicated as being scalar variables can be satisfied with a real value (with integrand fractional parts) except where noted. Scalar variables can range from -9.22*1018 to 9.22*1018 in value (displayed as -9.22E18 and 9.22E18). Numbers declared to be double-precision values can range from -1.8*10-307 to 1.8*10-307.

All numbers in this manual are decimal unless otherwise noted. Binary numbers are shown as ^B, octal numbers as ^, and hexadecimal numbers as ^H.

1-3 Integrated and Standard Control

When V+ is used with the Standard Control system, use ACE software to create I/O, logic, safety, and motion control functions.

When V+ is used with Robot Integrated Control, use IEC 61131-3 programming language specifications to create I/O, logic, safety, and motion control functions. With the addition of shared variables, shared signals, and function blocks for interacting with V+ program execution, you can develop an entire control solution from the software. Refer to *V+ User's Manual (Cat. No. I671)* and *NJ-series Robot Integrated CPU Unit User's Manual (Cat. No. O037)* for more information.

Make the following considerations when using keywords that specify a robot number.

- When these keywords are used with Standard Control systems, you must specify [1] as the robot number parameter. If no value or a value > 1 is used, the error *No robot connected to system (-622)* will be returned.
- When these keywords are used with a Robot Integrated Control system, all configured robots will be affected when specifying [] or [0] for the robot number parameter. Use the specific robot number to affect a particular robot in a multi-robot system.

Keyword types described below have a longer execution time when used with Robot Integrated Control systems compared to Standard systems. Refer to *V+ User's Manual (Cat. No. I671)* for more information about keyword syntax, parameters, and data types.

Keyword Type	Usage
Function keywords	Used to return values from the V+ control system.
Monitor command keywords	Used to issue individual operations in the Monitor Window or to create Monitor Command programs.
Program command keywords	Used to command operations in V+ Programs.
System parameter keyword	Used to manipulate system parameters in V+ Programs or with the Monitor Window.
System switch keywords	Used to manipulate system switches in V+ Programs or with the Monitor Window.
Other keywords	Used to specify units when using the SPEED program command keyword. Refer to *V+ User's Manual (Cat. No. I671)* for more information.

2

Keyword Quick Reference

Use the sections below as a quick reference to all keywords described in this document.

2-1 Function Keyword Summary

The table below provides a summary of all function keywords.

Keyword & Page Number	Description
3-1-1 ABS on page 3-8	Real-valued function that returns the absolute value (magnitude) of the argument provided.
3-1-2 ACCEL on page 3-8	Real-valued function that returns the current setting for robot acceleration, deceleration, or the maximum allowable percentage limits defined in the robot configuration profile.
3-1-3 ACOS on page 3-9	Real-valued function that returns the arc cosine of its argument.
3-1-4 ALIGN on page 3-10	Transformation function that aligns the input location with the nearest axis of the world coordinate system.
3-1-5 ASC on page 3-11	Real-valued function that returns an ASCII character value from within a string.
3-1-6 ASIN on page 3-12	Real-valued function that returns the arc sine of its argument.
3-1-7 ATAN2 on page 3-13	Real-valued function that returns the size of the angle in degrees that has its trigonometric tangent equal to value_ 1/value_2.
3-1-8 BASE on page 3-14	Transformation function that returns the transformation value that represents the translation and rotation set by the last BASE program command or monitor command.
3-1-9 BASE.TRANS on page 3-15	Transformation function that returns the transformation value that represents the translation and rotation set by the last BASE command or instruction.
3-1-10 BCD on page 3-15	Real-valued function that converts a real value to Binary Coded Decimal (BCD) format.
3-1-11 BELT on page 3-16	Real-valued function that returns information about a conveyor belt being tracked with the conveyor tracking feature.
3-1-12 BITS on page 3-17	Real-valued function that reads multiple digital signals and returns the value corresponding to the binary bit pattern.
3-1-13 BMASK on page 3-18	Real-valued function that creates a bit mask by setting individual bits.
3-1-14 BSTATUS on page 3-20	Real-valued function that returns information about the status of the conveyor tracking system.
3-1-15 CAS on page 3-21	Real-valued function that compares a real variable to a test value and conditionally sets a new value as one indivisible operation.
3-1-16 $CHR on page 3-22	String function that returns a one-character string corresponding to a given ASCII value.
3-1-17 COM on page 3-23	Perform the binary complement operation on a value.
3-1-18 CONFIG on page 3-24	Real-valued function that returns a value providing information about the robot's geometric configuration or the status of the motion servo control features.

Keyword & Page Number	Description
3-1-19 COS on page 3-27	Real-valued function that returns the trigonometric cosine of a given angle.
3-1-20 CUBRT on page 3-28	Real-valued function that returns the cube root of a value.
3-1-21 $DBLB on page 3-29	String function that returns an 8-byte string containing the binary representation of a real value in double-precision IEEE 754 floating-point format.
3-1-22 DBLB on page 3-30	Real-valued function that returns the value of eight bytes of a string interpreted as an IEEE double-precision floating-point number.
3-1-23 DCB on page 3-32	Real-valued function that converts BCD digits into an equivalent integer value.
3-1-24 $DECODE on page 3-33	String function that extracts part of a string as delimited by given break characters.
3-1-25 $DEFAULT on page 3-35	String function that returns a string containing the current or initial system default device, unit, and directory path for disk file access.
3-1-26 DEFINED on page 3-36	Real-valued function that determines if a variable has been defined.
3-1-27 DEST on page 3-37	Transformation function that returns a transformation value representing the planned destination location for the current robot motion.
3-1-28 DEVICE on page 3-39	Real-valued function that returns a real value from a specified device. The value may be data or status information, depending upon the device and the parameters.
3-1-29 DISTANCE on page 3-40	Real-valued function that determines the distance between the points defined by two location values.
3-1-30 DURATION on page 3-41	Real-valued function that returns the current setting for one of the motion DURATION specifications.
3-1-31 DX on page 3-42	Real-valued function that returns the X-axis component of a given transformation value.
3-1-32 DY on page 3-43	Real-valued function that returns the Y-axis component of a given transformation value.
3-1-33 DZ on page 3-44	Real-valued function that returns the Z-axis component of a given transformation value.
3-1-34 ENCLATCH on page 3-44	Real-valued function that returns the encoder position (mm) for any encoder in the system at the occurrence of the last latch.
3-1-35 $ENCODE on page 3-45	String function that returns a string created from output specifications.
3-1-36 $ERROR on page 3-47	String function that returns the error message associated with the given error code.
3-1-37 ERROR on page 3-48	Real-valued function that returns the message number of a recent system message that caused program execution to stop or caused a REACTE operation.
3-1-38 FALSE on page 3-51	Real-valued function that returns the value used by V+ to represent a logical false result.
3-1-39 FB.ERROR on page 3-52	Real-valued function that returns the most recent fieldbus error code.

Keyword & Page Number	Description
3-1-40 FB.STATE on page 3-58	Real-valued function that returns the state of the field-bus.
3-1-41 $FLTB on page 3-59	String function that returns a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.
3-1-42 FLTB on page 3-60	Real-valued function that returns the value of four bytes of a string interpreted as an IEEE single-precision floating- point number.
3-1-43 FRACT on page 3-62	Real-valued function that returns the fractional part of the argument.
3-1-44 FRAME on page 3-63	Transformation function that returns a transformation value defined by four positions.
3-1-45 FREE on page 3-64	Real-valued function that returns the amount of unused free memory of storage space.
3-1-46 GETC on page 3-65	Real-valued function that returns the next character (byte) from a device or input record on the specified logical unit.
3-1-47 GET.EVENT on page 3-67	Real-valued function that return events that are set for the specified task.
3-1-48 HERE on page 3-68	Transformation function that returns a transformation value that represents the current location of the robot tool point.
3-1-49 HOUR.METER on page 3-68	Real-valued function that returns the current value of the robot hour meter.
3-1-50 $ID on page 3-69	String function that returns the system ID string.
3-1-51 ID on page 3-70	Real-valued function that returns values that identify the configuration of the current system.
3-1-52 IDENTICAL on page 3-75	Real-valued function that determines whether two location values are exactly the same.
3-1-53 INRANGE on page 3-76	Real-valued function that returns a value that indicates if a specific location can be reached by the robot and provides additional information when a location cannot be reached.
3-1-55 $INTB on page 3-79	String function that returns a 2-byte string containing the binary representation of a 16-bit integer.
3-1-54 INT on page 3-78	Real-valued function that returns the integer part of the value.
3-1-56 INTB on page 3-80	Real-valued function that returns the value of two bytes of a string interpreted as a signed 16-bit binary integer.
3-1-57 INVERSE on page 3-82	Transformation function that returns the transformation value that is the mathematical inverse of the given transformation value.
3-1-58 IOSTAT on page 3-83	Real-valued function that returns status information for the last input / output operation for a device associated with a logical unit.
3-1-59 LAST on page 3-86	Real-valued function that returns the highest index used for an array (dimension).
3-1-60 LATCH on page 3-87	Transformation function that returns a transformation value representing the location of the robot at the occurrence of the last external trigger.

Keyword & Page Number	Description
3-1-61 LATCHED on page 3-88	Real-valued function that returns the status of the position latch and which input triggered it.
3-1-62 LEN on page 3-89	Real-valued function that returns the number of characters in the given string.
3-1-63 $LNGB on page 3-89	String function that returns a 4-byte string containing the binary representation of a 32-bit integer.
3-1-64 LNGB on page 3-91	Real-valued function that returns the value of four bytes of a string interpreted as a signed 32-bit binary integer.
3-1-65 MAX on page 3-92	Real-valued function that returns the maximum value contained in the list of values.
3-1-66 $MID on page 3-93	String function that returns a substring of the specified string.
3-1-67 MIN on page 3-94	Real-valued function that returns the minimum value contained in the list of values.
3-1-68 NETWORK on page 3-95	Real-valued function that returns network status and IP address information of the robot controller.
3-1-69 NORMAL on page 3-96	Transformation function that corrects a transformation for any mathematical round-off errors.
3-1-70 NOT on page 3-96	Operator that performs logical negation of a value.
3-1-71 NULL on page 3-97	Transformation function that returns a null transformation value(one with all zero components).
3-1-72 OFF on page 3-98	Real-valued function that returns the value used by V+ to represent a logical false result.
3-1-73 ON on page 3-98	Real-valued function that returns the value used by V+ to represent a logical true result.
3-1-74 OUTSIDE on page 3-99	Real-valued function that tests a value to determine if it is outside a specified range.
3-1-75 PARAMETER on page 3-100	Real-valued function that returns the current setting of the named system parameter.
3-1-76 #PDEST on page 3-101	Precision-point function that returns a precision-point value representing the planned destination location for the cur- rent robot motion.
3-1-77 #PHERE on page 3-101	Precision-point function that returns a precision-point value representing the current location of the currently selected robot.
3-1-78 PI on page 3-102	Real-valued function that returns the value of the mathematical constant pi (3.141593).
3-1-79 #PLATCH on page 3-102	Precision-point function that returns a precision-point value representing the location of the robot at the occurrence of the last external trigger.
3-1-80 POS on page 3-103	Real-valued function that returns the starting character position of a substring in a string.
3-1-81 #PPOINT on page 3-104	Precision-point function that returns a precision-point value composed from the given components.
3-1-82 PRIORITY on page 3-105	Real-valued function that returns the current reaction lock-out priority for the program.
3-1-83 RANDOM on page 3-106	Real-valued f unction that returns a pseudo-random number.
3-1-84 RX on page 3-107	Transformation function that returns a transformation describing a rotation about the X-axis.

Keyword & Page Number	Description
3-1-85 RY on page 3-107	Transformation function that returns a transformation describing a rotation about the Y-axis.
3-1-86 RZ on page 3-108	Transformation function that returns a transformation describing a rotation about the Z-axis.
3-1-87 SCALE on page 3-109	Transformation function that returns a transformation value equal to the transformation parameter with the position scaled by the scale factor.
3-1-88 SELECT on page 3-110	Real-valued function that returns information about the device specified for the currently selected task.
3-1-89 #SETPOINT on page 3-111	Precision point function that returns the commanded joint-angle positions computed by the trajectory generator during the last trajectory-evaluation cycle.
3-1-90 SHIFT on page 3-112	Transformation function that returns a transformation value resulting from shifting the position of the transformation parameter by the given shift amounts.
3-1-91 SIG.INS on page 3-113	Real-valued function that returns an indication of whether a digital/O signal is installed in the system or whether a software signal is available in the system.
3-1-92 SIGN on page 3-114	Real-valued f unction that returns the value 1, with the sign of the value parameter.
3-1-93 SIG on page 3-115	Real-valued function that returns the logical AND of the states of the indicated digital signals.
3-1-94 SIN on page 3-116	Real-valued function that returns the trigonometric sine of a given angle.
3-1-95 SOLVE.FLAGS on page 3-117	Real-valuedf unction that returns bit flags representing the robot configuration specified by an array of joint positions.
3-1-96 SPEED on page 3-118	Real-valued function that returns one of the system motion speed factors.
3-1-97 SQRT on page 3-120	Real-valued function that returns the square root of the parameter.
3-1-98 SQR on page 3-120	Real-valued function that returns the square of the parameter.
3-1-99 STATE on page 3-121	Real-valued function that returns a value to provide information about the robot system state.
3-1-100 STATUS on page 3-127	Real-valued function that returns status information for an application program.
3-1-101 STRDIF on page 3-128	Real-valued function that compares two strings byte-by-byte for the purpose of sorting.
3-1-102 SWITCH on page 3-129	Real-valued function that returns information about the setting of a system switch.
3-1-103 TAN on page 3-130	Real-valued function that returns the trigonometric tangent of a given angle.
3-1-104 TAS on page 3-131	Real-valued function that returns the current value of a real-valued variable and assigns it a new value. The two actions are done indivisibly so that no other program task can modify the variable at the same time.
3-1-105 TASK on page 3-134	Real-valued function that returns information about a program execution task.

Keyword & Page Number	Description
3-1-106 $TIME on page 3-136	String function that returns a string value containing either the current system date and time or the specified date and time.
3-1-107 $TIME4 on page 3-137	String function that returns a string value containing either the current system four-digit date and time or the specified four-digit date and time.
3-1-108 TIME on page 3-139	Real-valued function that returns an integer value representing either the date or the time specified in the given string parameter.
3-1-109 TIMER on page 3-141	Real-valued function that returns the current time value of the specified system timer.
3-1-110 TOOL on page 3-143	Transformation function that returns the value of the transformation specified in the last TOOL operation.
3-1-111 TPS on page 3-144	Real-valued function that returns the number of ticks of the system clock that occur per second (Ticks Per Second).
3-1-112 TRANS on page 3-144	Transformation function that returns a transformation value computed from the given X, Y, Z position displacements and y, p, r orientation rotations.
3-1-113 $TRANSB on page 3-146	String function that returns a 48-byte string containing the binary representation of a transformation value.
3-1-114 TRANSB on page 3-147	Transformation function that returns a transformation value represented by a 48-byte or 96-byte string.
3-1-115 TRUE on page 3-149	Real-valued function that returns the value used by V+ to represent a logical true result.
3-1-116 $TRUNCATE on page 3-149	String function that returns all characters in the input string until an ASCII NUL (or the end of the string) is encountered.
3-1-117 $UINTB on page 3-150	String function that returns a two-byte string containing the binary representation of a 16-bit unsigned integer.
3-1-118 UINTB on page 3-151	Real-valued function that returns the value of a two-byte string interpreted as a 16-bit unsigned integer.
3-1-119 $ULNGB on page 3-153	String function that returns a 4-byte string containing the binary representation of a 32-bit unsigned integer.
3-1-120 ULNGB on page 3-154	Real-valued function that returns the value of four bytes of a string interpreted as an unsigned 32-bit binary integer.
3-1-121 $UNPACK on page 3-155	String function that returns a substring from an array of 128-character string variables.
3-1-122 VAL on page 3-156	Real-valued function that returns the real value represented by the characters in the input string.
3-1-123 VLOCATION on page 3-157	Transformation function that returns a cartesian transformation result of the execution of the specified vision sequence.The returned value is a transform result as x, y, z, yaw, pitch, and roll.
3-1-124 VPARAMETER on page 3-160	Transformation function that returns the current value of a vision tool parameter.
3-1-125 VRESULT on page 3-161	Real-valued function that returns a specified result of a vision tool, or returns the status of a specified tool.
3-1-126 VSTATE on page 3-163	Real-valued function that returns the state of the execution of a sequence.

Keyword & Page Number	Description
3-1-127 WINDOW on page 3-164	Real-valued function that returns a value to indicate 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.

2-2 Monitor Command Keyword Summary

The table below provides a summary of all monitor command keywords.

Keyword & Page Number	Description
3-2-1 ABORT on page 3-167	Monitor command that terminates execution of an executable program.
3-2-2 BASE on page 3-168	Monitor command that translates and rotates the world reference frame relative to the robot.
3-2-3 BASE.TRANS on page 3-170	Monitor command that translates and rotates the reference frame relative to the robot base.
3-2-4 BITS on page 3-171	Monitor command that sets or clears a group of digital signals based on a value.
3-2-5 CALIBRATE on page 3-172	Monitor command that initializes the robot positioning system.
3-2-6 CD on page 3-174	Monitor command that displays or changes the default path for disk access.
3-2-7 COMMANDS on page 3-175	Monitor command that initiates processing of a Monitor Command program.
3-2-8 COPY on page 3-176	Monitor command that creates a new program as a copy of an existing program.
3-2-9 CYCLE.END on page 3-177	Monitor command that terminates the specified executable program the next time it executes a STOP operation or its equivalent. It will suspend processing of a command program until a program completes execution.
3-2-10 DEFUALT on page 3-179	Monitor command that defines the default relationship between the V+ disk logical device and the physical device to be accessed. This also displays the current default.
3-2-11 DELETE on page 3-182	Monitor command that deletes the specified programs from the system memory.
3-2-12 DELETEL on page 3-183	Monitor command that deletes the named location variables from the system memory.
3-2-13 DELETEM on page 3-184	Monitor command that deletes the named program module from the system memory.
3-2-14 DELETEP on page 3-185	Monitor command that deletes the named programs from the system memory.
3-2-15 DELETER on page 3-186	Monitor command that deletes the named real-valued variables from the system memory.
3-2-16 DELETES on page 3-188	Monitor command that deletes the named string variables from the system memory.
3-2-17 DIRECTORY on page 3-189	Monitor command that displays the names of some or all of the programs in the system memory.
3-2-18 DISABLE on page 3-190	Monitor command that turns OFF one or more system switches.

Keyword & Page Number	Description
3-2-19 DO on page 3-191	Monitor command that executes a keyword(s) as though it were the next step in an executable program or the next step in the specified task / program context.
3-2-20 ENABLE on page 3-193	Monitor command that turns ON one or more system switches.
3-2-21 ESTOP on page 3-194	Monitor command that stops the robot in the same manner as if an emergency stop signal was received.
3-2-22 EXECUTE on page 3-195	Monitor command that begins execution of a control program.
3-2-23 FCOPY on page 3-197	Monitor command that copies the information in an existing disk file to a new disk file.
3-2-24 FDELETE on page 3-199	Monitor command that deletes one or more disk files matching the given file specification.
3-2-25 FDIRECTORY on page 3-200	Monitor command that displays information about the files on a disk and the amount of space remaining for storage as well as creates and delete subdirectories on disks.
3-2-26 FLIST on page 3-203	Monitor command that lists the contents of the specified disk file on the Monitor Window.
3-2-27 FREE on page 3-204	Monitor command that displays the percentage of available system memory not currently in use.
3-2-28 FRENAME on page 3-205	Monitor command that changes the name of a disk file.
3-2-29 FSET on page 3-206	Monitor command that sets or modifies attributes of a network device.
3-2-30 HERE on page 3-207	Monitor command that defines the value of a transformation or precision-point variable to be equal to the current robot location.
3-2-31 ID on page 3-209	Monitor command that displays identity information about components of the system.
3-2-32 IO on page 3-211	Monitor command that displays the current states of external digital input / output signals or internal software signals.
3-2-33 JOG on page 3-212	Monitor command that moves the specified joint of the robot, or moves the robot tool along the specified Cartesian axis. Each time JOG is executed, the robot moves for up to 300 ms.
3-2-34 KILL on page 3-215	Monitor command that clears a program execution stack and detaches any I/O devices that are attached.
3-2-35 LIST on page 3-216	Monitor command that displays the value of the expression.
3-2-36 LISTL on page 3-217	Monitor command that displays the values of the listed locations.
3-2-37 LISTP on page 3-218	Monitor command that displays all the steps of the listed user programs.
3-2-38 LISTR on page 3-219	Monitor command that displays the values of the real expressions specified.
3-2-39 LISTS on page 3-220	Monitor command that displays the values of the specified strings.

Keyword & Page Number	Description
3-2-40 LOAD on page 3-222	Monitor command that loads the contents of the specified disk file into the system memory.
3-2-41 MDIRECTORY on page 3-224	Monitor command that displays the names of all the program modules in the system memory or the names of the programs in a specified program module.
3-2-42 MODULE on page 3-225	Monitor command that creates a new program module, or modifies the contents of an existing module.
3-2-43 NET on page 3-226	Monitor command that displays status information about the network. Also displays details about the remote mounts that are currently defined in the V+ system.
3-2-44 PANIC on page 3-228	Monitor command that simulates an external E-stop button press, stops all robots immediately but does not turn OFF robot high power.
3-2-45 PARAMETER on page 3-229	Monitor command that sets or displays the values of system parameters.
3-2-46 PING on page 3-231	Monitor command that tests the network connection to a node.
3-2-47 PRIME on page 3-232	Monitor command that prepares a program for execution but does not start execution.
3-2-48 PROCEED on page 3-233	Monitor command that resumes execution of an application program.
3-2-49 RENAME on page 3-234	Monitor command that changes the name of a user program in memory to the new name provided.
3-2-50 RESET on page 3-235	Monitor command that will turn OFF all the digital output signals.
3-2-51 RESET.LOCK on page 3-236	Monitor command that detaches a robot from the application program.
3-2-52 RETRY on page 3-236	Monitor command that repeats execution of the last interrupted statement and continues execution of the program.
3-2-53 SELECT on page 3-237	Monitor command that selects a robot for subsequent Monitor Window operations.
3-2-54 SIGNAL on page 3-238	Monitor command that turns ON or OFF digital output signals, internal software signals, or host signals.
3-2-55 SPEED on page 3-240	Monitor command that specifies monitor speed.
3-2-56 SSTEP on page 3-241	Monitor command that executes a single step or an entire subroutine of a control program.
3-2-57 STACK on page 3-242	Monitor command that specifies the amount of system memory reserved for a program task to use for subroutine calls and automatic variables.
3-2-58 STATUS on page 3-244	Monitor command that returns status information for the system and the programs being executed.
3-2-59 STORE on page 3-247	Monitor command that stores programs and variables in a disk file.
3-2-60 STOREL on page 3-249	Monitor command that stores location variables in a disk file.
3-2-61 STOREM on page 3-250	Monitor command that stores a specified program module to a disk file.
3-2-62 STOREP on page 3-252	Monitor command that stores program files to a disk file.

Keyword & Page Number	Description
3-2-63 STORER on page 3-253	Monitor command that stores real variables in a disk file.
3-2-64 STORES on page 3-254	Monitor command that stores a string variable in a disk file.
3-2-65 SWITCH on page 3-256	Monitor command that displays the settings of system switches in the Monitor Window.
3-2-66 TESTP on page 3-257	Monitor command that tests for the presence of the named program in the system memory.
3-2-67 TOOL on page 3-258	Monitor command that sets the internal transformation used to represent the location and orientation of the tool tip relative to the tool-mounting flange of the robot.
3-2-68 WAIT.START on page 3-259	Monitor command that puts a Monitor Command program into a wait state until a condition is satisfied.
3-2-69 WHERE on page 3-260	Monitor command that displays the current location of the robot and the hand opening.
3-2-70 XSTEP on page 3-261	Monitor command that executes a single step of a program.
3-2-71 ZERO on page 3-263	Monitor command that initializes the V+ system and deletes all the programs and data in system memory.

2-3 Other Keyword Summary

The table below provides a summary of all keywords not categorized as function, monitor command, program command, system parameter, or system switch keywords.

Keyword & Page Number	Description
3-3-1 .END on page 3-265	Keyword that marks the end of an V+ program.
3-3-2 IPS on page 3-265	Specify the units for a SPEED program command as inches per second.
3-3-3 MMPS on page 3-266	Specify the units for a SPEED program command as millimeters per second.

2-4 Program Command Keyword Summary

The table below provides a summary of all program command keywords.

Keyword & Page Number	Description
3-4-1 ABORT on page 3-268	Terminate execution of an executing program task.
3-4-2 ABOVE on page 3-269	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.
3-4-3 ACCEL on page 3-270	Set acceleration and deceleration for robot motions and optionally specify a defined acceleration profile.
3-4-4 ALIGN on page 3-272	Align the robot tool Z-axis with the nearest world axis.
3-4-5 ALTER on page 3-273	Specify the magnitude of the real-time path modification that is to be applied to the robot path during the next trajectory computation.
3-4-6 ALTOFF on page 3-274	Terminate real-time path-modification mode (alter mode).
3-4-7 ALTON on page 3-275	Enable real-time path-modification mode (alter mode) and specify the way in which alter coordinate information will be interpreted.
3-4-8 ANY on page 3-277	Signal the beginning of an alternative group of commands for the CASE structure.
3-4-9 APPRO on page 3-277	Start a robot motion toward a location defined relative to specified location with joint-interpolated motion.
3-4-10 APPROS on page 3-278	Start a robot motion toward a location defined relative to specified location with straight-line motion.
3-4-11 ATTACH on page 3-279	Make a device available for use by the application program.
3-4-12 AUTO on page 3-283	Declare temporary variables that are automatically created on the program stack when the program is entered.
3-4-13 BASE on page 3-286	Translate and rotate the world reference frame relative to the robot.
3-4-14 BASE.TRANS on page 3-287	Program command that translates and rotates the reference frame relative to the robot base.
3-4-15 BELOW on page 3-289	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.
3-4-16 BITS on page 3-290	Set or clear a group of digital signals based on a value.
3-4-17 BRAKE on page 3-291	Abort the current robot motion.
3-4-18 BREAK on page 3-292	Suspend program execution until the current motion completes.
3-4-19 BY on page 3-293	Completes the syntax of the SCALE and SHIFT functions.
3-4-20 CALIBRATE on page 3-294	Initialize the robot positioning system with the robot's current position.
3-4-21 CALL on page 3-297	Suspend execution of the current program and continue execution with a new subroutine program.

Keyword & Page Number	Description
3-4-22 CALLS on page 3-300	Suspend execution of the current program and continue execution with a new subroutine program specified with a string value.
3-4-23 CASE on page 3-301	Initiate processing of a CASE structure by defining the value of interest.
3-4-24 CLEAR.EVENT on page 3-303	Clear an event associated with the specified task.
3-4-25 CLEAR.LATCHES on page 3-304	Empties the latch buffer for the selected device.
3-4-26 COARSE on page 3-305	Enable a low-precision nulling tolerance for the robot.
3-4-27 COPY.ARRAY on page 3-307	Copy elements of the source array to the destination array.
3-4-28 CPOFF on page 3-308	Instruct the V+ system to stop the robot at the completion of the next motion operation (or all subsequent motion operations) and null position errors.
3-4-29 CPON on page 3-309	Instruct the V+ system to execute the next motion operations(or all subsequent motion operations) as part of a continuous path.
3-4-30 CYCLE.END on page 3-310	Terminate the executing program in the specified task the next time it executes a STOP program command (or its equivalent).
3-4-31 DECOMPOSE on page 3-312	Extract the real values of individual components of a location value.
3-4-32 DEF.DIO on page 3-313	Assign virtual digital I/O to standard V+ signal numbers for use by keywords.
3-4-33 DEFBELT on page 3-314	Define a belt variable for use with a conveyor tracking robot.
3-4-34 DEPART on page 3-316	Start a robot motion away from the current location with joint-interpolated motion.
3-4-35 DEPARTS on page 3-317	Start a robot motion away from the current location with straight-line motion.
3-4-36 DETACH on page 3-318	Release a specified device from the control of the application program.
3-4-37 DISABLE on page 3-320	Turn OFF one or more system switches.
3-4-38 DO on page 3-321	Introduce a DO program structure.
3-4-39 DOS on page 3-322	Execute a keyword defined by a string expression.
3-4-40 DRIVE on page 3-324	Execute a keyword defined by a string expression.
3-4-41 DURATION on page 3-325	Set the minimum execution time for subsequent robot motions.
3-4-42 ELSE on page 3-326	Separate the alternate group of statements in an IF ... THEN control structure.
3-4-43 ENABLE on page 3-327	Turn ON one or more system switches.
3-4-44 END on page 3-328	Mark the end of a control structure.
3-4-45 ESTOP on page 3-330	Stop the robot in the same manner as if an emergency-stop signal was received.
3-4-46 EXECUTE on page 3-331	Begin execution of a control program.
3-4-47 EXIT on page 3-334	Branch to the statement following the nth nested loop of a control structure.
3-4-48 EXTERNAL on page 3-335	Declare a variable that is shared between V+ and a host controller.
3-4-49 FCLOSE on page 3-337	Close the disk file currently open on the specified logical unit.

Keyword & Page Number	Description
3-4-50 FCMND on page 3-338	Generate a device-specific command to the input / out- put device specified by the logical unit.
3-4-51 FCOPY on page 3-341	Copy the information in an existing disk file to a new disk file.
3-4-52 FDELETE on page 3-342	Delete the specified disk file.
3-4-53 FEMPTY on page 3-343	Empty any internal buffers in use for a disk file by writing the buffers to the file if necessary.
3-4-54 FINE on page 3-344	Enable a high-precision nulling tolerance for the robot.
3-4-55 FLIP on page 3-346	Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a negative value.
3-4-56 FOPEN on page 3-348	Create and open a new TCP connection.
3-4-57 FOPENA on page 3-349	Opens a file for read-write-append access. If the specified file does not already exist, the file is created.
3-4-58 FOPEND on page 3-352	Opens a disk directory for reading.
3-4-59 FOPENR on page 3-354	Opens a file for read-only access.
3-4-60 FOPENW on page 3-356	Opens a file for read-write access. If the file already exists, an error occurs.
3-4-61 FOR on page 3-358	Execute a program loop a specified number of times.
3-4-62 FSEEK on page 3-360	Position a file open for random access and initiate a read operation on the specified record.
3-4-63 FSET on page 3-361	Set or modify attributes of a network device.
3-4-64 GLOBAL on page 3-362	Declare a variable to be global and specify the type of the variable.
3-4-65 GOTO on page 3-364	Perform an unconditional branch to the program step identified by the given label.
3-4-66 HALT on page 3-365	Stop program execution and do not allow the program to be resumed.
3-4-67 HERE on page 3-366	Set the value of a transformation or precision-point variable equal to the current robot location.
3-4-68 IF...GOTO on page 3-367	Branch to the specified step label if the value of the logical expression is TRUE (non-zero).
3-4-69 IF...THEN on page 3-368	Conditionally execute a group of keywords (or one of two groups) depending on the result of a logical expression.
3-4-70 IGNORE on page 3-369	Cancel the effect of a REACT or REACTI program command.
3-4-71 JMOVE on page 3-370	Moves all robot joints to positions described by a list of joint values. The robot performs a coordinated motion in joint-interpolated mode.
3-4-72 JOG on page 3-371	Jogs the specified joint of the robot or moves the robot tool along the specified cartesian direction.
3-4-73 KEYMODE on page 3-373	Set the behavior of a group of keys on the pendant.
3-4-74 KILL on page 3-375	Clear a program execution stack and detach any I/O devices that are attached.
3-4-75 LEFTY on page 3-376	Request a change in the robot configuration during the next motion to make the first two links of a SCARA robot use the left arm orientation.
3-4-76 LOCAL on page 3-377	Declare permanent variables that are defined only within the current program.

Keyword & Page Number	Description
3-4-77 LOCK on page 3-379	Set the program reaction lock-out priority to the value given.
3-4-78 MC on page 3-380	Introduce a monitor command within a Monitor Command program.
3-4-79 MCS on page 3-381	Invoke a monitor command from an application program.
3-4-81 MOVE on page 3-384	Initiate a robot motion to the position and orientation described by the given location with joint-interpolated motion.
3-4-82 MOVEC on page 3-385	Initiate a circular / arc-path robot motion using the positions and orientations described by the given locations.
3-4-83 MOVES on page 3-390	Initiate a robot motion to the position and orientation described by the given location with straight-line motion.
3-4-80 MULTIPLE on page 3-383	Allow full rotations of the robot wrist joints.
3-4-84 NEXT on page 3-391	Branch to the END statement of the nth nested loop, perform the loop test, and loop if appropriate.
3-4-85 NOFLIP on page 3-392	Request a change in the robot configuration during the next motion so the pitch angle of the robot wrist has a positive value.
3-4-86 NONULL on page 3-393	Instruct the V+ system to not wait for position errors to be nulled at the end of continuous-path motions.
3-4-87 NOOVERLAP on page 3-394	Instruct the V+ system to not wait for position errors to be nulled at the end of continuous-path motions.
3-4-88 NULL on page 3-396	Instruct the V+ system to wait for position errors to be nulled at the end of continuous path motions.
3-4-89 OVERLAP on page 3-397	Disable the NOOVERLAP limit-error checking either for the next motion or for all subsequent motions.
3-4-90 PACK on page 3-398	Replace a substring within an array of (128-character) string variables, or within a (non-array) string variable.
3-4-91 PANIC on page 3-400	Simulate an external E-Stop button press to stop all robots immediately, but do not turn off high power.
3-4-92 PARAMETER on page 3-400	Set the value of a system parameter.
3-4-93 PAUSE on page 3-401	Stop program execution but allow the program to be resumed.
3-4-94 PAYLOAD on page 3-402	Adjust the dynamic compensation for the selected robot based on the payload's physical properties.
3-4-95 PDNT.CLEAR on page 3-403	Clears the current notification window or custom message window on the T20 pendant, if any, and returns the T20 pendant back to the Home screen.
3-4-96 PDNT.NOTIFY on page 3-404	Creates a pendant notification.
3-4-97 PDNT.WRITE on page 3-405	Sets the pendant's Custom Message screen.
3-4-98 PENDANT on page 3-406	Return input from the manual control pendant.
3-4-100 PROCEED on page 3-410	Resume execution of an application program.
3-4-101 .PROGRAM on page 3-411	Define the arguments that are passed to a program when it is invoked.
3-4-102 PROMPT on page 3-413	Display a string on the Monitor Window and wait for operator input.

Keyword & Page Number	Description
3-4-103 REACT on page 3-415	Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal transitions.
3-4-104 REACTE on page 3-417	Initiate the monitoring of system messages that occur during execution of the current program task.
3-4-105 REACTI on page 3-419	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.
3-4-106 READ on page 3-421	Read a record from an open file or from an attached device that is not file oriented. For an network device, read a string from an attached and open TCP connection.
3-4-107 READY on page 3-423	Move the robot to the ready location.
3-4-108 RELEASE on page 3-424	Allow the next available program task to run.
3-4-109 RESET on page 3-425	Turn OFF all external output signals.
3-4-110 RETRY on page 3-426	Repeat execution of the last interrupted program command and continue execution of the program.
3-4-111 RETURN on page 3-427	Terminate execution of the current subroutine and resume execution of the suspended program at its next step.
3-4-112 RETURNE on page 3-428	Terminate execution of an error reaction subroutine and resume execution of the last-suspended program at the step following the statement that caused the subroutine to be invoked.
3-4-113 RIGHTY on page 3-428	Request a change in the robot configuration during the next motion to make the first two links of a SCARA robot use the right arm orientation.
3-4-114 RUNSIG on page 3-429	Turn ON or OFF the specified digital signal as long as execution of the invoking program task continues.
3-4-115 SELECT on page 3-431	Select a unit of the named device for access by the current task.
3-4-116 SET.EVENT on page 3-432	Set an event associated with the specified task.
3-4-117 SET on page 3-433	SET
3-4-118 SETBELT on page 3-434	Set the encoder offset of the specified belt variable equal to the value of the expression.
3-4-119 SETDEVICE on page 3-435	Initialize a device or set device parameters. The operation performed depends on the device referenced.
3-4-120 SIGNAL on page 3-437	Turn ON or OFF external digital output signals or internal software signals.
3-4-121 SINGLE on page 3-438	Limit rotations of the robot wrist joint to the range -180 degrees to +180 degrees.
3-4-122 SOLVE.ANGLES on page 3-439	Compute the robot joint positions for the current robot that are equivalent to a specified transformation.
3-4-123 SOLVE.TRANS on page 3-444	Compute the transformation equivalent to a given set of joint positions for the current robot.
3-4-124 SPEED on page 3-445	Set the nominal speed for subsequent robot motions.
3-4-125 STOP on page 3-448	Terminate execution of the current program cycle.
3-4-126 SWITCH on page 3-449	Enable or disable a system switch based on a value.
3-4-127 TIMER on page 3-450	Set the specified system timer to the given time value.

Keyword & Page Number	Description
3-4-128 TOOL on page 3-451	Set the internal transformation used to represent the location and orientation of the tool tip relative to the tool mounting flange of the robot.
3-4-129 TRANS.TO.POSE on page 3-452	Returns an array determined by translation and rotation in the extrinsic Euler angles X, Y, and Z that is based on a transformation.
3-4-130 TYPE on page 3-453	Display the information described by the output specifications on the Monitor Window.
3-4-131 UNTIL on page 3-455	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.
3-4-132 VALUE on page 3-456	Indicate the values that a CASE statement expression must match in order for the program statements immediately following to be executed.
3-4-133 VPARAMETER on page 3-457	Sets the current value of a vision tool parameter.
3-4-134 VRUN on page 3-458	Initiates the execution of a vision sequence.
3-4-135 VWAITI on page 3-459	Waits until the specified vision sequence reaches the state specified by the type parameter.
3-4-136 WAIT on page 3-460	Put the program into a wait loop for one trajectory cycle. If a condition is specified, wait until the condition is TRUE.
3-4-137 WAIT.EVENT on page 3-461	Suspend program execution until a specified event has occurred or until a specified amount of time has elapsed.
3-4-138 WHILE on page 3-463	Initiate processing of a WHILE structure if the condition is TRUE or skipping of the WHILE structure if the condition is initially FALSE.
3-4-139 WINDOW on page 3-464	Set the boundaries of the operating region of the specified belt variable for conveyor tracking.
3-4-140 WRITE on page 3-466	Write a record to an open file or to any I/O device. For network device, write a string to an attached device and open a TCP connection.

2-5 System Parameter Keyword Summary

The table below provides a summary of all system parameter keywords.

Keyword & Page Number	Description
3-5-1 BELT.MODE on page 3-468	System parameter that sets the characteristics of the conveyor tracking feature of the V+ system.
3-5-2 DEVIATION on page 3-470	Adds a path deviation from 1 to 100% to the motion in the singularity region when a robot is in singularity.
3-5-3 JOG.TIME on page 3-470	System parameter that sets the keep-alive time of a jog operation.
3-5-4 NOT.CALIBRATED on page 3-471	System parameter that indicates or asserts the calibration status of the robots connected to the system.
3-5-5 VTIMEOUT on page 3-473	System parameter that sets a timeout value so that an error message is returned if no response is received following a vision command.

2-6 System Switch Keyword Summary

The table below provides a summary of all system switch keywords.

Keyword & Paragraph Number	Description
3-6-1 AUTO.POWER.OFF on page 3-474	This system switch disables high power when certain motion errors occur.
3-6-2 CP on page 3-475	System switch that controls the continuous-path function of a robot.
3-6-3 DECEL.100 on page 3-476	System Switch that enables or disables the maximum deceleration of 100% for the ACCEL program command keyword.
3-6-4 DELAY.IN.TOL on page 3-476	Systems witch that controls the timing of coarse or fine nulling after the V+ system completes a motion segment.
3-6-5 DRY.RUN on page 3-478	System switch that controls whether or not V+ communicates with the robot.
3-6-6 MESSAGES on page 3-479	System Switch to enable or disable output to the Monitor Window from TYPE Program Commands.
3-6-7 OBSTACLE on page 3-480	System Switch that enables or disables up to four obstacles for a selected robot.
3-6-8 POWER on page 3-481	System Switch that controls or monitors the status of the robot high power.
3-6-8 POWER on page 3-481	System switch that enables or disables one robot or all robots.
3-6-10 SCALE.ACCEL on page 3-484	System switch that enables or disables the scaling of acceleration and deceleration as a function of program speed.
3-6-11 SCALE.ACCEL.ROT on page 3-486	System switch that specifies whether or not the SCALE.ACCEL switch takes into account the Cartesian rotational speed during straight-line motions.
3-6-12 UPPER on page 3-487	System switch that controls whether or not the case of each character is ignored when string comparisons are performed.

3

Keyword Details

This section provides details for all V+ keywords.

3-1 Function Keywords

Use the information in this section to understand function keywords and their use with the V+ system.

3-1-1 ABS

Real-valued function that returns the absolute value (magnitude) of the argument provided.

Syntax

ABS (value)

Parameter

Parameter	Description
value	Real-valued expression.

Examples

The following example returns 0.123.

```
ABS(0.123)
```

The following example returns 5.462

```
ABS(-5.462)
```

The following example returns 0.013125.

```
ABS(1.3125E-2)
```

The following example divides the variable "part.size" by the absolute value of the variable "belt.scale" and returns the result to the variable "belt.length".

```
belt.length = part.size/ABS(belt.scale)
```

3-1-2 ACCEL

Real-valued function that returns the current setting for robot acceleration, deceleration, or the maximum allowable percentage limits defined in the robot configuration profile.

Syntax

ACCEL (value)

Usage Considerations

The ACCEL function returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Parameter

Parameter	Description
value	Real-valued expression, the result of which is rounded to an integer to select the value that is returned. Setting the select parameter to the following values will return the specified information. • 0: Number of selected acceleration profile. • 1: Acceleration • 2: Deceleration • 3: Maximum allowable acceleration percentage • 4: Maximum allowable deceleration percentage • 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

The following example will return the current acceleration setting.

```
ACCEL(1)
```

The following example will return the current deceleration setting.

```
ACCEL(2)
```

Related Keywords

3-4-3 ACCEL on page 3-270 (program command)
3-6-10 SCALE.ACCEL on page 3-484
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-115 SELECT on page 3-431 (program command)

3-1-3 ACOS

Real-valued function that returns the arc cosine of its argument.

Syntax

```
ACOS (value)
```

Usage Considerations

The value parameter must be in the range of -1.0 to +1.0. Any value outside this range will cause an illegal value error.

Parameter

Parameter	Description
value	Real-valued expression that defines the cosine value to be considered.

Details

Returns the inverse cosine (arc cosine) of the argument, which is assumed to be in the range of -1.0 to +1.0. The resulting value is always in the range of 0.0 to +180.0, inclusive.

Examples

The following example returns a value of 90.

```
ACOS(0)
```

The following example returns a value of 90

```
ACOS(-1)
```

The following example returns a value of 84.2608295.

```
ACOS(0.1)
```

The following example returns a value of 60

```
ACOS(0.5)
```

Additional Information

TYPE, PROMPT, and other similar commands 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.

Related Keywords

3-1-19 COS on page 3-27
3-1-94 SIN on page 3-116
3-1-6 ASIN on page 3-12
3-1-103 TAN on page 3-130
3-1-7 ATAN2 on page 3-13

3-1-4 ALIGN

Transformation function that aligns the input location with the nearest axis of the world coordinate system.

Syntax

```
ALIGN (location)
```

Parameter

Parameter	Description
location	Transformation value to be used as a reference.

Details

Returns a modified version of the input location that is aligned parallel to the nearest axis of the World coordinate system.

Examples

The following example aligns the position of a robot to a location defined as "point1" and sets it to the value of the location variable "align.loc".

```
SET align.loc= ALIGN(point1)
```

Related Keywords

3-4-4 ALIGN on page 3-272 (program command)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-1-5 ASC

Real-valued function that returns an ASCII character value from within a string.

Syntax

ASC(string, index**)**

Parameter

Parameter	Description
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 specified character is returned as a real value.

Examples

The following example returns the ASCII value of the letter "a".

```
ASC("sample", 2)
```

The following example returns the ASCII value of the first character of the string contained in the variable "$name".

```
ASC($name)
```

The following example uses the value of the real variable "i" as an index to the character of interest in the string contained in the variable "$system".

```
ASC($system, i)
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-122 VAL on page 3-156

3-1-6 ASIN

Real-valued function that returns the arc sine of its argument.

Syntax

ASIN (value)

Usage Considerations

The value parameter must be in the range of -1.0 to +1.0. Any value outside this range will cause an illegal value error.

Parameter

Parameter	Description
value	Real-valued expression.

Details

Returns the inverse sine (arcsine) of the argument, which is assumed to be in the range of -1.0 to +1.0. The resulting value is always in the range of -90.0 to +90.0, inclusive.

Examples

The following example returns a value of 0.

```
ASIN(0)
```

The following example returns a value of -90.

```
ASIN(-1)
```

The following example returns a value of 5.73917047.

```
ASIN(0.1)
```

The following example returns a value of 30.

```
ASIN(0.5)
```

Additional Information

TYPE, PROMPT, and similar commands 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.

Related Keywords

3-1-19 COS on page 3-27
3-1-3 ACOS on page 3-9
3-1-94 SIN on page 3-116
3-1-103 TAN on page 3-130
3-1-7 ATAN2 on page 3-13

3-1-7 ATAN2

Real-valued function that returns the size of the angle in degrees that has its trigonometric tangent equal to value_1/value_2.

Syntax

ATAN2 (value_1, value_2)

Usage Considerations

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

Parameter

Parameter	Description
value_1	Real-valued expression.
value_2	Real-valued expression.

Examples

The following example returns 26.1067.

```
ATAN2(0.123,0.251)
```

The following example returns -6.600926.

```
ATAN2(-5.462,47.2)
```

The following example returns -90.56748.

```
ATAN2(1.3125E+2,-1.3)
```

The following example returns and angle value to the "slope" variable from trigonometric tangent equal to "rise"/"run".

```
slope = ATAN2(rise, run)
```

Additional Information

TYPE, PROMPT, and similar commands 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.

Related Keywords

3-1-19 COS on page 3-27
3-1-3 ACOS on page 3-9
3-1-94 SIN on page 3-116
3-1-6 ASIN on page 3-12
3-1-103 TAN on page 3-130

3-1-8 BASE

Transformation function that returns the transformation value that represents the translation and rotation set by the last BASE program command or monitor command.

Syntax

BASE

Usage Considerations

The BASE function returns information for the robot selected by the task executing the function. If the task executing this function does not have a robot selected, the output of this function is invalid.
The statement LISTL BASE can be used to display the current base setting.

Examples

The following example sets the new base location using the BASE program command and then moves to the new base location.

```
BASE 100,100
MOVE BASE
```

Related Keywords

3-2-2 BASE on page 3-168 (monitor command)
3-4-13 BASE on page 3-286 (program command)
3-2-36 LISTL on page 3-217

3-1-88 SELECT on page 3-110 (real-valued function)
3-4-115 SELECT on page 3-431 (program command)

3-1-9 BASE.TRANS

Transformation function that returns the translation and rotation values from the last BASE monitor command or BASE program command.

Syntax

```
BASE.TRANS
```

Usage Considerations

The BASE.TRANS function returns information for the robot selected by the task executing the function.
The statement LISTL BASE can be used to display the current base setting.

Additional Information

If the task executing this function does not have a robot selected, the output of this function is invalid.

Related Keywords

3-2-2 BASE on page 3-168 (monitor command)
3-1-8 BASE on page 3-14 (transformation function)
3-4-13 BASE on page 3-286 (program command)
3-4-14 BASE.TRANS on page 3-287 (program command)
3-2-3 BASE.TRANS on page 3-170 (monitor command)
3-2-53 SELECT on page 3-237 (monitor command)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-1-10 BCD

Real-valued function that converts a real value to Binary Coded Decimal (BCD) format.

Syntax

```
BCD (value)
```

Usage Considerations

The BCD function is most useful when used in conjunction with the BITS keyword.

Parameter

Parameter	Description
value	Real-valued expression 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 for example.

Examples

The following example converts the real variable "digit" to a BCD value and applies it to external digital output signals 4 through 8.

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

Related Keywords

3-1-23 DCB on page 3-32
3-4-16 BITS on page 3-290 (program command)
3-1-12 BITS on page 3-17 (real-valued function)
3-2-4 BITS on page 3-171 (monitor command)

3-1-11 BELT

Real-valued function that returns information about a conveyor belt being tracked with the conveyor tracking feature.

Syntax

```
BELT (%belt_var, mode)
```

Usage Considerations

The BELT system switch must be enabled before this function can be used.
The SETBELT program command is generally used in conjunction with the BELT function to set the effective belt position to zero. This must be done each time the robot will perform a sequence of motions relative to the belt and must be done shortly before the first motion of such a sequence.

⚠ WARNING

It is important to execute the SETBELT program command each time the robot is going to track the belt to make sure the difference between the current belt position as returned by the BELT function and the belt position of the specified belt variable does not exceed 8,388,607 (^H7FFFFF) during active belt tracking. Unpredictable robot motion may result if the difference does exceed this value while tracking the belt.

Parameter

Parameter	Description
%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, its value is equal to zero, or < -1, the BELT function returns the encoder reading in encoder counts of the belt specified by the belt variable. The value returned by this function is limited to an absolute value of 8,388,607 and will roll over to -8,388,608 after. If the value is equal to -1, the BELT function returns the last latched encoder position in encoder counts of the belt specified by the belt variable. This value equivalent to the value returned by DEVICE(0, enc, stt, 4) except it is not bounded to 8,388,607. If the value of the expression is greater than zero, the encoder velocity is returned in units of encoder counts per V+ cycle. For Standard Control systems, the V+ cycle time is approximately 4 ms. For Robot Integrated Control systems, the cycle time is approximately 2 or 4 ms.

Examples

The following example will 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)
```

The following example will save the current speed of the belt associated with the belt variable "%b".

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

Related Keywords

3-4-33 DEFBELT on page 3-314
3-1-28 DEVICE on page 3-39
3-4-118 SETBELT on page 3-434

3-1-12 BITS

Real-valued function that reads multiple digital signals and returns the value corresponding to the binary bit pattern.

Syntax

```
BITS (first_sig, num_sigs)
```

Usage Considerations

External digital input or output signals or internal software signals can be referenced.
A maximum of 32 signals can be read at one time.
Any group of up to 32 signals can be read provided that all the signals in the group are configured for use by the system.

Parameter

Parameter	Description
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 32.

Details

This function returns a value that corresponds to the binary bit pattern present on 1 to 32 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.

Examples

The example below assumes that the following input signal states are present.

```
Signal: 1008    1007    1006    1005    1004    1003    1002    1001
State:    1    1    0    1    0    1    1    0
```

The following statement will return a value of 5 for variable "x" because the four signals 1003, 1004, 1005, and 1006 can be interpreted as a binary representation of that value.

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

Related Keywords

3-4-16 BITS on page 3-290 (program command)
3-2-32 IO on page 3-211
3-2-50 RESET on page 3-235
3-1-93 SIG on page 3-115
3-1-91 SIG.INS on page 3-113
3-2-54 SIGNAL on page 3-238 (monitor command)
3-4-120 SIGNAL on page 3-437 (program command)

3-1-13 BMASK

Real-valued function that creates a bit mask by setting individual bits.

Syntax

```
BMASK (bit, bit, ..., bit)
```

Usage Considerations

A bit is considered ON when it has a non-zero value (typically the bit number for clarity).

Parameter

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

Details

This function creates a bit mask by turning ON the specified bits and leaving all other bits OFF. A bit is specified when it is assigned a number, any non-zero value is interpreted as the bit being ON.
Bit 32 is the sign bit and yields a negative number when set.

Examples

The following example compresses multiple boolean states into a single 8-bit byte. Each of the eight bit[i] variables corresponds to one of the 8 bits in the resulting byte from bit 1 (LSB) to bit 8 (MSB). In this example, these are selectively turned ON depending on specific simulated runtime conditions, such as emergency stop state, robot power, robot calibration, and a user-defined flag (app.started). The BMASK keyword encodes the bit states into a single byte (byte_val). After encoding, the result (byte_val) contains the compressed bit values.
The TYPE keyword is used to display the results of the example in the Monitor Window. The displayed results are converted to 0 or 1 using $ENCODE in reverse order (bit 8 to bit 1), followed by the resulting byte value.

```
AUTO REAL b[8]

TYPE $CHR(27)+"[H"+$CHR(27)+"[J"

app.started = TRUE

FOR i = 1 TO 8
        bit[i] = 0
END

IF STATE(4) BAND ^b100 THEN
    bit[1] = 1
END
IF SWITCH(POWER) THEN
```

```
    bit[2] = 2
END
IF NOT PARAMETER(NOT.CALIBRATED) THEN
    bit[3] = 3 ; Robot is calibrated
END
IF app.started THEN
    bit[7] = 7 ; Application has started
END

byte_val = BMASK(bit[1], bit[2], bit[3], bit[4], bit[5], bit[6], bit[7], bit[8])

FOR i = 1 TO 8
    b[i] = ABS(bit[i] <> 0)
END

TYPE $ENCODE("Encoding bits ", /I0, b[8], b[7], b[6], b[5], b[4], b[3], b[2], b[1])
, /S

TYPE " into 'byte_val' -->", byte_val
TYPE /X21, "^LSB"
```

3-1-14 BSTATUS

Real-valued function that returns information about the status of the conveyor tracking system.

Syntax

BSTATUS

Usage Considerations

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

Details

This function is normally used when the BELT.MODE system parameter bit 4 is set.
This function returns a value that is equivalent to the binary value represented by a set of bit flags that indicate the following conditions of the conveyor tracking software.

Bit Flag	Condition
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 Flag	Condition
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

3-1-11 BELT on page 3-16 (real-valued function)
3-2-45 PARAMETER on page 3-229 (monitor command)
3-4-92 PARAMETER on page 3-400 (Program command)
3-1-75 PARAMETER on page 3-100 (real-valued function)
3-4-139 WINDOW on page 3-464 (program command)
3-1-127 WINDOW on page 3-164 (real-valued function)

3-1-15 CAS

Real-valued function that compares a real variable to a test value and conditionally sets a new value as one indivisible operation.

Syntax

```
CAS (variable, test_value, new_value)
```

Usage Considerations

The V+ system does not enforce any protection scheme for global variables that are shared by multiple program tasks. It is the programmer's responsibility to keep track of the usage of such global variables. The CAS real-valued function (or the similar TAS function) can be used to implement logical interlocks on access to shared variables.
This function can also be used to bypass a restriction on the simultaneous access of global arrays by multiple program tasks. Program execution can fail if two or more tasks attempt to increase the size of an array at the same time. Refer to *V+ User's Manual (Cat. No. I671)* for more information about global array access restriction.

Parameter

Parameters	Description
variable	Name of the real-valued variable to be tested and assigned the new value given.
test_value	Real value, variable, or expression that defines the comparison value.

Parameters	Description
new_value	Real value, variable, or expression that defines the new value to be assigned to the specified variable.

Details

If the variable is equal to the test value, the new value is stored in the variable. Otherwise the variable is not modified. The original value of the variable is returned as the function value.
The compare and set-new-value operations occur with interrupts locked so that the operation is indivisible. This function provides a way for setting semaphores between tasks, similar to the TAS real-valued function. Refer to TAS for more information.
If the variable is undefined when the function is executed, it is treated as having the value zero.

Related Keywords

3-1-104 TAS on page 3-131

3-1-16 $CHR

String function that returns a one-character string corresponding to a given ASCII value.

Syntax

```
$CHR(value)
```

Parameter

Parameter	Description
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.

Examples

The following example returns the character "A" (ASCII value 65).

```
$CHR(65)
```

Related Keywords

3-1-5 ASC on page 3-11
3-1-21 $DBLB on page 3-29
3-1-41 $FLTB on page 3-59
3-1-55 $INTB on page 3-79

3-1-63 $LNGB on page 3-89

3-1-17 COM

Perform the binary complement operation on a value.

Syntax

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

Usage Considerations

The COM function is meaningful only when performed on an integer value. Only the integer parts of real values are used. Any fractional parts are ignored.

Parameter

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

Details

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

To review the order of evaluation for operators within expressions, refer to *V+ User's Manual (Cat. No. I671)*.

Examples

The following example returns the value of -41.

```
COM 40
```

Related Keywords

3-1-70 NOT on page 3-96

3-1-18 CONFIG

Real-valued function that returns a value providing information about the robot's geometric configuration or the status of the motion servo control features.

Syntax

```
CONFIG (select)
```

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

Parameter	Description
select	Optional real value, variable, or expression interpreted as an integer that has a value from 0 to 13 and selects the category of the configuration information to be returned.

Details

This function returns a value that is interpreted as a series of bit flags. The interpretation of the value returned by this function 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. Each bit in the value represents one characteristic of a robot configuration.
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. Each bit in the value represents the state of one motion servo control feature.
When the select parameter is 6, 7, or 8, the CONFIG function returns a value that represents the setting of the FINE tolerance. Limitations to this functionality apply when using FINE with the ALWAYS parameter. Refer to *3-4-54 FINE* on page 3-344 for more information.
When the select parameter is 9, 10, or 11, the CONFIG function returns a value that represents the setting of the COARSE tolerance. Limitations to this functionality apply when using COARSE with the ALWAYS parameter. Refer to *3-4-26 COARSE* on page 3-305 for more information.
The parameter values in this group determine which robot configuration is returned by the function.

- **select = 0**

  This returns the robot's current configuration. The default value is 0.

- **select = 1**

  This returns the configuration the robot will achieve at the completion of the current motion or the current configuration if no motion is in progress (while the robot is attached).

● select = 2

This returns the configuration the robot achieves at the completion of the next motion assuming that it is a joint-interpolated and not straight-line motion.
The interpretations of the bit flags returned when Select = 1, 2, or 3 are shown in the table below.

Bit #	Bit Mask	Indication if Bit ON
1	1	Robot has righty configuration.
2	2	Robot has below configuration.
3	4	Robot has flipped configuration.

● select = 3

This returns the permanent settings of the robot motion servo control features that are defined by keywords that specify the ALWAYS qualifier.

● select = 4

This returns the temporary settings for the motion currently executing or the last motion completed if no motion is in progress.

● select = 5

This returns 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 shown in the table below.

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

● select = 6

This returns the FINE tolerance as the permanent setting as a percentage of the standard tolerance.

● select = 7

This returns FINE tolerance as the setting used for the previous or current motion as a percentage of the standard tolerance.

● select = 8

This returns FINE tolerance as the setting to be used for the next motion as a percentage of the standard tolerance.

● select = 9

This returns COARSE tolerance as the permanent setting as a percentage of the standard tolerance.

● select = 10

This returns COARSE tolerance as the setting used for the previous or current motion as a percentage of the standard tolerance.

● select = 11

This returns the COARSE tolerance as the setting to be used for the next motion as a percentage of the standard tolerance.

● select = 12

When select = 12, the available joint configuration options for the selected robot are returned as shown in the table 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 no flip configuration.
18	^H20000	Robot supports the OVERLAP and NOOVERLAP keywords.
22	^H20000	Robot's last rotary joint can be limited to ±180 degrees with SINGLE program commands.

● select = 13

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

Bit#	Bit Mask	Description
1	1	This bit is set if the motion is joint interpolated. It is cleared for straight-line motion.

Examples

The following example will check the if the robot is moving to a righty or lefty configuration. Text will be displayed in the Monitor Window to indicate the configuration.

```
IF (CONFIG(1)==0) OR (CONFIG(1)==4 THEN
TYPE "Robot is moving to a lefty position."
END
IF (CONFIG(1)==1) OR (CONFIG(1)==5) THEN
TYPE "Robot is moving to a righty position."
END
```

Related Keywords

3-4-2 ABOVE on page 3-269

3-4-15 BELOW on page 3-289
3-4-26 COARSE on page 3-305
3-4-28 CPOFF on page 3-308
3-4-29 CPON on page 3-309
3-4-54 FINE on page 3-344
3-4-55 FLIP on page 3-346
3-4-75 LEFTY on page 3-376
3-4-85 NOFLIP on page 3-392
3-4-86 NONULL on page 3-393
3-4-87 NOOVERLAP on page 3-394
3-1-71 NULL on page 3-97
3-4-89 OVERLAP on page 3-397
3-4-113 RIGHTY on page 3-428
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-121 SINGLE on page 3-438
3-1-99 STATE on page 3-121

3-1-19 COS

Real-valued function that returns the trigonometric cosine of a given angle.

Syntax

```
COS(angle)
```

Usage Considerations

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

Parameter

Parameter	Description
angle	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 is always in the range of -1.0 to +1.0, inclusive.

Examples

The following example returns a value of 0.999962.

```
COS(0.5)
```

The following example returns a value of 0.9954596.

```
COS(-5.462)
```

The following example returns a value of 0.4999999.

```
COS(60)
```

The following example returns a value of -0.659345.

```
COS(1.3125E+2)
```

Additional Information

TYPE, PROMPT, and similar commands 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.

Related Keywords

3-1-3 ACOS on page 3-9
3-1-94 SIN on page 3-116
3-1-6 ASIN on page 3-12
3-1-103 TAN on page 3-130
3-1-7 ATAN2 on page 3-13

3-1-20 CUBRT

Real-valued function that returns the cube root of a value.

Syntax

CUBRT(value)

Parameter

Parameter	Description
value	Real-valued expression defining the value whose cube root is to be computed.

Examples

The following example returns a value of 0.497319.

```
CUBRT(0.123)
```

The following example returns a value of 2.0.

```
CUBRT(8)
```

The following example returns a value of -1.7611.

```
CUBRT(-5.462)
```

The following example returns a value of 5.081982.

```
CUBRT(1.3125E+2)
```

Additional Information

TYPE, PROMPT, and similar commands 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.

Related Keywords

3-1-98 SQR on page 3-120
3-1-97 SQRT on page 3-120

3-1-21 $DBLB

String function that returns an 8-byte string containing the binary representation of a real value in double-precision IEEE 754 floating-point format.

Syntax

```
$DBLB(value, littleEndian)
```

Parameter

Parameter	Description
value	Real-valued expression which is converted to its IEEE 754 floating-point binary representation.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

The primary use of this function is to convert a double-precision real value to its binary representation in an output record of a data file.
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. Refer to the real-valued function *3-1-22 DBLB* on page 3-30 for a more detailed description of IEEE 754 floating-point format.

Examples

This example demonstrates how to convert a V+ numeric variable (out_double) into its binary representation using the IEEE 754 double-precision floating-point format (Float64). The result is a sequence of eight ASCII bytes stored in a V+ string variable ($s).
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
TYPE $CHR(27)+"[H"+$CHR(27)+"[J"
out_double = PI
TYPE "Variable 'out_double': ", /F20.18, out_double
```

```
little.endian = 1
$s = $DBLB(out_double, little.endian)

TYPE "Variable 'out_double' converted to 8 bytes:      ", /S
TYPE $ENCODE(/H2, ASC($s,1), "-", ASC($s,2), "-", ASC($s,3), "-", ASC($s,4), "-"),
/S

TYPE $ENCODE(/H2, ASC($s,5), "-", ASC($s,6), "-", ASC($s,7), "-", ASC($s,8)), /C1
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-42 FLTB on page 3-60
3-1-41 $FLTB on page 3-59
3-1-55 $INTB on page 3-79
3-1-113 $TRANSB on page 3-146

3-1-22 DBLB

Real-valued function that returns the value of eight bytes of a string interpreted as an IEEE double-precision floating-point number.

Syntax

DBLB ($string, first_char, litteEndian**)**

Parameter

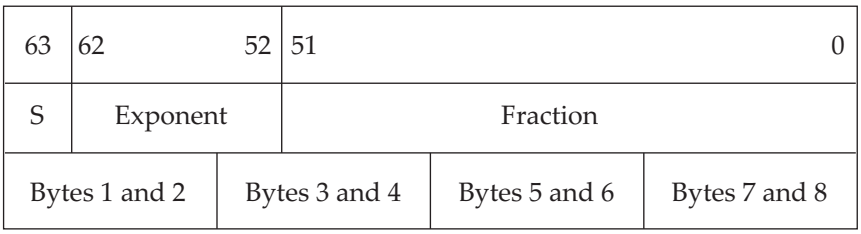
Parameter	Description
$string	String expression that contains the eight bytes to be converted.
first_char	Optional real-valued expression that specifies the position of the first of the eight bytes in the string. If first_char is omitted or has a value of 0 or 1, the first eight bytes of the string are 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. If first_char specifies eight bytes that are beyond the end of the input string, an error is generated.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

The primary 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+.

● Floating-point Numbers

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.



Item	Description
S	Sign bit (0 = positive, 1 = negative)
Exponent	Binary exponent, biased by -1023
Fraction	Binary fraction with an implied 1 to the left of the binary point

The value of a floating point number is shown below for 0 < exponent < 2047.

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

Double-precision real values have the following special values.

Exponent	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 x 10-308 to 1.8 x 10307

Examples

This example demonstrates how to convert a V+ numeric variable (out_double) into its binary representation using the IEEE 754 double-precision floating-point format (Float64). The result is a sequence of eight ASCII bytes stored in a V+ string variable ($s). The string variable ($s) is converted to a V+ numeric variable (in_double) using the DBLB keyword to reconstruct the original floating-point number.

The TYPE keyword is used to display the results of the example in the Monitor Window.

```
TYPE $CHR(27)+"[H"+$CHR(27)+"[J"
out_double = PI
TYPE "Variable 'out_double': ", /F20.18, out_double

little.endian = 1
$s = $DBLB(out_double, little.endian)
```

```
TYPE "Variable 'out_double' converted to 8 bytes:      ", /S
TYPE $ENCODE(/H2, ASC($s,1), "-", ASC($s,2), "-", ASC($s,3), "-", ASC($s,4), "-"),
/S

TYPE $ENCODE(/H2, ASC($s,5), "-", ASC($s,6), "-", ASC($s,7), "-", ASC($s,8)), /C1

TYPE "Simulate receiving 8 bytes:      ", /S
TYPE $ENCODE(/H2, ASC($s,1), "-", ASC($s,2), "-", ASC($s,3), "-", ASC($s,4), "-"),
/S

TYPE $ENCODE(/H2, ASC($s,5), "-", ASC($s,6), "-", ASC($s,7), "-", ASC($s,8))

in_double = DBLB($s, , little.endian)

TYPE "Variable 'in_double' interpreted from 8 bytes:   ", /F20.18, in_double
```

Related Keywords

3-1-5 ASC on page 3-11
3-1-21 $DBLB on page 3-29
3-1-42 FLTB on page 3-60
3-1-41 $FLTB on page 3-59
3-1-54 INT on page 3-78
3-1-114 TRANSB on page 3-147
3-1-122 VAL on page 3-156

3-1-23 DCB

Real-valued function that converts BCD digits into an equivalent integer value.

Syntax

DCB(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

Parameter	Description
value	Real value interpreted as a binary bit pattern representing up to four BCD digits.*1

*1. An error is reported if any digit is not a valid BCD digit (if a digit is greater than 9).

Examples

The following example sets the real variable "input" equal to the integer equivalent of the BCD input on the specified signals if external input signals 1001- 1008 (8 bits of input) receive two BCD digits from an external device.

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

Related Keywords

3-1-12 BITS on page 3-17
3-1-10 BCD on page 3-15

3-1-24 $DECODE

String function that extracts part of a string as delimited by given break characters.

Syntax

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

Usage Considerations

$DECODE modifies the input $string_var variable as well as returning a string value.
The evaluation 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

Parameter	Description
$string_var[*1]	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.
$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.

Parameter	Description
mode	Optional real value, variable, or expression that controls the operation performed by the function. Mode values are -3, -2, 0, and 1. If the mode is negative, 0, 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 0, characters up to the first non-break character are removed from the input string and returned as the output of the function. This case returns all the leading break characters in the input string.

*1. 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 contains the value returned by the function.

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 all break characters (mode 1). $DECODE can return and delete all the characters up to the first break character for a 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 extracted from the input string (see the first example below).

The modes -2 and -3 copy all non-break characters up to the first break characters plus the first break character.

Mode-2 is equivalent to the following statements.

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

The following statement has the same functionality.

```
$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 examples below extract consecutive numbers from the string "$input", assuming that the numbers are separated by some combination of spaces and commas.

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

In a case where "$string_var" contains a sequence of numeric values as strings separated by spaces, commas, or any combination of spaces and commas, executing the following statement results in the first four elements of the "value" array getting the values shown below.

```
$string_var = "1234. 93465.2, .4358,3458103"
value[0] = 1234.0
value[1] = 93465.2
value[2] = 0.4358
value[3] = 3458103.0
```

The string variable input ($string_var) also contains an empty string ("").

As shown above, use of the $DECODE function normally involves two string variables as 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 example, the same variable is used on both sides of the equal sign because the programmer wants to discard all the white space (spaces and tabs) characters at the end of the input string.

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

3-1-116 $TRUNCATE on page 3-149
3-1-66 $MID on page 3-93

3-1-25 $DEFAULT

String function that returns a string containing the current or initial system default device, unit, and directory path for disk file access.

Syntax

```
$DEFAULT (mode, empty_arg)
```

Usage Considerations

Parentheses must be included even when mode is omitted.

Parameter

Parameter	Description
mode	Optional real value, variable, or expression (interpreted as an integer) that specifies the default path to be returned. If the parameter is omitted, or has the value 0, the current system default path is returned. If the parameter has the value 1, the default path returned is the one that was in effect immediately after the V+ system was booted from disk.
empty_arg	This should be NULL

Details

The system default device, unit, and directory path can be set by the CD or DEFAULT command. The $DEFAULT function returns the current or initial default values as a string as shown below.

```
device>disk_unit:directory_path
```

This string contains the portions of the following information that have been set.

Device	Description
DISK	A local disk.
SYSTEM	A disk device, drive, and subdirectory path currently set with the DEFAULT monitor command.*1

*1. Refer to the *3-4-11 ATTACH* on page 3-279 command for more information on valid devices.

Parameter	Description
disk_unit	The disk unit specified to the DEFAULT command. The colon (:) is omitted if no unit was specified.
directory_path	Any input to the DEFAULT command that followed the device and unit. The directory path is omitted if no additional input was specified.

Examples

The following example sets the default drive specification to DISK>D:\TEST\ and then displays it on the terminal for confirmation.

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

Related Keywords

3-2-6 CD on page 3-174
3-2-10 DEFUALT on page 3-179

3-1-26 DEFINED

Real-valued function that determines if a variable has been defined.

Syntax

```
DEFINED (var_name)
```

Parameter

Parameter	Description
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.
If a specific array element is specified for array variables, the single array element is tested. If no array element is specified, this function returns a TRUE value if any element of the array is defined.
For non-real 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. You cannot assign a value to undefined non-real arguments within the called program. If you attempt to assign a value to an undefined non-real argument, an undefined value error message is returned.
When using DEFINED to test for user input, assign a default value to the variable before testing it as shown in the examples below.

Examples

The following statement returns a value of TRUE if the variable "base_part" is defined.

```
DEFINED(base_part)
```

The following statement returns a value of TRUE if any element of the array "corner" has been defined.

```
DEFINED(corner[])
```

Related Keywords

3-1-100 STATUS on page 3-127
3-2-66 TESTP on page 3-257

3-1-27 DEST

Transformation function that returns a transformation value representing the planned destination location for the current robot motion.

Syntax

```
DEST
```

Usage Considerations

The DEST function returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Details

DEST returns the location to which a robot was moving when its motion was interrupted. This applies to motion keywords with the following considerations.

- Motions to named locations.

```
MOVE start MOVES #part[10]
```

- 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)
```

- Motions to special locations.

```
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 an emergency stop operation occurs, the robot stops immediately, but the DEST function still returns the location to which the robot was moving.
If a motion is not started because V+ detects the destination location cannot be reached, then the DEST function is not set to the goal location.

Examples

The following example continues a motion that has been interrupted by a reaction initiated by a REACTI keyword. The subroutine automatically invoked can contain steps such as the following to process the interruption and resume the original motion.

```
SET save = HERE
SET old.dest = DEST old.speed = SPEED(3) DEPART 50.0
.
.
.
APPRO save, 50.0
MOVES save SPEED old.speed MOVES old.dest
```

Related Keywords

3-1-48 HERE on page 3-68
3-1-76 #PDEST on page 3-101
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-115 SELECT on page 3-431 (program command)

3-1-28 DEVICE

Real-valued function that returns a real value from a specified device. The value may be data or status information, depending upon the device and the para- meters.

Syntax

```
DEVICE (type, unit, error, p1, p2,...)
```

Usage Considerations

The syntax contains optional parameters that may be useful only for specific device types and information requests.

Parameter

Keyword	Description
type	Real value, variable, or expression interpreted as an integer that indicates the type of device being referenced. The following types are available. • 0 = Belt encoder. • 1 = (Reserved for future use)) • 2 = Force Processor Board • 3 = Robot device • 4 = Vision • 5 = (Reserved for future use)
unit	Real value that indicates the device unit number. The value must be in the range 0 to -1, where -1 is the maximum number of devices of the specified type. The value should be 0 if 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 the request. The number of values specified and the meanings of the values depend upon the particular device type.

Details

DEVICE is a function for returning data and status information from external devices.

Examples

The following example captures the current belt encoder position when the last parameter =1 and captures the next latched encoder position from the buffer when the last parameter = 4.

```
IF (sv.emulate.mode) THEN latch.num = 1
latch.value = DEVICE(0,pick.encoder[0]-1,sts,1)
```

```
ELSE
latch.value = DEVICE(0,pick.encoder[0]-1,sts,4)
END
SETBELT %pick.belt[0] = latch.value
```

Related Keywords

3-4-119 SETDEVICE on page 3-435

3-1-29 DISTANCE

Real-valued function that determines the distance between the points defined by two location values.

Syntax

DISTANCE (location_1, location_2)

Usage Considerations

Only X, Y, and Z axes are considered.
Either location can be expressed as a compound transformation.

Parameter

Keyword	Description
location_1	Transformation value that defines the first point of interest. This can be a function, a variable, or a compound transformation.
location_2	Transformation value that defines the second point 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.

Examples

The following example sets the value of the real variable x to be the distance between where the robot tool point is currently located and the point defined by the transformation part.

```
x = DISTANCE(HERE, part)
```

Related Keywords

3-1-52 IDENTICAL on page 3-75

3-1-30 DURATION

Real-valued function that returns the current setting for one of the motion DURATION specifications.

Syntax

```
DURATION (select)
```

Usage Considerations

The DURATION function returns information for the robot selected by the task executing the function. If the task executing this function does not have a robot selected, the output of this function is invalid.

Parameter

Keyword	Description
select	Real-valued expression whose value determines which duration value should be returned.

Details

This function returns the user-specified minimum robot motion duration in seconds corresponding to the select parameter value. Refer to the *3-4-41 DURATION* on page 3-325 program command for an explanation of the specification of motion duration times.
Different select values determine when the duration time returned applies as described below. The acceptable range for the select parameter is 2 to 4. Other values for this parameter are invalid.

Select	DURATION value returned
2	Permanent minimum robot motion duration (set by a DURATION ...ALWAYS statement)
3	Temporary motion duration for the current or last motion.
4	Temporary motion duration to be used for the next motion.

Examples

The following example will check if the DURATION setting is 2 using the real- valued function and if not, will set it to 2 using the program command.

```
SELECT ROBOT = 1
IF DURATION(2)<>2 THEN
```

```
DURATION 2 ALWAYS
END
```

Related Keywords

3-1-18 CONFIG on page 3-24
3-1-30 DURATION on page 3-41 (program command)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-1-31 DX

Real-valued function that returns the X-axis component of a given trans- formation value.

Syntax

DX (location)

Parameter

Keyword	Description
location	Transformation value from which a component is desired. This can be a function, a variable, or a compound transformation.

Details

This function returns the respective X-axis component of the specified transformation value.
The DECOMPOSE command can also be used to obtain the displacement components of a transformation value. If the rotation components are desired, the DECOMPOSE command must be used. DECOMPOSE is more efficient if more than one element is needed and the location is a compound transformation.

Examples

Consider the transformation "start" with the following components.

```
125, 250, -50, 135, 50, 75
```

The following example will return a value of 125.00.

```
DX(start)
```

Related Keywords

3-4-31 DECOMPOSE on page 3-312
3-1-32 DY on page 3-43
3-1-33 DZ on page 3-44

3-1-84 RX on page 3-107
3-1-85 RY on page 3-107
3-1-86 RZ on page 3-108

3-1-32 DY

Real-valued function that returns the Y-axis component of a given transformation value.

Syntax

```
DY (location)
```

Parameter

Keyword	Description
location	Transformation value from which a component is desired. This can be a function, a variable, or a compound trans- formation.

Details

This function returns the respective Y-axis component of the specified transformation value.
The DECOMPOSE command can also be used to obtain the displacement components of a transformation value. If the rotation components are desired, the DECOMPOSE command must be used. DECOMPOSE is more efficient if more than one element is needed and the location is a compound transformation.

Examples

Consider the transformation "start" with the following components.

```
125, 250, -50, 135, 50, 75
```

The following example will return a value of 250.00.

```
DY(start)
```

Related Keywords

3-4-31 DECOMPOSE on page 3-312
3-1-31 DX on page 3-42
3-1-33 DZ on page 3-44
3-1-84 RX on page 3-107
3-1-85 RY on page 3-107
3-1-86 RZ on page 3-108

3-1-33 DZ

Real-valued function that returns the Z-axis component of a given transformation value.

Syntax

```
DZ (location)
```

Parameter

Keyword	Description
location	Transformation value from which a component is desired. This can be a function, a variable, or a compound transformation.

Details

This function returns the respective Z-axis component of the specified transformation value.
The DECOMPOSE command can also be used to obtain the displacement components of a transformation value. If the rotation components are desired, the DECOMPOSE command must be used. DECOMPOSE is more efficient if more than one element is needed and the location is a compound transformation.

Examples

Consider the transformation "start" with the following components.

```
125, 250, -50, 135, 50, 75
```

The following example will return a value of -50.00.

```
DZ(start)
```

Related Keywords

3-4-31 DECOMPOSE on page 3-312
3-1-31 DX on page 3-42
3-1-32 DY on page 3-43
3-1-84 RX on page 3-107
3-1-85 RY on page 3-107
3-1-86 RZ on page 3-108

3-1-34 ENCLATCH

Real-valued function that returns the encoder position (mm) for any encoder in the system at the occurrence of the last latch.

Syntax

```
ENCLATCH(select)
```

Usage Considerations

If an encoder that is not configured is specified for the select parameter, executing the ENCLATCH keyword will return an error.

Parameter

Keyword	Description
select	Optional integer, expression, or real variable specifying the encoder id. This can be a value from 1 to 116.

Details

The ENCLATCH keyword returns a real-value that represents the location in millimeters of the encoder when the last trigger occurred. The LATCHED keyword should be used to determine when trigger has occurred and a valid location has been recorded. The DEVICE keyword may be also used to read the latched value of an encoder as well as other information of the encoder.
Operation of the external trigger can be configured with the Sysmac Studio or ACE software. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Examples

The following example returns the encoder position of encoder 1 and assigns the value to "Latch_pos".

```
encoder_id = 1
Latch_sig = LATCHED(-1*encid)
Latch_pos = ENCLATCH(encid)
```

Related Keywords

3-1-28 DEVICE on page 3-39
3-1-61 LATCHED on page 3-88

3-1-35 $ENCODE

String function that returns a string created from output specifications.

Syntax

```
$ENCODE (output_specification, output_specification, ...)
```

Parameter

An output specification can consist of any of the following components in any order, separated by commas.

- A string expression.
- A real-valued expression that is evaluated to determine a value to be displayed.
- Format control specifier that determines the format of the output message.

Details

This function makes strings normally produced by the TYPE command available within a program. The $ENCODE function creates a string value but does not generate any output.
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 (*).

Specifier	Format
/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 0 as the width field. This causes the actual field size to vary to fit the value and all leading spaces to be suppressed. This is useful when a value is displayed within a line of text or at the end of a line.

Specifier	Format
/En.m	Format values in scientific notation (for example, -1.234E+02) in fields n spaces wide with m digits in the fractional parts. If n is not zero, it must be large enough to include space for a minus sign (if the displayed value is negative), one digit to the left of the decimal point, a decimal point (if m is not zero), m digits, and four or five characters for the exponent.
/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 /En.m format is used.
/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.

Specifier	Format
/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 Monitor Window, this results in n blank lines if the format control specifier is at the beginning or end of the function parameter list. Otherwise, n-1 blank lines result.

Specifier	Format
/Un	Include the characters necessary to move the cursor up n lines if the resulting string is output to the Monitor Window..
/Xn	Include n spaces.

Examples

The following example adds a formatted representation of the value of count to the string contained in $output.

```
$output = $output+$ENCODE(/F6.2, count)
```

The following example provides a method of adding a format control specifier to the output from a PROMPT command. This example displays the value of the motor as part of the prompt message.

```
PROMPT $ENCODE("Start motor #",/I0,motor," (Y/N)? "), $answer
```

This PROMPT command displays the following user prompt when the value of motor is 3.

```
Start motor #3 (Y/N)?
```

Related Keywords

3-4-102 PROMPT on page 3-413
3-4-130 TYPE on page 3-453

3-1-36 $ERROR

String function that returns the error message associated with the given error code.

Syntax

$ERROR (error_code)

Parameter

Keyword	Description
error_code	Real-valued expression, with a negative value, that identifies an error condition.

Details

Any error code returned by the IOSTAT function or by message string and also 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.
Refer to *V+ User's Manual (Cat. No. I671)* for a list of all the V+ error messages and their error codes.

Examples

The following example 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

3-1-37 ERROR on page 3-48
3-1-58 IOSTAT on page 3-83
3-4-104 REACTE on page 3-417

3-1-37 ERROR

Real-valued function that returns the message number of a recent system message that caused program execution to stop or caused a REACTE operation.

Syntax

```
ERROR (source, select)
```

Usage Considerations

Executing a REACTE statement clears any system messages for the current task and prevents the ERROR function from returning messages as expected.
A first-in-first-out buffer is available that receives all asynchronous messages that occur from the time an enable high power request is issued until power is disabled for any reason. The buffer is accessed using the this function.
The asynchronous message buffer is not valid while the robot is in the power- down initialization state. User programs should wait until STATE(1) <> 0 before calling this function with a source parameter > 1000.
This keyword can be used to return the most recent error number from the fieldbus, similar to the FB.ERROR real-valued function. The source parameter must be 2000 if used in this way.

Additional Information

Refer to *V+ User's Manual (Cat. No. I671)* for error details.

Parameter

Parameter	Description	
source	Real value, variable, or expression interpreted as an integer whose value selects the source of the message code as follows.	
	-2	Return additional message code for current robot. ERROR(-2,0): Returns the standard V+ message ERROR(-2, 1) Returns the motor mask for the current robot. This bit mask indicates the motor(s) referenced for the message number returned by the statement ERROR(-2,0):. The LSB indicates motor 1, etc. If the statement ERROR(-2, 1)=0, the message is not associated with a specific motor.
	-1	Return the number of the most recent message from the program in which the ERROR function is executed.
	0	Return the number of the most recent message from the program executing as task 0.
	0 < source ≤ 63	Return the number of the most recent message from the program executing as the corresponding task number.
	2000	Returns the last V+ system message.
	2001 to 2008	Returns the last message associated with a specific robot where 2001 to 2008 corresponds with robots 1 to 8.

Parameter	Description	
select	Optional real value, variable, or expression interpreted as an integer that selects the message information to be returned. The value 0 is assumed if this parameter is omitted.	
	0	Return the message number of the most recent program execution error excluding I/O errors. Refer to *3-1-58 IOSTAT* on page 3-83 for the specified program task.
	1	If the most recent message for the specified program task had a code in the range -1100 to -1199, return the variable part of the corresponding message as a numeric value. If the most recent message had an error code in the range -1000 to -1099, return the variable portion of the corresponding message as a bit mask indicating the joints or motors to which the message applies. Zero is returned if the message did not have a variable portion in its message. Refer to select = 3 below.
	2	Return the message number of the most recent message from an MCS Program command executed by the specified program task.
	3	Return the number of the robot associated with the most recent message for the specified program task. Zero is returned if the message was not associated with a specific robot. Refer to select = 1.

Details

An V+ task can access any messages that result in robot power being disabled. These include the asynchronous messages that previously were output only to the Monitor Window.
This function is useful in a REACTE subroutine program to determine why the REACTE was triggered. Refer to *V+ User's Manual (Cat. No. I671)* for a list of all the V+ system messages and their error numbers.
When the select parameter is 1, the value returned by this function should be interpreted as a 6-bit numeric value. The following example illustrates how the value should be interpreted.

Additional Information

The ERROR function does not report errors reported by the IOSTAT function.

Examples

The example below will return the message corresponding to an error code where the following conditions are present.

- "code" is the basic error code from ERROR(n) or IOSTAT(lun) for example.
- "vcode"is the variable part of the error code from ERROR (n,1) for example.
- "robot" is the number of the robot associated with the error from ERROR (n,3) for example.
- "$msg"is the corresponding error message. This may be null.

```
.PROGRAM error.string(code, vcode, robot, $msg)
    AUTO i, n
    $msg = ""
    IF code < 0 THEN
    $msg = $ERROR(code)
    IF (-1100 < code) AND (code <= -1000) THEN
    n = 1
    FOR i = 1 TO 7
        IF vcode BAND n THEN
            $msg = $msg+$ENCODE(i)
        END
        n = 2*n
    END
END
        IF (-1200 < code) AND (code <= -1100) THEN
            $msg = $msg+$ENCODE(vcode)
        END
        IF robot AND (SELECT(ROBOT,-1) > 1) THEN
            $msg = $msg+" (Robot"+$ENCODE(robot)+")"
        END
    END
    RETURN
.END
```

Related Keywords

3-1-36 $ERROR on page 3-47
3-1-58 IOSTAT on page 3-83
3-4-79 MCS on page 3-381
3-4-104 REACTE on page 3-417
3-1-39 FB.ERROR on page 3-52

3-1-38 FALSE

Real-valued function that returns the value used by V+ to represent a logical false result.

Syntax

```
FALSE
```

Details

This named constant is useful for situations where true and false conditions need to be specified. The value returned is 0.

Examples

The following example program loop will execute continuously until the sub- routine cycle returns a FALSE value for the real variable "continue".

```
DO
    CALL cycle(continue)
UNTIL continue == FALSE
```

Related Keywords

3-1-72 OFF on page 3-98
3-1-73 ON on page 3-98
3-1-115 TRUE on page 3-149

3-1-39 FB.ERROR

Real-valued function that returns the most recent fieldbus error code.

Syntax

```
FB.ERROR ($error_description)
```

Usage Considerations

The FB.ERROR keyword is not available for all robots. This keyword is only compatible with robots that support EtherCAT SubDevice and PROFINET fieldbus types. Refer to the robot user's manual for more information.
This function can be executed using the Monitor Window.
The system does not store past errors.
Errors can also be returned using the ERROR function keyword. To return errors with the ERROR keyword the source parameter of must be 2000 if used in this way. Refer to *3-1-37 ERROR* on page 3-48 for more information.

Parameter

Keyword	Description
$error_ description	Optional string variable that returns additional error information.

Details

The FB.ERROR keyword returns error information for both the PROFINET and the EtherCAT fieldbus types. The information returned is unique to each fieldbus type.

● PROFINET Error Codes

PROFINET error codes and their description details returned from the FB.ERROR keyword are provided below.

Error Code	Error Description	Details
-1202 (fieldbus system initialization fault)	Error Code 1	• I&M configuration corrupted. • Memory not initialized. • Network interface not available.
	Error Code 2	PROFINET stack initialization failed.
	Error Code 3	LLDP system description in illegal status. • Port description TLV (port 0). • System name TLV. • System description TLV.
	Error Code 4	LLDP port description in illegal status. Port description TLV (port 1).
	Error Code 5	LLDP PHY not initialized.
	Error Code 6	PROFINET cannot be initialized because of a version mismatch.
-1201 (connection lost)	Connection rejected x y z a where: • x = Error code • y = Error decode • z = Error code 1 • a = Error code 2	The connection request was rejected by the PROFINET controller.
	Communication closed because of a comm loss	The connection has been aborted by the PROFINET controller.

Error Code	Error Description	Details
-1203 (invalid data)	Not able to read data because the PLC has stopped	Communication data is invalid because the Controller has stopped.
	Not able to read data because supervisor is blocking submodule	Communication data is invalid because the supervisor is blocking the communication.
	Not able to read data because the submodule is invalid	Communication data is invalid because the submodule is invalid.
-1200 (invalid configuration)	Module Diff (Expected x, y, Actual z a) at b, c where: • x, z = Module ident number • y, a = Submodule ident number • b = slot number • c = subslot number	A mismatch between the PROFINET I/O and ACE submodule configuration exists.
	Peer Mismatch x where x = extended channel error type	The peer port is mismatched in the port name.
	MAU Type Mismatch: Expected x, Actual y where x, y = MAU Type	The peer port mismatched in the MAU type.
	Link State Mismatched (Expected x y, Actual z a) where: • x, z = Link State • y, a = Port State	The peer port is mismatched in the link.
	Variable Type Mismatch (Expected: 0xY, Real: 0xA) at slot b, subslot c where: • Y, A = submodule Ident Number in hex • b = slot number • c = subslot number	A mismatch between PROFINET I/O and the ACE submodule configuration occurred.
	Robot Type Mismatch (Expected: 0xY, Real: 0xA) at slot b, subslot c where: • Y, A = Module Ident Number in hex • b = slot number • c = subslot number	The actual robot is different than the one configured in the PROFINET controller.

Additional Information

Refer to *Industrial Robot Fieldbus Configuration User's Guide (Cat. No. M130)* for more detailed information.

● EtherCAT Error Codes

EtherCAT error codes and their description details returned from the FB.ERROR keyword are provided below.

Error Code	Error Description	Details
-1200 (invalid configuration)	Failed to create PDO mapping	General MainDevice invalid configuration error.
	Tx / Rx PDO setting error	Invalid SyncManager (SM) configuration in the MainDevice.
	Tx / Rx PDO mapping error	Invalid PDO or EtherCAT Objects assignment in MainDevice configuration.
	The Tx / Rx mapping object to be assigned does not exist	Invalid MainDevice PDO configuration.
	A value that does not exist in the data type of the mapped entry is used in Tx / Rx	
	The assigned size exceeds the registered input buffer size for Tx / Rx	
	Length mismatch between the mapped object and the object dictionary table in Tx / Rx	
	The entry index in mapping parameter of a Tx / Rx is not registered in the object dictionary	
	The number of subindexes requested to assign to Subindex0 exceeds the maximum number of assignments	
	For SDO complete access, the data size exceeds the maximum number of assignments	
	For SDO single access, the size assigned to the Subindex0 is not 1 byte	
	For SDO access to write 0x1C12 / 0x1C13, Subindex1 size is not 2 bytes	
	The index to assign to 0x1C12 is out of the range from 0x1600 to 0x17FF	
	The index assigned to 0x1C13 is out of the range from 0x1A00 to 0x1AFF	
	Flexible PDO configured with a subindex that does not exist in Tx / Rx	
	The mapped entry does not exist in the table for Rx / Tx	
	The number of subindexes requested to map to Subindex0 exceeds the maximum number in Tx / Rx	

Error Code	Error Description	Details
-1200 (invalid configuration)	For SDO complete access, data size following the Subindex1 exceeds the maximum number in Tx / Rx	Invalid MainDevice PDO configuration.
	For SDO single access, the size mapped to the Subindex0 is not 1 byte in TX / Rx	
	For SDO access to configure flexible PDO, Subindex1 is not 4 bytes in Tx / Rx	
	Input object has been assigned to output PDO or vice versa	
	Output / Input object 0xXXXX:YY configured in MainDevice is not configured in SubDevice	XXXX is the object ID and YY is the subindex. MainDevice configuration has one or more objects that has not been configured in ACE.
	Unsupported sync cycle time	Invalid cycle time that is not compatible with the robot has been set in MainDevice.
	The subindex of a mapping parameter to one following the Subindex1 does not exist in Tx / Rx	Invalid MainDevice PDO configuration
	Length mismatch between mapped object and object dictionary table in Tx / Rx	
	The entry index in mapping parameter of a Tx / Rx is not registered in the object dictionary	
	For SDO single access, the number of subindexes requested to map exceeds the maximum in Tx / Rx	
	Input/Output PDO configuration has subindexes of an object of type 'GROUP OF' that does not start from 1	First configured subindex of all objects called “GROUP OF” must be number 1.
	Input / Output PDO configuration has subindexes of an object of type 'GROUP OF' that are not consecutive	All configured subindex of all objects called “GROUP OF” must be consecutive (such as {1, 2, 3} and not {1, 3, 4}).
	Input / Output PDO configuration has subindexes of an object of type 'GROUP OF' that are duplicated	All configured subindex of all objects under “GROUP OF” cannot be duplicated (such as. {1, 2, 3} and not {1, 2, 2}).
	Input / Output object ARRAY OF BYTE has been added to a flexible PDO	Objects ARRAY OF BYTES (676x and 776x) cannot be added to any flexible PDO but just selected using their fixed PDO.

Error Code	Error Description	Details
-1201 (connection lost)	FreeRun mode watchdog timer triggered	In FreeRun mode, time out happens between packets.
	PDO sync lost	In DC mode, more than 1 EtherCAT packet has been lost.
-1202 (system init fault)	Mismatch in contents of first 8 words of the SII EEPROM (EtherCAT SubDevice Information Electrically Erasable Programmable Read-Only Memory)	SubDevice internal non-volatile memory corruption.
	EEPROM loaded signal timeout or PDO access error	
	SII error	
	CPU error	System fault
	Bootstrap not supported	MainDevice requested SubDevice to transit to Bootstrap state
	No sync when trying to pass to OP (operational) state	System fault
	Mailbox SM setting error	Invalid MainDevice Mailbox setting.
	The length of SM is 0 even when it is active.	Invalid MainDevice SM configuration
	The mapping size does not agree with the size set to SM.	
	Invalid SM direction	
	The operation mode of SM Control Register is set to Reserved even when SM is active	
	The start address is set to a value less than 0x1000 in the Pre-Op state even when SM is active	
	The buffer size and start address of SM exceed 0x2FFF in the Pre-Op state even when SM is active	
	The SM address configured in PreOp has been changed when passing to OP	
	A size other than 0 is set to SM or the mapping size is not 0 even when SM is not active	
	The start address is odd even when SM is active	
	The SM3 buffer overlaps the SM2 buffer, or vice versa.	
	Unknown error	Abnormal situation not covered by error cases
-1024 (invalid write input variable)	--	Write read-only variables
-1025 (conversion overflow)	--	Write V+ variable out of range for more information.

Additional Information

Refer to *Industrial Robot Fieldbus Configuration User's Guide (Cat. No. M130)* for more detailed information.

Examples

The following example will create $info as an empty string. If an error is present, it will then return the error code and additional error information.

```
$info = ""

error_nr = FB.ERROR($info)
IF error_nr THEN
   TYPE "Error info: ", $info
END
```

Related Keywords

3-1-37 ERROR on page 3-48

3-1-40 FB.STATE

Real-valued function that returns the state of the fieldbus.

Syntax

FB.STATE

Usage Considerations

The FB.STATE keyword is not available for all robots. This keyword is only compatible with robots that support EtherCAT SubDevice and PROFINET fieldbus types. Refer to the robot user's manual for more information.

Precautions for Correct Use

Disconnecting EtherCAT communications while the robot is transitioning between control states may cause an invalid fieldbus state. Cycle power to the robot to recover from this condition.

Details

The FB.STATE function returns a numerical output that corresponds to the fieldbus status information shown in the table below.
The definition of the numerical output is dependent upon whether the fieldbus is configured for PROFINET or EtherCAT SubDevice functionality

Output	Fieldbus Status	
	PROFINET	EtherCAT
0	Not enabled.	
1	Enabled but inactive.	
2	Enabled and connected, but there is no data exchange. Refer to the *Industrial Robot Fieldbus Configuration User's Guide (Cat. No. M130)* for more information.	Enabled and connected, but with limited data exchange. The EtherCAT SubDevice (robot) is in PreOp or SafeOp state. Refer to *Industrial Robot Fieldbus Configuration User's Guide (Cat. No. M130)* for more information.
3	Enabled, connected, and exchanging data.	

Examples

The following example returns the state of the fieldbus to the "state" variable.

```
state = FB.STATE
```

3-1-41 $FLTB

String function that returns a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.

Syntax

```
$FLTB (value, littleEndian)
```

Parameter

Parameter	Description
value	Real-valued expression, the value of which is converted to its IEEE floating-point binary representation.
littleEndian	Optional value to specify the type of little endian. 1 specifies little endian, 0 specifies big endian.

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. Refer to the *3-1-42 FLTB* on page 3-60 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.

Examples

The following example demonstrates how to encode a V+ numeric variable (real_value) into its IEEE 754 single-precision floating-point format, resulting in a 4-byte binary string ($s).
This example also demonstrates the differences between little-endian and big-endian encoding by displaying the resulting byte order in each case using the TYPE keyword.
The encoded bytes are displayed as hexadecimal values for comparison.

```
TYPE $CHR(27)+"[H"+$CHR(27)+"[J"
real_value = 1.215

little.endian = 1
$s = $FLTB(real_value,little.endian)

TYPE $ENCODE("Little Endian: ", /H2, ASC($s,1), "-", ASC($s,2), "-", ASC($s,3), "-"
, ASC($s,4))

little.endian = 0
$s = $FLTB(real_value, little.endian)

TYPE $ENCODE("Big Endian:    ", /H2, ASC($s,1), "-", ASC($s,2), "-", ASC($s,3), "-"
, ASC($s,4))
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-22 DBLB on page 3-30
3-1-21 $DBLB on page 3-29
3-1-42 FLTB on page 3-60
3-1-55 $INTB on page 3-79
3-1-64 LNGB on page 3-91
3-1-63 $LNGB on page 3-89
3-1-114 TRANSB on page 3-147

3-1-42 FLTB

Real-valued function that returns the value of four bytes of a string interpreted as an IEEE single-precision floating-point number.

Syntax

```
FLTB ($string, first_char, littleEndian)
```

Parameter

Parameter	Description
$string	Real-valued function that returns the value of four bytes of a string interpreted as an IEEE single-precision floating-point number.
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 the first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 indicates 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.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian, 0 specifies big endian.

Details

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+.
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.

<table>
<tr><td>31</td><td colspan="2">30 23</td><td colspan="4">22 0</td></tr>
<tr><td>S</td><td colspan="2">Exponent</td><td colspan="4">Fraction</td></tr>
<tr><td colspan="2">Byte 1</td><td colspan="2">Byte 2</td><td>Byte 3</td><td>Byte 4</td></tr>
</table>

Item	Description
S	Sign bit (0 = positive, 1 = negative)
Exponent	Binary exponent, biased by -127
Fraction	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:

$-1s * (1.fraction) * 2^{exp\ -127}$

For exp = 0, the value is zero; for exp = 255, an overflow error exists.

Examples

This example encodes a V+ variable (out_float) into a 4-byte binary string ($s) using $FLTB. Then, the binary string is decoded to a float value (in_float) using FLTB.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
out_float = -1.5716
TYPE "Variable 'out_float':    ", /F6.3, out_float
```

```
little.endian = 1
$s = $FLTB(out_float,little.endian)

TYPE "out_float encoded (Little Endian):    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))
TYPE

TYPE "Simulate receiving 4 bytes:    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))

in_float = FLTB($s,,little.endian)

TYPE "Variable 'in_float' interpreted from 4 bytes: ", /F6.3, in_float
```

Related Keywords

3-1-5 ASC on page 3-11
3-1-22 DBLB on page 3-30
3-1-21 $DBLB on page 3-29
3-1-41 $FLTB on page 3-59
3-1-56 INTB on page 3-80
3-1-64 LNGB on page 3-91
3-1-63 $LNGB on page 3-89
3-1-114 TRANSB on page 3-147
3-1-122 VAL on page 3-156

3-1-43 FRACT

Real-valued function that returns the fractional part of the argument.

Syntax

FRACT (value)

Parameter

Parameter	Description
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

The following example returns 0.123.

```
FRACT(0.123)
```

The following example returns -0.462.

```
FRACT(-5.462)
```

The following example returns 0.25.

```
FRACT(1.3125E+2)
```

Related Keywords

3-1-54 INT on page 3-78

3-1-44 FRAME

Transformation function that returns a transformation value defined by four positions.

Syntax

```
FRAME (location_1, location_2, location_3, location_4)
```

Parameter

Parameter	Description
location_1	Transformation, compound transformation, or a transformation-valued function whose position is used to define the X axis of the computed frame. The positive X axis is parallel to a line passing through the points defined by location_1 and location_2 in the direction from location_1 to location_2.
location_2	Transformation, compound transformation, or a transformation-valued function whose position is used to define the X axis of the computed frame. The positive X axis is parallel to a line passing through the points defined by location_1 and location_2 in the direction from location_1 to location_2.
location_3	Transformation, compound transformation, or a transformation-valued function whose position is used to define the Y axis of the computed frame. The X-Y plane is parallel to the plane that contains the points defined by location_1, location_2, and location_3.
location_4	Transformation, compound transformation, or a transformation-valued function whose position is returned as the position of the computed frame transformation. The origin is at the point defined by location_4.

Details

While the robot can be used to define an X, Y, Z position very accurately, it is often difficult to define an orientation precisely. 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.

Examples

The following example 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 Keywords

3-1-112 TRANS on page 3-144

3-1-45 FREE

Real-valued function that returns the amount of unused free memory of storage space.

Syntax

```
FREE (memory, select)
```

Function

Return the amount of unused free memory storage space.

Parameters

Parameter	Description
memory	Optional real value, variable, or expression interpreted as an integer that specifies which portion of system memory is to be examined. The value zero is assumed if the parameter is omitted. • 0: Program memory • 1: Reserved for future use • 2: Reserved for future use
select	Optional real value, variable, or expression interpreted as an integer that specifies what information about the memory is to be returned. The value zero is assumed if the parameter is omitted. • 0: Percentage of memory available • 1: Available memory in kilobytes (1024 bytes)

Details

This function returns the information displayed by the FREE monitor command. Unlike the FREE monitor command, this function returns only one value determined by the values specified for the memory and select parameters.

Examples

The following example will display the available memory in kilobytes in the Monitor Window.

```
memory = FREE(0,1)
TYPE memory
```

Related Keywords

3-1-45 FREE on page 3-64 (monitor command)

3-1-46 GETC

Real-valued function that returns the next character (byte) from a device or input record on the specified logical unit.

Syntax

```
GETC (lun, mode)
```

Usage Considerations

The logical unit must be attached by the program for normal, variable-length record input / output.

Parameter

Parameter	Description
lun	Real value, variable, or expression interpreted as an integer that identifies the device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 command for a description of the unit numbers.

Parameter	Description
mode	Real value, variable, or expression interpreted as an integer that specifies the mode of the read operation. The mode is used only for the terminal. 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 (least significant bit): disable waiting for input (mask value = 1)** If this bit is OFF, program execution is suspended until the next byte is received. If the bit is ON and no bytes are available, the function immediately returns the error code for "No data received" (-526). 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 OFF, input from the terminal is echoed back to the source. If the bit is ON, characters are not echoed back to the source.

Details

The next byte from the device is returned. When reading from a record-oriented device such as the disk file or Monitor Window, 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.
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. Refer to *V+ User's Manual (Cat. No. I671)* for more information about error numbers.

Examples

The following example 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

3-4-11 ATTACH on page 3-279
3-4-106 READ on page 3-421

3-1-47 GET.EVENT

Real-valued function that return events that are set for the specified task.

Syntax

```
GET.EVENT) task)
```

Usage Considerations

Do not confuse this function with the GETEVENT program command.

Parameter

Parameter	Description
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 64, inclusive. If the parameter is omitted or has the value -1, the current task is referenced.

Details

This function returns events of the specified task. If the task parameter is set to a value of -1, it will return events of the current task.
The events are returned in a value that should be interpreted as a sequence of bit flags as detailed below.
Bit 1 (least significant bit) I/O Completion (mask value = 1)
When this bit is ON, it indicates that a system input/ output operation has completed. Refer to *3-4-116 SET.EVENT* on page 3-432 and *3-4-137 WAIT.EVENT* on page 3-461 keywords for more information.

Related Keywords

3-4-24 CLEAR.EVENT on page 3-303
3-4-116 SET.EVENT on page 3-432
3-4-137 WAIT.EVENT on page 3-461

3-1-48 HERE

Transformation function that returns a transformation value that represents the current location of the robot tool point.

Syntax

HERE

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.
The HERE function returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Examples

The following example calculates the distance between the current robot location and the location the robot is currently moving to.

```
dist = DISTANCE(HERE, DEST)
```

Related Keywords

3-1-29 DISTANCE on page 3-40
3-2-30 HERE on page 3-207 (monitor command)
3-4-67 HERE on page 3-366 (program command)
3-4-121 SINGLE on page 3-438 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-1-49 HOUR.METER

Real-valued function that returns the current value of the robot hour meter.

Syntax

HOUR.METER(robot number**)**

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 in robot controller. If the Robot Signature Card does not respond, the HOUR.METER function returns the value -1.

Examples

The following example assigns the current reading of the robot hour meter to the real-valued variable "start".

```
start = HOUR.METER
```


3-1-50 $ID

String function that returns the system ID string.

Syntax

$ID (select)

Parameter

Parameter	Description	
select	Integer	Description
	-1	Returns the system revision message.
	-2	Returns the revision letter and issue number for the V+ system.
	-3	Returns the vision revision message string.
	-4	-4 Returns the servo revision message string.

Details

This function returns a string that identifies the revision and date of the requested system software component.

Examples

The following example will display the revision letter and issue number for the V+ system in the Monitor Window.

```
$string1=$ID(-2)
TYPE $string1
```

Related Keywords

3-2-31 ID on page 3-209 (monitor command)
3-1-51 ID on page 3-70 (function)

3-1-51 ID

Real-valued function that returns values that identify the configuration of the current system.

Syntax

```
ID (component, device, board)
```

Usage Considerations

The function returns the value 0 for devices that do not exist. Device numbers that do not exist return the value 0. For valid devices, an *Invalid argument* error message is reported if the requested component is not valid.

Parameter

Parameter	Description
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 the device parameter value equals 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 keyword. The values of the components are the same as the fields displayed by that command.
Use the table below to understand parameter settings and the associated information that is returned with the ID real-valued function.

Device Number: 1 Basic System	
Component	**Information Returned**
1: Model Number	The model designation of the robot is returned when the robot is in operation mode.
2: Serial Number	The NJ-series Robot Integrated CPU Unit returns 3000. When the robot is in service mode, the serial number of the robot is returned.
3: V+ Internal Major Version	The V+ internal major version is returned.
4: V+ Internal Minor Version	The V+ internal minor version is returned.

Device Number: 1 Basic System	
Component	**Information Returned**
11: Controller Hardware Configuration	A value of 4 is returned if Emulation Mode is active. A value of 0 is returned if Emulation Mode is not active.
14: V+ External Version	A value of 3 is returned (V+3).

Device Number: 2 T20 Pendant	
Component	**Information Returned**
1: Pendant Version	The model designation of the robot is returned when the robot is in operation mode. If Emulation Mode is operating or the Pendant is not connected, the error *Pendant Not Connected* (- 657) is returned.
2: Pendant Major Version	The major version number of the Pendant software is returned. If Emulation Mode is operating or the Pendant is not connected, the error *Pendant Not Connected* (- 657) is returned.
3: Pendant Minor Version	The minor version number of the Pendant software is returned. If Emulation Mode is operating or the Pendant is not connected, the error *Pendant Not Connected* (- 657) is returned.
4: V+ Internal Minor Version	The build version of the Pendant software is returned. If Emulation Mode is operating or the Pendant is not connected, the error *Pendant Not Connected* (- 657) is returned.
5: Pendant Revision Version	The revision version of the Pendant software is returned. If Emulation Mode is operating or the Pendant is not connected, the error *Pendant Not Connected* (- 657) is returned.
6: Pendant Connected	The status of the pendant connection is returned. • -1: Pendant is connected and communicating. • 0: A time out occurred and the pendant could not be detected. If Emulation Mode is operating or the Pendant is not connected, the error *Pendant Not Connected* (- 657) is returned.

Device Number: 4 System CPUs		
Component	**Board**	**Information Returned**
1: Number of the CPU	CPU Board (-32768 to 32767)	The number of the CPU is returned. A non-existant CPU will return a value of - 1. The main CPU will return a value of 1.
5: CPU Type	Not Used	The NJ-series Robot Integrated CPU Unit returns a value of 9.

Device Number: 8 Robot Configuration for currently selected robot	
Component	**Information Returned**
1: Robot Model	Returns the model designation of the robot
2: Robot Serial Number	Returns the serial number of the robot.
3: Number of Motors	Returns the number of motors configured for the robot.
4: Value Interpreted as Bit Flags for the Robot Joints That Are Enabled	Returns a value interpreted as bit flags for the robot joints that are enabled. A value of 0 is returned if the robot does not have joints that can be selectively disabled.
5: Kinematic Module ID	Returns the robot control module identification number.
7: Number of Robot Joints configured for use	Returns the number of robot joints configured for use.
8: First Robot Option Word	Returns the first Robot Option Word. Refer to *Robot Option Words* on page 3-74 for more information.
9: Product Type	Returns the robot product type.
10: Number of Cartesian Axes Used During Motion Planning	Returns the number of axis used during motion planning. This number specifies how many values must be defined in the robot load module arrays to determine the maximum cartesian velocities and accelerations.
11: Second Robot Option Word	Returns the Second Robot Option Word. Refer to *Robot Option Words* on page 3-74 for more information.
12: Information About the Robot Module	Returns information about the robot module. Only bit 1 (mask 1) is defined. If this bit is ON, the specified robot is an OMRON robot.
13: Robot Safety Level	Returns the safety level configured for the robot. • 0: No safety level configured. • 1: Configured as a Category 1 Robot System per ISO 10218 and EN954. • 3: Configured as a Category 3 Robot System per ISO 10218 and EN954.
15: eSeries Robot Type	Returns the eSeries robot type. • 0: eSeries Lite • 1: eSeries Standard • 2: eSeries Pro • 3: sSeries
16: First and Second Parts of the Robot Security ID	Returns the first and second part of the Robot Security ID, shown as "aaaa-bbbb" below. Security ID: aaaa-bbbb-cccc
17: Third Part of the Robot Security ID	Returns the third part of the Robot Security ID, shown as "cccc" below. Security ID: aaaa-bbbb-cccc
18: Number of Tasks Currently Attached to the Robot	Returns the number of the task currently attached to the robot.
19: Robot Joint Control	Returns a value of -1 if the robot can be moved under joint control mode. Returns a value of 0 if the robot can be moved under joint control mode.

Device Number: 10 Belt	
Component	**Information Returned**
7: Number of Belts	Returns the number of belts configured in the system.

Device Number: 11 to 18 Robot configuration for robots 1 to 8 (10 + robot number)	
Component	**Information Returned**
1: Robot Model	Returns the model designation of the robot.
2: Robot Serial Number	Returns the serial number of the robot.
3: Number of Motors	Returns the number of motors configured for the robot.
4: Value Interpreted as Bit Flags for the Robot Joints That Are Enabled	Returns a value interpreted as bit flags for the robot joints that are enabled. A value of 0 is returned if the robot does not have joints that can be selectively disabled.
5: Kinematic Module ID	Returns the robot control module identification number.
7: Number of Robot Joints Configured for Use	Returns the number of robot joints configured for use.
8: First Robot Option Word	Returns the first Robot Option word. Refer to *Robot Option Words* on page 3-74 for more information.
9: Product Type	Returns the robot product type.
10: Number of Cartesian Axes Used During Motion Planning	Returns the number of axis used during motion planning. This number specifies how many values must be defined in the robot load module arrays to determine the maximum cartesian velocities and accelerations.
11: Second Robot Option Word	Returns the second Robot Option word. Refer to *Robot Option Words* on page 3-74 for more information.
12: Information About the Robot Module	Returns information about the robot module. Only bit 1 (mask 1) is defined. If this bit is ON, the specified robot is an OMRON robot.
13: Robot Safety Level	Returns the safety level configured for the robot. • 0: No safety level configured • 1: Configured as a Category 1 Robot System per ISO 10218 and EN954 • 3: Configured as a Category 3 Robot System per ISO 10218 and EN954 • 3: Configured as a Category 3 Robot System per ISO 10218 and EN954
15: eSeries Robot Type	Returns the eSeries robot type. • 0: eSeries Lite • 1: eSeries Standard • 2: eSeries Pro • 3: sSeries
16: First and Second Parts of the Robot Security ID	Returns the first and second part of the Robot Security ID, shown as "aaaa-bbbb" below. Security ID: aaaa-bbbb-cccc

Device Number: 11 to 18 Robot configuration for robots 1 to 8 (10 + robot number)	
Component	**Information Returned**
17: Third Part of the Robot Security ID	Returns the third part of the Robot Security ID, shown as "cccc" below. Security ID: aaaa-bbbb-cccc
18: Number of Tasks Currently Attached to the Robot	Returns the number of the task currently attached to the robot.
19: Robot Joint Control	Returns a value to indicate if the robot can be moved under joint control. • -1: robot can be moved under joint control • 0: robot cannot be moved under joint control

● Robot Option Words

The interpretation of the high-order bits in the first option word are described in the following table.

Bit Number	Mask Value		Description
	Decimal	Hexadecimal	
9	256	100	When this bit is ON, motor limit- stop testing is enabled. This should be enabled only for robots that have excessive motor coupling and that have motor limit-stop data allocated.
10	512	200	When this bit is ON, free mode power OFF is enabled. Any limit violations during free mode cause the robot power to be disabled.
11	1024	400	When this bit is ON, the system will automatically execute a CALIBRATE monitor command upon boot.
12	Reserved for future use.		
13	4096	1000	When this bit is ON, the system checks for collisions with static obstacles in cartesian space during joint-interpolated motions. When this bit is ON, joint-interpolated motions have the same computational load as straight line motions.
14	8192	2000	Reserved for future use.
15	Reserved for future use.		
16			

The interpretation of the bits in the second option word are described in the following table.

Bit Number	Mask Value		Description
	Decimal	Hexadecimal	
1	1	1	When this bit is ON, the robot has an RSC.
2	2	2	When this bit is ON, the robot is equipped with an extended-length quill.
3	4	4	When this bit is ON, the robot is equipped with the clean room option.
4	8	8	Reserved for future use.
5	16	10	When this bit is ON, the robot is equipped with the the high-torque option.
6	32	20	Reserved for future use.
7	64	40	When this bit is ON, the robot is equipped with the EC certification option.
8	128	80	When this bit is ON, the robot has a high resolution joint 4.
9	256	100	Reserved for future use.
10	512	200	When this bit is ON, the robot has an amber LED. This option is only available with eCobra robot models.
11	Reserved for future use		
12			
13			
14			
15			
16			

Related Keywords

3-2-31 ID on page 3-209 (monitor command)
3-1-50 $ID on page 3-69 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-1-52 IDENTICAL

Real-valued function that determines whether two location values are exactly the same.

Syntax

```
IDENTICAL (location1, location2)
```

Parameter

Parameters	Description
location1	Transformation value that defines one of the locations of interest. This can be a function, a variable, or a com- pound transformation.
location 2	Transformation value that defines one of the locations of interest. This can be a function, a variable, or a com- pound 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 results in the value FALSE being returned.

Examples

The following example sets 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".

```
x = IDENTICAL(base.1:loc,part)
```

Related Keywords

3-1-29 DISTANCE on page 3-40

3-1-53 INRANGE

Real-valued function that returns a value that indicates if a specific location can be reached by the robot and provides additional information when a location cannot be reached.

Syntax

```
INRANGE (location)
```

Usage Considerations

The INRANGE function returns information for the robot selected by the task executing the function. Use INRANGE( ) to check if the selected robot current joint positions are within range of the joint motion limits for programmed motion. For instance, if you manually rotate the tool flange of a SCARA robot out of its default range [- 360,360] while you are teaching the robot a new position, INRANGE( )

will report Joint 4 out of range. In such a case, the robot can still be jogged back into range by jogging Joint 4.
Use INRANGE(HERE) to check if the selected robot Cartesian position HERE is within range; however, it cannot be used to properly detect a SCARA robot Joint4 or articulated robot Joint 6 that is not within its motion limits.
INRANGE(location) checks if the reduced transformation is within the work envelope of the robot, and it is possible that a joint is not in range while the reduced transformation is in range. This can result in a false positive indicating the current robot position is in range when it is not.

Parameter

Parameter	Description
location	Optional transformation function, variable, or compound that specifies a desired position and orientation for the robot tool tip. If this parameter is omitted, INRANGE will indicate if the current location of the selected robot can be reached.

Details

This function returns a value that indicates whether or not the given location can be reached by the robot. A returned value of 0 indicates that the specified location can be reached.
If the location cannot be reached, the returned value is a coded binary number that provides additional information. A bit equal to 1 in the value indicates that the corresponding robot constraint would be violated, as shown in the table below.

Bit Number	Mask Value		Indication if bit set
	Hex	Decimal	
1	1	1	Joint or motor 1 is limiting
2	2	2	Joint or motor 2 is limiting
3	4	4	Joint or motor 3 is limiting
4	8	8	Joint or motor 4 is limiting
5	10	16	Joint or motor 5 is limiting
6	20	32	Joint or motor 6 is limiting
7	40	64	Joint or motor 7 is limiting
8	80	128	Joint or motor 8 is limiting
9	100	256	Joint or motor 9 is limiting
10	200	512	Joint or motor 10 is limiting
11	400	1024	Joint or motor 11 is limiting
12	800	2048	Joint or motor 12 is limiting
13	1000	4096	Collision detected
14	2000	8192	Location is too close in
15	4000	16384	Location is too far out
16	8000	32768	Motor is limiting, rather than joint (see below)

Bit Number	Mask Value		Indication if bit set
	Hex	Decimal	
17	10000	65536	Orientation is out of range for the Quattro platform
18	20000	131072	Kinematic solution not found

If the motion system is configured to return motor-limit as well as joint-limit errors, bit 16 indicates whether a joint or motor would limit motion to location. If bit 16 is set, all the joints passed their limit checks and the indicated motor is limiting. Otherwise, the indicated joint is limiting.
The mask values indicated above can be used with the BAND keyword to determine if a corresponding bit is set.

Examples

The following example returns a value of 0 if the robot can reach the location defined by the compound transformation "pallet:hole". If both joints 2 and 3 would prevent the motion from being made, the value returned would be 6.

```
INRANGE(pallet:hole)
```

Related Keywords

3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-1-54 INT

Real-valued function that returns the integer part of the value.

Syntax

```
INT (value)
```

Parameter

Parameter	Description
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

The following example returns 0.0.

```
INT(0.123)
```

The following example returns 10.0.

```
INT(10.8)
```

The following example returns -5.0.

```
INT(-5.462)
```

The following example returns 131.0.

```
INT(1.3125E+2)
```

The following example returns the value of "cost" truncated to an integer.

```
INT(cost)
```

The following example returns the value of "cost" rounded to the nearest integer. The sign of the function needs to be included to correctly round negative values of "cost".

```
INT(cost+0.5*SIGN(cost))
```

Related Keywords

3-1-43 FRACT on page 3-62

3-1-55 $INTB

String function that returns a 2-byte string containing the binary representation of a 16-bit integer.

Syntax

$INTB(value, littleEndian**)**

Parameter

Parameter	Description
value	Real-valued expression, the value of which is converted to its binary representation.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

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 the following.

```
$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.

Examples

The example demonstrates how to encode and decode a V+ numeric variable. The V+ variable (int_value) is encoded using the $INTB keyword and returned as a 2-byte string ($s). Then, the string is decoded using the INTB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
int_value = -12345
TYPE "Original 'int_value': ", int_value

little.endian = 1
$s = $INTB(int_value,little.endian)

TYPE "int_value encoded (Little Endian):    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2))
TYPE

TYPE "Simulate receiving 2 bytes:    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2))

decoded_int = INTB($s,,little.endian)

TYPE "Decoded 'decoded_int' from 2 bytes:    ", decoded_int
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-21 $DBLB on page 3-29
3-1-41 $FLTB on page 3-59
3-1-56 INTB on page 3-80
3-1-63 $LNGB on page 3-89

3-1-56 INTB

Real-valued function that returns the value of two bytes of a string interpreted as a signed 16-bit binary integer.

Syntax

```
INTB ($string, first_char, littleEndian)
```

Parameter

Parameter	Description
$string	String expression that contains the two bytes to be converted.

Parameter	Description
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 establishes 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.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

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 two expression below (INTB vs. ASC) have the same functionality.

```
value = INTB($string, first_char)
value = ASC($string,first_char)*256 + ASC($string,first_char+1)
IF value > 32767 THEN
    value = value-65536
END
```

To compute an unsigned integer, use the following statement.

```
INTB($string) BAND ^HFFFF
```

Examples

The example demonstrates how to encode and decode a V+ numeric variable. The V+ variable (int_value) is encoded using the $INTB keyword and returned as a 2-byte string ($s). Then, the string is decoded using the INTB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
int_value = -12345
TYPE "Original 'int_value': ", int_value

little.endian = 1
$s = $INTB(int_value,little.endian)

TYPE "int_value encoded (Little Endian):    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2))
TYPE

TYPE "Simulate receiving 2 bytes:    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2))

decoded_int = INTB($s,,little.endian)
```

```
TYPE "Decoded 'decoded_int' from 2 bytes:    ", decoded_int
```

Related Keywords

3-1-5 ASC on page 3-11
3-1-22 DBLB on page 3-30
3-1-42 FLTB on page 3-60
3-1-55 $INTB on page 3-79
3-1-64 LNGB on page 3-91
3-1-122 VAL on page 3-156

3-1-57 INVERSE

Transformation function that returns the transformation value that is the mathematical inverse of the given transformation value.

Syntax

INVERSE (transformation)

Parameter

Parameter	Description
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
```

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.

Examples

The following example will encode an INT16 variable as a 2-byte ASCII string and then decode the incoming 2 byte string into an INT16 variable. The 2 bytes contain the binary representation as unsigned character values.

```
int16 = -32767

little.endian = 0
$s = $INTB(int16,little.endian)

TYPE $ENCODE("int16 as 2 outgoing bytes: ",/H2,ASC($s,1),"-",ASC($s,2))
```

```
little.endian = 0
TYPE $ENCODE("incoming 2 bytes (hex): ",/H2,ASC($s,1),"-",ASC($s,2))
int16 = INTB($s,,little.endian)

TYPE "translate to: ", int16
```

The Monitor Window will display the output shown below.

```
int16 as 2 outgoing bytes: 80- 1
incoming 2 bytes (hex): 80- 1
translate to:  -32767
```

Related Keywords

3-1-48 HERE on page 3-68
3-4-117 SET on page 3-433

3-1-58 IOSTAT

Real-valued function that returns status information for the last input/output operation for a device associated with a logical unit.

Syntax

```
IOSTAT (lun, mode)
```

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.

Precautions for Correct Use

IOSTAT should be used after each I/O operation. If one I/O operation fails, a subsequent operation may have unexpected behavior. For example, if opening a file fails, a write operation may not be issued. I/O operation errors do not stop program execution and therefore cannot be handled asynchronously with REACTE.

When reading a file of unknown length, IOSTAT is the only method to determine when the end of the file is reached.

Parameter

Parameters	Description
lun	Real-valued expression whose integer value is the logical unit number for the I/O device of interest. Refer to *3-4-11 ATTACH* on page 3-279 for information on the logical unit numbers recognized by the V+ system and how logical units are associated with I/O devices.

Parameters	Description	
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. 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.
	3	Status of any outstanding write request.

Details

Unlike most V+ keywords, I/O associated keywords 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.
The value returned for modes 0, 1, and 3 is shown in the table below.

IOSTAT Value Returned on EOF	Description
1	Normal success - for mode 3, this value indicates that no write request is outstanding.
0	Operation not yet complete
< 0	Standard V+ error number. Refer to *V+ User's Manual (Cat. No. I671)* for error number information.

Examples

The following example attempts to open a file for reading and ensures the file exists. If the file does exist, it will return its size in bytes to the variable "file.size". If the file does not exist, a message "Error opening file" is returned.

```
ATTACH (dlun, 4) "DISK"
FOPENR (dlun) "RECORD.DAT"
IF IOSTAT(dlun) < 0 THEN
    TYPE "Error opening file"
    HALT
END
file.size = IOSTAT(dlun,2)
```

The following example will read and display records until the end of the file is reached.

```
ieeof = -504
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 whether 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 1:
    TYPE "Data received. Handle =", handle
VALUE 100:
    TYPE "New connection established. Handle =", handle
VALUE 101:
    TYPE "Connection closed. Handle =", handle
VALUE -526:
    WAIT
ANY
    TYPE "Error during READ: ", $ERROR(status)
    GOTO 100
END
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-49 FCLOSE on page 3-337
3-4-50 FCMND on page 3-338
3-4-53 FEMPTY on page 3-343
3-4-56 FOPEN on page 3-348
3-4-62 FSEEK on page 3-360
3-4-106 READ on page 3-421
3-4-140 WRITE on page 3-466

3-1-59 LAST

Real-valued function that returns the highest index used for an array (dimension).

Syntax

```
LAST (array_name[])
```

Usage Considerations

If an automatic variable is referenced, this function returns the index specified in the AUTO statement that declared this array, regardless of which elements have been assigned values.

Parameter

Parameter	Description
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.

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.
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. There cannot be an index specified to the right of an omitted index.
The value returned by this function is an index, not an array element. 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.
The value of -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. For example, LAST(a[20,]) returns -1 if LAST(a[,]) returns 19 because 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

If the array part[ ] has all its elements defined from part[0] through part[10], the following example returns the value 10 (not 11, the number of elements defined).

```
LAST(part[])
```

If the given two-dimension array has elements [2,0], [2,3], and [2,5] defined, the following example returns the value 5 regardless of the status of elements [i,j] for i other than 2.

```
LAST($names[2,])
```

Related Keywords

3-4-12 AUTO on page 3-283

3-1-60 LATCH

Transformation function that returns a transformation value representing the location of the robot at the occurrence of the last external trigger.

Syntax

```
LATCH(select)
```

Usage Considerations

LATCH(0) returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Parameter

Parameter	Description
select	Optional integer, expression, or real variable specifying the following values. • 0: Robot position latch of currently selected robot (default). • n: Robot position latch of robot n.

Details

LATCH 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 from the Sysmac Studio or ACE software. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Additional Information

The DEVICE function may be used to read the latched value of an external encoder

Related Keywords

3-1-28 DEVICE on page 3-39
3-1-61 LATCHED on page 3-88
3-4-25 CLEAR.LATCHES on page 3-304
3-1-79 #PLATCH on page 3-102

3-1-61 LATCHED

Real-valued function that returns the status of the position latch and which input triggered it.

Syntax

```
LATCHED(select)
```

Usage Considerations

A maximum of 8 signals can be associated with a single belt encoder.
Configuration of the position latch signal ranges can be set from the V+ System Configuration Editor in the Sysmac Studio or ACE software. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Parameter

Parameter	Description
select	Integer,expression, or real variable that determines whether any latches have occurred since the last time the function was executed. • 0: Returns latch information for currently selected robot • -n (<0): Returns latch information for belt encoder n • +n (>0): Returns latch information for robot n

Details

This function returns a non-zero value if a position latch occurred while the robot location or belt encoder position has been latched since the LATCHED function was last used. The following values are returned where n represents any digital input signal on the controller.

- 0:No latch event has been detected.
- n: A rising edge was detected where n represents the digital input signal on the controller.
- -n: A falling edge was detected where n represents the digital input signal on the controller.

When this function returns a non-zero value, the data for the latch event is made available for retrieval with the following keywords.

- DEVICE- Returns position information for the external encoder.
- BELT- Returns position information for the external encoder.
- LATCH- Returns robot location information as a transformation value.

• #PLATCH- Returns robot location information as a precision point value

memo:

After one or multiple non-zero values are returned by this function and the latch buffer is empty, subsequent use of the function returns the value FALSE until the next occurrence of a latch trigger.

Related Keywords

3-1-28 DEVICE on page 3-39
3-1-11 BELT on page 3-16
3-1-34 ENCLATCH on page 3-44
3-1-60 LATCH on page 3-87
3-4-25 CLEAR.LATCHES on page 3-304
3-1-79 #PLATCH on page 3-102

3-1-62 LEN

Real-valued function that returns the number of characters in the given string.

Syntax

```
LEN (string)
```

Parameter

Parameter	Description
string	String constant, variable, or expression whose length is to be computed.

Examples

The following example returns the number of characters in the string "$str" to variable "str.len" with a value of 5.

```
$str = "Hello"
str.len = LEN($str)
```

3-1-63 $LNGB

String function that returns a 4-byte string containing the binary representation of a 32-bit integer.

Syntax

```
$LNGB (value, littleEndian)
```

Usage Considerations

Real values are rounded and any fractional part is lost. Values must be in the range ^H7FFFFFFF to ^H80000000

Parameter

Parameter	Description
value	Real-valued expression, the value of which is converted to its binary representation.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

The primary use of this function is to convert integer values to binary representation within an output record of a data file.
The integer part of a real value is converted into its binary representation. The low 32-bits of that binary representation are assembled into a string as four 8-bit characters. Bits 25 to 32 are assembled into the first byte, followed by bits 17 to 24 in the second byte, and so forth.
The operation performed by this function is equivalent to the following statement.

```
$CHR(INT(value/^H1000000) BAND ^HFF)+ $CHR(INT(value/^H10000) BAND ^HFF)+
$CHR(INT(value/^H100) BAND ^HFF)+ $CHR(INT(value) BAND ^HFF)
```

Examples

The example demonstrates how to encode and decode a V+ numeric variable. The V+ variable (int32_value) is encoded to a 32-bit integer binary representation using the $LNGB keyword and returned as a 4-byte string ($s). Then, the string is decoded to into the original signed 32-bit integer using the LNGB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
int32_value = -1.23456789E+08
TYPE "Original 'int32_value': ", int32_value

little.endian = 1
$s = $LNGB(int32_value,little.endian)

TYPE "int32_value encoded (Little Endian):    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))

TYPE "Simulate receiving 4 bytes:    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))

decoded_int32 = LNGB($s,,little.endian)

TYPE "Decoded 'decoded_int32' from 4 bytes:     ", decoded_int32
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-41 $FLTB on page 3-59
3-1-55 $INTB on page 3-79
3-1-64 LNGB on page 3-91
3-1-114 TRANSB on page 3-147

3-1-64 LNGB

Real-valued function that returns the value of four bytes of a string interpreted as a signed 32-bit binary integer.

Syntax

```
LNGB ($string, first_char, littleEndian)
```

Usage Considerations

Single-precision numbers are stored internally with only 24 bits of significance and input values that contain more than 24 significant bits are converted with some loss in precision.
Double-precision numbers are stored with 32 bits of significance with the most- significant bit as the sign bit and are converted with no loss of precision.
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+.

Parameter

Parameter	Description
$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).
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

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, and so on.
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 on.

Examples

The example demonstrates how to encode and decode a V+ numeric variable. The V+ variable (int32_value) is encoded to a 32-bit integer binary representation using the $LNGB keyword and returned as a 4-byte string ($s). Then, the string is decoded to into the original signed 32-bit integer using the LNGB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
int32_value = -1.23456789E+08
TYPE "Original 'int32_value': ", int32_value

little.endian = 1
$s = $LNGB(int32_value,little.endian)

TYPE "int32_value encoded (Little Endian):    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))

TYPE "Simulate receiving 4 bytes:    ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))

decoded_int32 = LNGB($s,,little.endian)

TYPE "Decoded 'decoded_int32' from 4 bytes:     ", decoded_int32
```

Related Keywords

3-1-5 ASC on page 3-11
3-1-22 DBLB on page 3-30
3-1-42 FLTB on page 3-60
3-1-56 INTB on page 3-80
3-1-63 $LNGB on page 3-89
3-1-114 TRANSB on page 3-147
3-1-122 VAL on page 3-156

3-1-65 MAX

Real-valued function that returns the maximum value contained in the list of values.

Syntax

```
MAX (value, ..., value)
```

Parameter

Parameter	Description
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. For example, the value -10 is considered larger than -100.

Examples

The following example sets "max.value" to the largest value of the variables x, y, and z, or to 0 if all three variables have values less than 0.

```
max.value = MAX(x, y, z, 0)
```

Related Keywords

3-1-67 MIN on page 3-94
3-1-74 OUTSIDE on page 3-99

3-1-66 $MID

String function that returns a substring of the specified string.

Syntax

```
$MID (string, first_char, num_chars)
```

Parameter

Parameter	Description
$string	String variable, constant, or expression from which the substring is extracted.
first_char	Optional real-valued expression that specifies the first character of the sub-string.
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 returns 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 consists of only the characters up to the end of the input string. No error results and the output string is not extended to the requested length.

Examples

The following example result in the string variable "$substring" containing the string cd. cd is the 2-character string that starts at character position 3 of the string abcde contained in the string variable "$string".

```
$string = "abcdef"
$substring = $MID($string, 3, 2)
```

Related Keywords

3-1-24 $DECODE on page 3-33
3-1-121 $UNPACK on page 3-155

3-1-67 MIN

Real-valued function that returns the minimum value contained in the list of values.

Syntax

```
MIN (value, ..., value)
```

Parameter

Parameter	Description
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.

Examples

The following example sets "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.

```
min.value = MIN(1000, x, y, z)
```

Related Keywords

3-1-65 MAX on page 3-92
3-1-74 OUTSIDE on page 3-99

3-1-68 NETWORK

Real-valued function that returns network status and IP address information of the robot controller.

Syntax

```
NETWORK (component, code)
```

Parameter

Parameters	Description
component	Real-valued expression that identifies the component of the network that is of interest. • 1 = TCP • 3 = reserved for future use
code	Optional real-valued expression that identifies the information desired where ADn is the nth byte of the IP address and NMn is the nth byte of the Network Mask. This value is only used when component = 1. 0 = Return status value as described below (default). 1 = Return AD1*256 + AD2 2 = Return AD3*256 + AD4 3 = Return NM1*256 + NM2 4 = Return NM3*256 + NM4 11 = Return AD1 12 = Return AD2 13 = Return AD3 14 = Return AD4 15 = Return NM1 16 = Return NM2 17 = Return NM3 18 = Return NM4

Details

This function returns one of the following values if status is requested (code = 0).

Value	Meaning
0	Hard ware not present
-1	Hardware present
1	Driver is running

Examples

The following example will display the last octet of the robot controller's IP address in the Monitor Window.

```
ip[1]=NETWORK(1,14)
$last.octet=$ENCODE(ip[1])
TYPE $last.octet
```

3-1-69 NORMAL

Transformation function that corrects a transformation for any mathematical round-off errors.

Syntax

```
NORMAL (transformation_value)
```

Parameter

Parameter	Description
transformation_value	Transformation,transformation valued function,or compound transformation whose value is to be normalized.

Details

The NORMAL function returns a transformation value that is similar to the input argument but has the orientation portion of the value corrected for any small accumulation of computational errors that may have occurred.
Use this function after a lengthy series of computations that modifies a trans- formation 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. Because of the small inaccuracies that occur in computer computations after being incrementally modified many times, these vectors can become non-perpendicular or not of unit length.

Examples

The following example calculates a local transformation relative to the current robot location and returns the value to the variable "normal.loc".

```
SET normal.loc=NORMAL(HERE)
```

3-1-70 NOT

Operator that performs logical negation of a value.

Syntax

```
... NOT value ...
```

Details

The NOT operator operates on a single value, converting it from logically true to false, and vice versa. If the single value is 0, a -1.0 (TRUE) is returned. Otherwise, a 0.0 (FALSE) value is returned.

Examples

The following example, if the variable "initialized" has a false value, the keywords in the IF structure will be executed.

```
IF NOT initialized THEN
CALL appl.setup()
initialized = TRUE
END
```

In the following example, the value of 40 is interpreted as logically TRUE since it is nonzero. The statement below returns a value of 0.0 (FALSE).

```
NOT 40
```

Related Keywords

3-1-17 COM on page 3-23

3-1-71 NULL

Transformation function that returns a null transformation value (one with all zero components).

Syntax

NULL

Usage Considerations

The trajectory of the robot in the emulation mode is considered to be the same as the current target position. Therefore, unlike with an actual robot, no positioning error occurs when the operation is completed.

Details

A null transformation corresponds to a null vector (X = Y = Z = 0) and no rotation (yaw = pitch = roll = 0). A null transformation is useful with a SHIFT function to create a transformation representing a translation with no rotation for example.

Examples

The following example will define the 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
```

The following example will determine the length of the vector described by the transformation "test.loc".

```
dist = DISTANCE(NULL, test.loc)
```

Related Keywords

3-1-18 CONFIG on page 3-24
3-1-71 NULL on page 3-97
3-1-90 SHIFT on page 3-112

3-1-72 OFF

Real-valued function that returns the value used by V+ to represent a logical false result.

Syntax

OFF

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.

Examples

The following example executes any statements between the DO and UNTIL keywords until the system switch POWER turns OFF

```
DO
    ;statement
    ;statement
    ;statement
UNTIL SWITCH(POWER) = = OFF
```

Related Keywords

3-1-38 FALSE on page 3-51
3-1-73 ON on page 3-98

3-1-73 ON

Real-valued function that returns the value used by V+ to represent a logical true result.

Syntax

ON

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.

Examples

The following example displays "System switch POWER is ON" in the Monitor Window when the system switch POWER is enabled.

```
IF(SWITCH(POWER)==ON) THEN
    TYPE "System switch POWER is ON"
END
```

Related Keywords

3-1-72 OFF on page 3-98
3-1-115 TRUE on page 3-149

3-1-74 OUTSIDE

Real-valued function that tests a value to determine if it is outside a specified range.

Syntax

```
OUTSIDE (low, test, high)
```

Parameter

Parameter	Description
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.

Examples

The following example evaluates the value of "angle" and displays "Angle is out of reach." if it is outside of the values of 0 to 180.

```
IF OUTSIDE(0,angle,180) THEN
    TYPE "Angle is out of reach."
END
```

Related Keywords

3-1-65 MAX on page 3-92
3-1-67 MIN on page 3-94

3-1-75 PARAMETER

Real-valued function that returns the current setting of the named system parameter.

Syntax

```
PARAMETER (parameter_name)
PARAMETER (parameter_name[index])
```

Parameter

Parameter	Description
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.

Examples

The following example illustrates how the current setting of the BELT.MODE system parameter can be displayed on the Monitor Window during program execution:

```
TYPE "The BELT.MODE parameter is set to", PARAMETER (BELT.MODE)
```

Related Keywords

3-5-1 BELT.MODE on page 3-468
3-5-4 NOT.CALIBRATED on page 3-471
3-2-45 PARAMETER on page 3-229
3-4-92 PARAMETER on page 3-400

3-1-76 #PDEST

Precision-point function that returns a precision-point value representing the planned destination location for the current robot motion.

Syntax

#PDEST

Usage Considerations

The #PDEST function returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Details

The #PDEST function can be used to determine the robot's destination before its motion was interrupted.
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. Refer to *3-1-27 DEST* on page 3-37 for more information on the use of both the #PDEST and DEST functions.

Examples

The following example displays the joint values at the destination location "point1".

```
MOVE point1
DECOMPOSE j[]=#PDEST
TYPE j[0],j[1],j[2],j[3],j[4],j[5]
```

Related Keywords

3-1-27 DEST on page 3-37
3-1-48 HERE on page 3-68
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-1-77 #PHERE

Precision-point function that returns a precision-point value representing the current location of the currently selected robot.

Syntax

#PHERE

Usage Considerations

The #PHERE function is considered to be a precision-point keyword and the "#" character must precede the function name whenever it is used.
PHERE is a reserved word in V+ and cannot be used for a variable.

Details

The PHERE real-valued function is equivalent to the HERE.

Examples

The following example uses #PHERE to set a precision point value "pp".

```
SET #pp = #PHERE
```

Related Keywords

3-1-48 HERE on page 3-68

3-1-78 PI

Real-valued function that returns the value of the mathematical constant pi (3.141593).

Syntax

```
PI
```

Usage Considerations

TYPE, PROMPT, and similar commands display the result of the example below as a single-precision value. However, pi is stored and manipulated as a double- precision value. The LISTR monitor command displays real values to full precision.

Examples

The following example returns the circumference of a semi-circle using values of "arc_length" and "radius".

```
arc_length=PI*radius
```

3-1-79 #PLATCH

Precision-point function that returns a precision-point value representing the location of the robot at the occurrence of the last external trigger.

Syntax

```
#PLATCH(select)
```

Usage Considerations

The function name #PLATCH is considered to be a precision-point keyword and the "#" character must precede all uses of the function.
#PLATCH(0) returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Parameter

Parameter	Description
select	Optional integer, expression, or real variable specifying the following values. • 0: Robot position latch of currently selected robot (default) • n: Robot position latch of robot n

Details

#PLATCH returns a precision-point value that represents the location of the robot when the last trigger occurred. The LATCHED 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 from the V+ System Configuration Editor in the Sysmac Studio or ACE software. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Related Keywords

3-1-60 LATCH on page 3-87
3-1-61 LATCHED on page 3-88

3-1-80 POS

Real-valued function that returns the starting character position of a substring in a string.

Syntax

```
POS (search_string, sub_string, start)
```

Parameter

Parameter	Description
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

The following example returns 5.

```
POS("file.ext", ".")
```

The following example returns 0.

```
POS("file", ".")
```

The following example returns 4.

```
POS("abcdefgh", "DE")
```

The following example returns 6.

```
POS("1-2-3-4", "-", 5)
```

3-1-81 #PPOINT

Precision-point function that returns a precision-point value composed from the given components.

Syntax

```
#PPOINT (j1_value, j2_value, j3_value, j4_value, j5_value, j6_value, j7_value, j8_
value, j9_value, j10_value, j11_value, j12_value)
```

Usage Considerations

The #PPOINT function name is considered to be a precision-point name. The "#" character must precede all uses of the function.
If more values are specified than the number of robot joints, the extra values are ignored.
A0 value is assumed for any parameter that is omitted.

Parameter

Parameter	Description
j1_value	Optional real-valued expressions for the respective robot joint positions.
j2_value	
j3_value	
...	
j12_value	

Details

Returns a precision-point value composed from the given components, which are the positions of the first through last robot joints, respectively.

Examples

The following example assumes 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 statements define the current location and fill an array with components.

```
HERE #ref
DECOMPOSE x[] = #ref
```

The following statement moves the robot to the new precision points defined with modified components.

```
MOVE #PPOINT(x[0], x[1]+a, x[2]-a/2, x[3])
```

The following statements lead to the same final location, but robot joints 2 and 3 are not moved simultaneously with this method.

```
DRIVE 2, a, 100
DRIVE 3, -a/2, 100
```

Related Keywords

3-4-31 DECOMPOSE on page 3-312
3-1-112 TRANS on page 3-144

3-1-82 PRIORITY

Real-valued function that returns the current reaction lock-out priority for the program.

Syntax

PRIORITY

Usage Considerations

This function returns the reaction lock-out priority, not the program priority of the executing program.

Details

The PRIORITY function can be used to determine the current setting of the reaction lock-out priority for the task executing the function.
The reaction lock-out priority for each program task is set to 0 when execution of the task is initiated. The priority can be changed by the program at any time with the LOCK command, or the priority is set automatically when a reaction occurs as prescribed by a REACT or REACTI commands.

Examples

The following 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

IF save < 10 THEN
    LOCK 10
END
LOCK save
```

Related Keywords

3-4-77 LOCK on page 3-379
3-4-103 REACT on page 3-415
3-4-105 REACTI on page 3-419

3-1-83 RANDOM

Real-valued function that returns a pseudo-random number.

Syntax

RANDOM

Details

Returns a pseudo-random number in the range 0.0 to 1.0, inclusive. Each time the RANDOM function is evaluated, it returns a different value.
The numbers generated by this function are pseudo-random because the sequence repeats after this function has been called 2E24 times (16,777,216 times).

Examples

The following example will return a random number between 0.0 and 1.0 to the variable "random1".

```
random1 = RANDOM
```

3-1-84 RX

Transformation function that returns a transformation describing a rotation about the X-axis.

Syntax

```
RX (angle)
```

Parameter

Parameter	Description
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 X-axis and a 0 displacement (X, Y, Z = 0).

Examples

The following example creates a transformation "frame2" equivalent to "frame1" rotated around the X-axis by 45 degrees.

```
SET frame1 = TRANS(100,100,-960,0,180,0)
SET frame2 = frame1:RX(45)
```

Related Keywords

3-1-31 DX on page 3-42
3-1-32 DY on page 3-43
3-1-33 DZ on page 3-44
3-1-85 RY on page 3-107
3-1-86 RZ on page 3-108

3-1-85 RY

Transformation function that returns a transformation describing a rotation about the Y-axis.

Syntax

```
RY (angle)
```

Parameter

Parameter	Description
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 Y-axis and a 0 displacement (X, Y, Z = 0).

Examples

The following example creates a transformation "frame2" equivalent to "frame1" rotated around the Y-axis by 45 degrees.

```
SET frame1 = TRANS(100,100,-960,0,180,0)
SET frame2 = frame1:RY(45)
```

Related Keywords

3-1-31 DX on page 3-42
3-1-32 DY on page 3-43
3-1-33 DZ on page 3-44
3-1-84 RX on page 3-107
3-1-85 RY on page 3-107

3-1-86 RZ

Transformation function that returns a transformation describing a rotation about the Z-axis.

Syntax

```
RZ (angle)
```

Parameter

Parameter	Description
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 Z-axis and a 0 displacement (X, Y, Z = 0).

Examples

The following example creates a transformation "frame2" equivalent to "frame1" rotated around the Z-axis by 30 degrees.

```
SET frame1 = TRANS(0,0,0,0,0,30)
SET frame2 = frame1:RZ(30)
```

Related Keywords

3-1-31 DX on page 3-42
3-1-32 DY on page 3-43
3-1-33 DZ on page 3-44
3-1-84 RX on page 3-107
3-1-85 RY on page 3-107

3-1-87 SCALE

Transformation function that returns a transformation value equal to the transformation parameter with the position scaled by the scale factor.

Syntax

```
SCALE (transformation BY scale_factor)
```

Parameter

Parameter	Description
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.

Examples

The following example will result in the transformation y receiving the value of (250, 187.5, 125, 10, 20, 30) if the original transformation x has the value (200, 150,100,10, 20, 30).

```
SET y = SCALE(x BY 1.25)
```

Related Keywords

3-1-90 SHIFT on page 3-112

3-1-88 SELECT

Real-valued function that returns information about the device specified for the currently selected task.

Syntax

```
SELECT (device_type, mode)
```

Usage Considerations

The SELECT keyword is only available for use with Robot Integrated Control systems.

Parameter

Parameter	Description
device_type	Parameter that identifies the type of device that is to be selected. The only valid device_type is ROBOT.
mode*1	Optional real value, variable, or expression interpreted as an integer that specifies the device_type number for the function. If this parameter is omitted or has the value 0, the function returns the number of the device_type currently selected or 0 if no device_type is selected. If mode has the value -1, the function returns the total number of units available and communicating on the EtherCAT network.

*1. If a previously selected robot becomes disconnected from the EtherCAT network, the statement SELECT (ROBOT) will return a 2 even if the EtherCAT connection has been restored. The previously selected robot is not updated automatically.

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. It also can be used to retrieve the state of a specified robot.
If the V+ system is not configured to control a robot, the selected robot is always1 and the total number of robots is 0.
SELECT(ROBOT)returns the number of the currently selected robot.
SELECT(ROBOT,-1) returns the number of robots that are communicating with EtherCAT.
SELECT(ROBOT, rob_num), where "rob_num" represents a specific robot, returns one of the following values to represent the state of the robot.

- 0: Not Configured - the robot is not in the NJ-series Robot Integrated CPU Unit configuration.

- 1: Not Synchronized - the EtherCAT configuration of the robot and the NJ-series Robot Integrated CPU Unit does not match. The robot did not connect during normal startup.
- 2: Not Connected - The robot is not connected to the EtherCAT network.
- 3: Ready - The robot is configured, synchronized, and connected.

Additional Information

Refer to *V+ User's Manual (Cat. No. I671)* for information about associated errors -622, -508, and -315.

Examples

The following example returns the unit number of the robot selected for the current task to the variable "our.robot".

```
our.robot = SELECT(ROBOT)
```

The following example returns the total number of robots connected to the controller to the variable "num.robots".

```
num.robots = SELECT(ROBOT,-1)
```

Related Keywords

3-2-53 SELECT on page 3-237 (monitor command)
3-4-115 SELECT on page 3-431 (program command)

3-1-89 #SETPOINT

Precision point function that returns the commanded joint-angle positions computed by the trajectory generator during the last trajectory-evaluation cycle.

Syntax

```
#SET.POINT
```

Usage Considerations

The name "set.point" cannot be used as a program.

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.

Examples

The following example will save the robot joint values every 100 milliseconds to the array "set.points".

```
MOVE point2
i = 0
WHILE STATE(2) == 1 DO
    DECOMPOSE angles[i,] = #SET.POINT
    i = i+1
    WAIT.EVENT , 0.1
END
```

3-1-90 SHIFT

Transformation function that returns a transformation value resulting from shifting the position of the transformation parameter by the given shift amounts.

Syntax

SHIFT (transformation BY x_shift, y_shift, z_shift**)**

Parameter

Parameter	Description
transformation	Transformation expression that is to be shifted.
x_shift	Optional real-valued expressions that are added to the respective position components of the transformation parameter.
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 0.

Examples

The following example will result in the transformation y receiving the value (205,145, 110, 10, 20, 30) if the original transformation x has the value of (200,150,100, 10, 20, 30).

```
SET y = SHIFT(x BY 5,-5,10)
```

Related Keywords

3-1-87 SCALE on page 3-109
3-1-112 TRANS on page 3-144

3-1-91 SIG.INS

Real-valued function that returns an indication of whether a digital I/O signal is installed in the system or whether a software signal is available in the system.

Syntax

```
SIG.INS (signal_num)
```

Usage Considerations

The SIG.INS function keyword will return TRUE if the host signal is available and FALSE if it is not available (not mapped).
If the SIG.INS function is used to determine the status of digital I/O associated with an IOBlox unit that is either not configured or does not physically exist, errors will occur as described in the details section below.

Parameter

Parameter	Description
signal_num	Real-valued expression that defines the number of the digital I/O or software signal to check. The absolute value is used and 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 0. This function can be used to ensure the digital I/O signals are installed as expected by the application program.
Executing this function for an IOBlox signal that does not physically exist will cause a -508 *Device not ready* error.
Executing this function for an IOBlox signal that is not configured will cause a - 405 *Illegal digital signal* error.

Additional Information

When using the Emulator, executing this function for an IOBlox signal that is not configured will cause a -405 *Illegal digital signal* error.

Examples

The following example checks whether digital I/O signal 12 is installed as an input signal (referenced as signal 1012). A message is displayed on the Monitor Window 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

3-2-4 BITS on page 3-171 (monitor command)
3-4-16 BITS on page 3-290 (program command)
3-1-12 BITS on page 3-17 (real-valued function)
3-2-32 IO on page 3-211
3-2-50 RESET on page 3-235
3-4-114 RUNSIG on page 3-429
3-2-54 SIGNAL on page 3-238 (monitor command)
3-4-120 SIGNAL on page 3-437 (program command)

3-1-92 SIGN

Real-valued function that returns the value 1, with the sign of the value parameter.

Syntax

SIGN (value)

Parameter

Parameter	Description
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.

Examples

The following example returns 1.0.

```
SIGN(0)
```

The following example returns 1.0.

```
SIGN(0.123)
```

The following example returns -1.0.

```
SIGN(-5.462)
```

The following example returns 1.0.

```
SIGN(1.3125E+2)
```

3-1-93 SIG

Real-valued function that returns the logical AND of the states of the indicated digital signals.

Syntax

```
SIG (sigal_num, ..., signal_num)
```

Parameter

Parameter	Description
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. SIG will return TRUE if all the specified signal states are TRUE. Otherwise, SIG will return FALSE.

Refer to the supporting robot user manual and *V+ User's Manual (Cat. No. I671)* for more information about signal numbers for your particular robot.

Only digital signals that are present on the system can be used. You can use the IO monitor command or the SIG.INS function to check the current digital I/O configuration. Signals 3001 and 3002 refer to the robot selected by the current task. Signal 3001 is the state of the hand-close solenoid. Signal 3002 is the state of the hand-open solenoid.

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. If the signal_num parameter is zero SIG will return a value of TRUE.

Examples

The examples below it is assumed 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 example returns -1.0 (TRUE).

```
SIG(1001)
```

The following example returns 0.0 (FALSE).

```
SIG(-1004)
```

The following example returns 0.0 (FALSE).

```
SIG(1001,1004)
```

The following example returns -1.0 (TRUE).

```
SIG(1001,-1004)
```

Related Keywords

3-2-4 BITS on page 3-171 (monitor command)
3-4-16 BITS on page 3-290 (program command)
3-1-12 BITS on page 3-17 (real-valued function)
3-2-32 IO on page 3-211
3-4-109 RESET on page 3-425
3-4-114 RUNSIG on page 3-429
3-4-120 SIGNAL on page 3-437 (program command)
3-4-120 SIGNAL on page 3-437 (monitor command)

3-1-94 SIN

Real-valued function that returns the trigonometric sine of a given angle.

Syntax

```
SIN (value)
```

Usage Considerations

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

Parameter

Parameter	Description
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 is always in the range of -1.0 to +1.0, inclusive.

Examples

The following example returns 2.146753E-03.

```
SIN(0.123)
```

The following example returns -0.09518556.

```
SIN(-5.462)
```

The following example returns .5.

```
SIN(30)
```

Additional Information

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

Related Keywords

3-1-6 ASIN on page 3-12
3-1-19 COS on page 3-27
3-1-3 ACOS on page 3-9
3-1-103 TAN on page 3-130
3-1-7 ATAN2 on page 3-13

3-1-95 SOLVE.FLAGS

Real-valued function that returns bit flags representing the robot configuration specified by an array of joint positions.

Syntax

```
SOLVE.FLAGS (joints[index])
```

Usage Considerations

The SOLVE.FLAGS 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 causes an error.

Parameter

Parameter	Description
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 0 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 command.
The bits of the value returned by this function are interpreted as described below.
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.

Related Keywords

3-4-2 ABOVE on page 3-269
3-4-15 BELOW on page 3-289
3-4-31 DECOMPOSE on page 3-312
3-4-75 LEFTY on page 3-376
3-4-113 RIGHTY on page 3-428
3-4-55 FLIP on page 3-346
3-4-85 NOFLIP on page 3-392
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-122 SOLVE.ANGLES on page 3-439
3-4-123 SOLVE.TRANS on page 3-444

3-1-96 SPEED

Real-valued function that returns one of the system motion speed factors.

Syntax

```
SPEED (select)
```

Usage Considerations

The SPEED function returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Parameter

Parameter	Description
select	Real-valued expression whose value determines which speed factor should be returned.

Details

This function returns the system motion speed factor corresponding to the select parameter value. The value returned should be interpreted as a percentage of normal speed even if the program speed was set by a SPEED program command that specified a speed setting.
The acceptable parameter values and the corresponding speed values returned are described below.

Select	Speed value returned
1	Monitor speed.
2	Permanent program speed (set by a SPEED program command).
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.

Examples

The following example makes one motion at 1/2 of the permanent program speed

```
new.speed = SPEED(2)/2 SPEED new.speed
MOVE pick.up
```

The following example has same operation as the example above.

```
SPEED SPEED(2)/2
MOVE pick.up
```

Related Keywords

3-1-2 ACCEL on page 3-8
3-1-30 DURATION on page 3-41
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-115 SELECT on page 3-431 (program command)
3-2-55 SPEED on page 3-240 (monitor command)
3-4-124 SPEED on page 3-445 (program command)

3-1-97 SQRT

Real-valued function that returns the square root of the parameter.

Syntax

```
SQRT (value)
```

Parameter

Parameter	Description
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 0. An error results if the argument is less than 0.

Examples

The following example returns 0.3507136.

```
SQRT(0.123)
```

The following example returns 2.0.

```
SQRT(4)
```

The following example returns *Negative square root*.

```
SQRT(-5.462)
```

The following example returns 11.45644.

```
SQRT(1.3125E+2)
```

Additional Information

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

Related Keywords

3-1-98 SQR on page 3-120

3-1-98 SQR

Real-valued function that returns the square of the parameter.

Syntax

```
SQR (value)
```

Parameter

Parameter	Description
value	Real-valued expression whose value is to be squared.

Details

This function computes the square of a value. The result is equal to (value * value).

Examples

The following example returns 0.015129.

```
SQR(0.123)
```

The following example returns 16.

```
SQR(4)
```

The following example returns 29.8334.

```
SQR(-5.462)
```

The following example returns 17226.56 .

```
SQR(1.3125E+2)
```

Additional Information

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

Related Keywords

3-1-97 SQRT on page 3-120

3-1-99 STATE

Real-valued function that returns a value to provide information about the robot system state.

Syntax

```
STATE (select)
```

Usage Considerations

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

Additional Information

Refer to *V+ User's Manual (Cat. No. I671)* for more information about robot states.

Parameter

Parameter	Description
select	Real value, variable, or expression interpreted as an integer that selects the category of state information returned. The information categories are listed below. Refer to *Details* on page 3-122 for more information. • 1: Overall robot state • 2: Current or previous motion • 3: Current manual control mode • 4: Controller interface panel settings and hardware status • 5: Front Panel operating mode selection • 6: Alter mode status for current motion • 7: Alter mode status for next motion • 8: Robot connected to pendant • 9: Time until completion of robot motion • 10: Percentage of current motion completed • 11: Which portion of the acceleration profile is currently being generated • 12: ALTER command status • 13: Trajectory generator execution rate in Hz • 14: Reserved for future use • 15: Number of the motion being executed • 16 to 18: Reserved for future use • 19: Reserved for future use (returns *Invalid argument* -407) • 20: Acceleration ramp up time • 21: Constant acceleration time • 22: Acceleration ramp down time • 23: Constant velocity time • 24: Deceleration ramp up time • 25: Constant deceleration time • 26: Deceleration ramp down time • 27: Total motion time • 28 to 29: Reserved for future use • 30: State of Front Panel indicator • 31 to 29: Reserved for future use • 40: Belt tracking status

Details

When select = 1, the function returns information about the overall robot state as follows.

Value	Description
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 to turn ON robot power.
3	Robot power was turned ON and initialization is occurring.
4	Indicates that Manual control mode is active. Refer to select =3.
5	A CALIBRATE command is executing.
6	Reserved for future use.
7	Robot is under program control.
8	Robot power is ON and robot is not calibrated and cannot be moved.
9	Reserved for future use.
10	Front panel power light is blinking slowly (1 Hz) and waiting to be pressed. Robot power will be turned ON when the blinking button is pressed.
11	Front panel power light is blinking rapidly (4 Hz). Robot power will be turned ON when the COMP/PWR button on the pendant is pressed.*1

*1. Pressing the front panel power light when it is blinking rapidly will cause robot power to be turned OFF. The normal process will be required to turn power back ON.

When select = 2, the function returns information about the current or previous robot motion as follows. These modes can change only when the robot is under program control (when STATE(1) = 7).

Value	Description
0	No motion operations executed yet.
1	Normal trajectory evaluation is in progress (including normal acceleration, deceleration and segment transitions).
2	Motion stopped at a planned location. A RETRY command has no effect.
3	Position error is being nulled at unplanned final location.
4	Motion stopped at an unplanned location due to a belt window violation. A RETRY command completes the previous motion.
5	Decelerating due to a triggered REACTI or BRAKE operation.
6	Stopped due to a triggered REACTI or BRAKE command. A RETRY command completes the previous motion.
7	Decelerating due to a hardware error or ESTOP operation.
8	Stopped due to a hardware error or ESTOP operation. A RETRY command completes the previous motion.
9	Decelerating due to a stop-on-force condition.
10	Stopped due to a stop-on-force condition.

When select = 3, the function returns information about the current manual control mode as follows (refer to the *3-4-72 JOG* on page 3-371 program command for more information).

Value	Description
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.
6	Reserved for future use.
7	JogTo mode.
8	Align mode.
9	Frame mode.

When select = 4, the returned value returns information about the hardware status to be read by programs. Interpret the value as a set of bit flags, each of which indicates a corresponding condition.

Value	Description
^H1	Reserved for future use.
^H2	Reserved for future use.
^H4	ESTOP circuit is open (see note below).
^H8	HIGHPOWER button is pushed (see note below)
^H10	ESTOP channel 1 relay is open.
^H20	ESTOP channel 2 is open.
^H40	Front Panel key switch is in manual state.
^H80	Reserved for future use.
^H100 to ^HF00	ESTOP source: 0x0 = No ESTOP or pending (waiting for 120 ms settling period) 0x1 = ESTOP from loss of ESTOP source 0x2 = ESTOP from Front Panel 0x3 = ESTOP from Pendant 0x4 = ESTOP from User ESTOP 0x5 = ESTOP from Line ESTOP Input 0x6 = ESTOP from Muted Safety Gate (auto mode only) 0x7 = ESTOP from AUTO to Manual change 0x8 = ESTOP from Manual to AUTO change 0x9 - 0xE = Reserved for future use 0xF = Unresolved ESTOP source
^H1000	3-position Enable switch is closed (reported only in manual mode).

When select = 5, the returned value indicates the settings of the switch on the Front Panel.

Value	Description
1	Automatic mode.
2	Manual mode.

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 command 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 command that initiated the path modification.
When select = 8, the number of the robot connected to the manual control pendant is returned.
When select = 9, the function returns the time in seconds left until completion of the current motion. A returned value of 0 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 changes to the time left in the next motion. 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 percentage will be returned with an accuracy of 0.8%. 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 changes to reflect the start of the next motion. 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	Description
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 during the transition section between motions.
6	Constant acceleration during the transition section between motions.
7	Ramping down acceleration 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 command 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 commands with the processing of the data by the trajectory generator.
When select = 13, the function returns the trajectory generator execution rate in Hertz. If the trajectory generator is executed,, this function returns the value 62.5.
When select = 15, the function returns the number of the motion that is being executed by the selected robot. This number is 0 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 resets to 0.

The following program commands affect the value of STATE(15):

```
ALIGN, APPRO, APPROS, DEPART, DEPARTS, DRIVE, JMOVE, MOVE, MOVES, READY
```

Commands that affect one or more subsequent motions (e.g., ACCEL, AMOVE, ABOVE, BELOW, DURATION, FLIP, LEFTY, NOFLIP, MULTIPLE, RIGHTY, SINGLE, SPEED, TOOL, UNIDIRECT, ...) do not affect the value of STATE(15), because they do not actually initiate a motion. Function keywords do not affect the value of STATE(15), because these do not cause a motion. Location-valued functions (e.g., DEST, FRAME, INVERSE, SCALE, SHIFT, TRANS, etc.) simply compute location values.

When select is 20 through 27, the function returns detailed information on the planned execution time of the current motion (or the previously executed motion if the robot is stopped).

Unlike STATE(9) that returns the remaining motion execution time corrected for the monitor speed setting, the values returned by STATE(20) through STATE(27) are the planned values and are not affected by the setting of the monitor speed value. The values returned by these functions (in units of seconds) are:

select	Information returned
20	Acceleration ramp up time.
21	Constant acceleration time.
22	Acceleration ramp down time.
23	Constant velocity time.
24	Deceleration ramp up time.
25	Constant deceleration time.
26	Deceleration ramp down time.
27	Total motion time (sum of STATE(20) through STATE(26)).

When select = 30, the function returns the state of the Front Panel power light. The possible values returned by this function are described below.

Value	Description
0	Light is OFF.
1	Light is ON.
2	Light is blinking at 4 Hz.
3	Light is blinking at 1 Hz.

When select = 40, the function returns a flag that is set to nonzero when the currently selected robot is tracking a belt.

Examples

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
MOVES final BREAK
CASE STATE(2) OF
    VALUE 2:
        TYPE "Motion completed normally"
    VALUE 6:
        TYPE "Motion stopped by REACTI"
```

```
    VALUE 8:
        TYPE "Motion stopped by panic button"
END
```

Related Keywords

3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-58 STATUS on page 3-244 (monitor command)
3-1-100 STATUS on page 3-127 (real-valued function)

3-1-100 STATUS

Real-valued function that returns status information for an application program.

Syntax

STATUS (program_name)

Parameter

Parameter	Description
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, but the parameter cannot be omitted.

Details

This function returns information about the execution status of the specified program.
If a program is being executed by multiple tasks, the STATUS function returns -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 does not return -1 until all the tasks stop executing the program. The function returns a not defined status if an invalid program name is specified (for example, if the name does not start with a letter).
If no program name is specified (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. Refer to *V+ User's Manual (Cat. No. I671)* for more information about tasks and task numbers.
If a program name is specified as the function parameter, the status of that program is returned as described below

Value Returned	Program Status
-1	Not executing.
-2	Not defined.

Value Returned	Program Status
-3	Write-interlocked when the program is being copied, deleted, renamed, or edited in read-write mode.
-4	Not executable when the program contains a structure error or bad syntax.
-5	Read-interlocked when the program is executing by one or more tasks or is being edited in read-only mode.

Examples

The following example 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
```

The STATUS function does not return -1 if the program is being executed by any program task. This example may not be appropriate for some situations. Refer to the example shown for the *3-4-46 EXECUTE* on page 3-331 program command for another technique for initiating execution of another program task.

Related Keywords

3-1-26 DEFINED on page 3-36
3-1-99 STATE on page 3-121
3-1-100 STATUS on page 3-127
3-2-66 TESTP on page 3-257

3-1-101 STRDIF

Real-valued function that compares two strings byte-by-byte for the purpose of sorting.

Syntax

```
STRDIF ($a, $b)
```

Usage Considerations

This function always compares bytes exactly. It ignores the setting of the UPPER system switch.

Parameter

Parameter	Description
$a	A string constant, variable, or expression that contains the bytes to be compared with those in $b.

Parameter	Description
$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 regardless of the UPPER system switch setting. The two strings can have different lengths. The returned values and their meanings are described below.

Returned Value	Description
-1	$a is less than $b.
0	$a is exactly the same as $b.
1	$a is greater than $b.

Examples

The following example will sort two string variables in alphabetical order and then display the results with the TYPE command.

```
$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]
```

Related Keywords

3-6-12 UPPER on page 3-487

3-1-102 SWITCH

Real-valued function that returns information about the setting of a system switch.

Syntax

```
SWITCH (switch_name)
SWITCH (switch_name[index])
```

Parameter

Parameter	Description
switch_name	Name of the system switch of interest.
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.
Refer to *V+ User's Manual (Cat. No. I671)* for more information about system switches.

Examples

The following example checks whether the DRY.RUN switch is enabled. If it is, a message is displayed in the Monitor Window.

```
IF SWITCH(DRY.RUN) THEN
    TYPE "DRY RUN mode is enabled"
END
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)

3-1-103 TAN

Real-valued function that returns the trigonometric tangent of a given angle.

Syntax

```
TAN (value)
```

Usage Considerations

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

Parameter

Parameter	Description
value	Real-valued expression that defines the angular value to be considered.

Details

Returns the trigonometric tangent of the argument, which is assumed to have units of degrees.

Examples

The following example returns 0.5773503.

```
TAN(30)
```

The following example returns 1.

```
TAN(45)
```

Additional Information

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

Related Keywords

3-1-6 ASIN on page 3-12
3-1-19 COS on page 3-27
3-1-3 ACOS on page 3-9
3-1-94 SIN on page 3-116
3-1-103 TAN on page 3-130
3-1-7 ATAN2 on page 3-13

3-1-104 TAS

Real-valued function that returns the current value of a real-valued variable and assigns it a new value. The two actions are done indivisibly so that no other pro- gram task can modify the variable at the same time.

Syntax

```
TAS (variable, new_value)
```

Usage Considerations

The V+ system does not enforce any protection scheme for global variables that are shared by multiple program tasks. It is the programmer's responsibility to manage the usage of such global variables. The TAS function can be used to implement logical interlocks on access to shared variables.
This function can also be used to manage a restriction on the simultaneous access of global arrays by multiple program tasks. Program execution can fail if two or more tasks attempt to increase the size of an array at the same time. Refer to *V+ User's Manual (Cat. No. I671)* for more information.

Parameter

Parameter	Description
variable	Name of the real-valued variable to be evaluated 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

Different program tasks execute simultaneously which causes time sharing of the system processor, so 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 can be used to allow multiple V+ tasks to modify shared data structures. This function provides a way for a task to lock out others while the locking task modifies the data structures. 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 for example.
As an example of the use of shared variables, consider this V+ program that increments and decrements a global variable:

```
.PROGRAM tas_test()
    AUTO i
    FOR i = 1 TO 1E+06
        WHILE TAS(locked,TRUE) DO
            WAIT
        END
        counter = counter+1
        counter = counter-1
        locked = FALSE
    END
.END
```

If the variable "counter" starts at 0, and the program "tas_test" is run simultaneously in tasks 1 and 2, you might expect that counter will have the value 0 after execution completes but this is normally not correct. Since the two tasks are modifying the same variable at the same time, the value becomes corrupted.

This can be addressed by modifying the program to employ an interlock as shown below.

```
.PROGRAM tas_test()
AUTO i
FOR i = 1 TO 1E+06

WHILE TAS(locked,TRUE) DO WAIT
END

counter = counter+1 counter = counter-1
.END

END
```

When the program is executed simultaneously in two or more tasks, the value of the variable "counter" will always be the same as before the program starts because each task blocks the other task(s) while accessing the shared variable.
The global variable locked does not need to be initialized before the program is executed because the TAS function returns FALSE if the variable is not defined. The lock implicitly begins in the OFF state.
The lock should be released as soon as possible because the other task could be waiting for it to be released. Take care to make sure the lock is always released after it gets applied. Otherwise the other task could be blocked forever, and the current task would also be blocked the next time it tries to acquire the lock.

Examples

The following example demonstrates the key aspects of using the TAS function to ensure exclusive access by an application program to data that is also used by another program task. The same command sequence must be used in any other application program that needs to access the data.
The real-variable "data.locked" has the value FALSE when the data is not interlocked and the value TRUE when the data is interlocked. This variable is set to TRUE with the TAS function, to detect if the other program task has already set it to TRUE. Since TAS evaluates and sets the value indivisibly, both programs will not set "data.locked" to TRUE simultaneously without detecting a conflict.
Use of the semaphore variable "data.locked" involves the three steps shown below.

- Look for the lock variable to have the "unlocked" setting (FALSE) and simultaneously apply the "lock" setting (TRUE). This loop will cycle continuously until another task sets the lock variable to the "unlocked" setting (FALSE), at which time this task asserts the lock for itself as shown below.

```
WHILE TAS(data.locked,TRUE) DO
    WAIT
END
```

1. Perform desired operations accessing the shared data
2. Release the lock on the shared data structure.

```
data.locked = FALSE
```

The WHILE loop causes program execution to be blocked until the variable "data.locked" is found to have the value FALSE. The program is blocked if the other program has locked the semaphore variable in order to access the shared data. The TAS function will set the variable "data.locked" to TRUE each time the function is executed, but that will have no effect if the variable already has that value. Once the program gains exclusive access to the shared data, it can safely access the data.

The last statement in step 3 above releases the data for access by the application executing as the other program task.

Related Keywords

3-1-15 CAS on page 3-21

3-1-105 TASK

Real-valued function that returns information about a program execution task.

Syntax

```
TASK (select, task_num)
```

Usage Considerations

If select = 1 and a value of 3 is returned (TASK stopped due to ABORT), the PROCEED program command can be used to resume execution of this program. It is recommended to check that the program has stopped executing before checking this state. Refer to the *Examples* on page 3-135 section for more information.

Parameter

Parameter	Description
select	Optional real-valued expression that has a value of 0, 1, or 2 and selects the category of task information returned. 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.

Details

This function returns various information about the system program execution tasks. Refer to *V+ User's Manual (Cat. No. I671)* for more information about task execution.
The select parameter determines the type of information that is returned as described below.

Parameter	Description
select = 0	Task number: The function returns the number of the task executing the current program.

Parameter	Description		
select = 1	Task run state: Returns the run state for the task specified by the task_num parameter. The value returned should be interpreted as described below.		
	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		Task stopped due to ABORT.
	4		Executing
	5		Stopped due to PAUSE or breakpoint.
	7		Stopped due to single-step execution.
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_num parameter.		
	Bit Number	Bit Mask	Indication if bit is ON
	1	1	Reserved for future use.
	2	2	Task has robot attached.

Examples

The following example will check that the program has stopped executing after a value of 3 is returned for select = 1. The PROCEED program command is then used to resume execution of the program.

The following example will display the task number the program is running in the Monitor Window.

```
TYPE "This program is running as task number :", TASK()
```

The following example demonstrates how the TASK function can be used to decide whether to initiate execution of a program "named pc.job.2" with task 2

```
IF TASK(1,2) <> 4 THEN
    IF STATUS("pc.job.2") == -1 THEN
        EXECUTE 2 pc.job.2()
    ELSE
TYPE /B, "Can't start task 2"
    END
END
```

The following example checks if the current task has a robot attached and types a message in the Monitor Window based on the outcome.

```
IF (TASK(2,TASK(0))<>2)THEN
    TYPE "robot not attached"    END
    IF (TASK(2,TASK(0))==2)THEN
    TYPE "robot is attached"
END
```

Related Keywords

3-1-37 ERROR on page 3-48
3-4-100 PROCEED on page 3-410 (program command)
3-1-99 STATE on page 3-121
3-2-58 STATUS on page 3-244 (monitor command)
3-1-100 STATUS on page 3-127 (real-valued function)

3-1-106 $TIME

String function that returns a string value containing either the current system date and time or the specified date and time.

Syntax

```
$TIME (date, time)
```

Parameter

Parameter	Description
date	Optional integer value representing the year, month, and day. The value is interpreted as shown below where the 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. The value is interpreted as shown below where the hour ranges from 0 to 23. `time = hour*2048 + minute*32 + second/2` .

Additional Information

This function always returns a string containing both the date and the time. This 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 function.
The date and time are output in the format dd-mmm-yy hh:mm:ss in which the individual elements are defined as described below.

Element	Description
dd	The day of the month (1 to 31).

Element	Description
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 and 00 to 79 represent 2000 through 2079.
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)

The $TIME function converts passed arguments instead of the system time when either the date or the time parameter is supplied. However, the function attempts to generate a string representation of both parameters. It returns the date 01-Jan-80 if you do not provide a date value. It returns the time substring 00:00:00 if you do not specify a time value. The following statements can be used to return only the date and the time.

```
$date = $MID($TIME(date,),1,9)

$time = $MID($TIME(,time),11,8)
```

Examples

The following example displays time in the Monitor Window based on a bit-mask.

```
TYPE $TIME(10333,41472)
```

The following example displays the current system time in the Monitor Window.

```
$current_time=$TIME()
TYPE $current_time
```

Related Keywords

3-1-108 TIME on page 3-139
3-1-107 $TIME4 on page 3-137 (string function)

3-1-107 $TIME4

String function that returns a string value containing either the current system four-digit date and time or the specified four-digit date and time.

Syntax

```
$TIME4 (date, time)
```

Parameter

Parameter	Description
date	Optional integer value representing the year, month, and day.The value is interpreted as shown below where 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. The value is interpreted as shown below where hour ranges from 0 to 23. `time = hour*2048 + minute*32 + second/2`

Additional Information

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 function.

The date and time are output in the format dd-mmm-yyyy hh:mm:ss in which the individual elements are defined as described below. Using values that are not within the ranges below may cause unexpected values to be returned.

Element	Description
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 and 00 to 79 represent 2000 through 2079.
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)

Additional Information

:The $TIME function converts passed arguments instead of the system time when either the date or the time parameter is supplied. However, the function attempts to generate a string representation of both parameters. It returns the date 01-Jan-80 if you do not provide a date value. It returns the time substring 00:00:00 if you do not specify a time value. The following statements can be used to return only the date and the time.

```
$date = $MID($TIME4(date,),1,11)

$time = $MID($TIME4(,time),13,8)
```

Examples

The following example will display "31-Jan-2020 20:16:00" in the Monitor Window.

```
AUTO REAL date, year, month
AUTO REAL day, time, hour, minute, second


date = (year-1980)*512 + month*32 + day time = hour*2048 + minute*32 + second/2 TYP
E $TIME4(20543,41472)
```

Related Keywords

3-1-108 TIME on page 3-139 (real-valued function)
3-1-106 $TIME on page 3-136 (string function)

3-1-108 TIME

Real-valued function that returns an integer value representing either the date or the time specified in the given string parameter.

Syntax

TIME (string, **select)**

Parameter

Parameter	Description		
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
	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.

In a Standard Control system, system time can be set and displayed. However, In a Robot Integrated Control system, system time can only be displayed. If you attempt to set time, Error -423 Illegal operation occurs. You should also 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 value of the string must have one of the following formats

```
dd-mmm-yy hh:mm:ss    dd-mmm-yyyy hh:mm:ss dd-mmm-yy hh:mm    dd-mmm-yyyy 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:

Precautions for Correct Use

Using values that are not within the ranges below may cause unexpected values to be returned.

Element	Description
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 and 00 to 79 represent 2000 through 2079.
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)

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.

Examples

The following example demonstrates how the TIME real-valued function can be used to make sure you enter a valid date and time after a prompt:

```
PROMPT "Enter the date and time (dd-mmm-yy hh:mm:ss): ", $time
WHILE (TIME($time,1) == -1) DO
    TYPE /B, "    Cannot interpret date/time."
    PROMPT "Try again (dd-mmm-yy hh:mm:ss): ", $time
END
TIME $time
```

Related Keywords

3-1-106 $TIME on page 3-136
3-1-107 $TIME4 on page 3-137

3-1-109 TIMER

Real-valued function that returns the current time value of the specified system timer.

Syntax

```
TIMER (timer_number)
```

Usage Considerations

The accuracy and resolution of the timers vary according to which timer is selected. Double variables should be used to achieve maximum resolution.

Parameter

<table>
<tr><th>Parameter</th><th colspan="2">Description</th></tr>
<tr><td rowspan="8">timer_number</td><td colspan="2">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 -4 to 15.</td></tr>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>1-15</td><td>Timers with a resolution of one millisecond and a maximum count of > 2.E+009. They can be used to measure an interval of up to 596 hours from when they were set by the TIMER keyword.</td></tr>
<tr><td>0</td><td>Returns the number of seconds since the V+ system was started with a resolution of 1 millisecond and a maximum count of about 2.E +009. It is valid only during the first 596 hours of system operation and should generally not be used.</td></tr>
<tr><td>-1</td><td>Returns the low 24 bits of the time since the V+ system was started, in counts of 16 milliseconds. It can be used to compute time intervals of up to 74 hours.</td></tr>
<tr><td>-2</td><td>Returns the low 24 bits of the time since the V+ system was started, in counts of 1 millisecond. It can be used to compute time intervals of up to 4.6 hours.</td></tr>
<tr><td>-3</td><td>Returns the time in seconds since V+ was started as a 52-bit double-precision value. This timer has a resolution of 1 millisecond and a maximum count of > 4.E+015. It can be used to compute intervals of > 100,000 years. It is used like timers -1 and -2 except that double variables are required and no BAND operation or scale factors are used.
<code>AUTO DOUBLE start_time, interval start_time = TIMER(-3)
...
interval = TIMER(-3)-start_time</code></td></tr>
<tr><td>-4</td><td>Returns the double-precision time of the current robot-position or belt-encoder latch for this task. The timer resolution is 1 microsecond. This time will only be valid for 128 seconds.</td></tr>
</table>

Details

● Timers -1 and -2

If you do not want to use timers 1 through 15 or need more than 15 timers, Timers -1 and -2 may be used as follows.

```
AUTO DOUBLE start_time, interval, scale
scale = 62.5    )
start_time = TIMER(-1)
...
interval = ((TIMER(-1)-start_time) BAND ^HFFFFFF)/scale
```

The timer_number = -3 setting provides a preferred method for computing such intervals provided that a double value can be used.

The type of V+ variable used in time computations affects the maximum interval that can be computed with full resolution with the following considerations.

- Standard real variables have only 24 bits of resolution which limits the time interval to 16,777,216 (224) counts. This limit corresponds to about 4.6 hours for millisecond timers and 74 hours for 16-millisecond timers.
- Double real variables have 52 bits of precision, which stores the full resolution of the various timers. This is the default type used when none is explicitly specified.

Examples

The following example shows how the TIMER program command and the TIMER function can be used to time the execution of a subroutine.

```
TIMER 1 = 0
CALL test.routine()
TYPE "Elapsed time =", TIMER(1)," seconds"
```

Related Keywords

3-1-109 TIMER on page 3-141

3-1-110 TOOL

Transformation function that returns the value of the transformation specified in the last TOOL operation.

Syntax

TOOL

Usage Considerations

The TOOL function returns information for the robot selected by the task executing the function.
If the task executing this function does not have a robot selected, the output of this function is invalid.

Additional Information

The monitor command statement LISTL TOOL can be used to display the current tool setting.

Examples

The following example displays the value of the current TOOL transformation in the Monitor Window.

```
LISTL TOOL
```

The following example saves the value of the current TOOL.

```
SET save.tool = TOOL
```

Related Keywords

3-4-128 TOOL on page 3-451 (program command)

3-1-111 TPS

Real-valued function that returns the number of ticks of the system clock that occur per second (Ticks Per Second).

Syntax

TPS

Examples

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 keyword.

```
FOR ticks = 1 TO 5*TPS
    IF SIG(1001) THEN
        TYPE "Signal ON"
        HALT
    END
    WAIT
END
TYPE "Time-out while waiting for signal 1001"
```

3-1-112 TRANS

Transformation function that returns a transformation value computed from the given X, Y, Z position displacements and y, p, r orientation rotations.

Syntax

TRANS (X_value, Y_value, Z_value, y_value, p_value, r_value**)**

Usage Considerations

If any parameter is omitted, its value is taken to be zero.

Parameter

Parameter	Description
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

The following example returns points on a circle where "r" is the radius and "angle" is the angle of rotation about the circle.

```
TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0, 0)
```

The following example moves the robot tool point around the circle in steps of 1 degree. "frame" is a transformation defining the position of the center of the circle and the plane in which it lies, "r" is the radius of the circle and "angle" is the angle of rotation about the circle.

```
FOR angle = 0 TO 360-1
    MOVE frame:TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0, 0)
END
```

Related Keywords

3-4-31 DECOMPOSE on page 3-312
3-1-31 DX on page 3-42
3-1-32 DY on page 3-43
3-1-33 DZ on page 3-44
3-1-44 FRAME on page 3-63
3-1-81 #PPOINT on page 3-104
3-4-117 SET on page 3-433
3-1-90 SHIFT on page 3-112
3-1-114 TRANSB on page 3-147

3-1-113 $TRANSB

String function that returns a 48-byte string containing the binary representation of a transformation value.

Syntax

```
$TRANSB (transformation type, littleEndian)
```

Usage Considerations

Use float64 (type = 1) if full numerical accuracy is required, especially for long-distance robot movements or precise tooling.

Parameter

Parameter	Description
transformation	Transformation variable or function (or compound transformation) that defines the value to be converted to a string value.
type	Optional real expression which specifies the type of the position coordinates (0 = real, 1 = double).
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

The primary use of this function is to convert a transformation value to its binary representation in an output record of a data file.
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 x 4 transformation matrix, stored by column. Each of the twelve 32-bit values is assembled as four successive 8-bit characters in a string, resulting in a total of 48 or 96 characters.
The IEEE single-precision or double-precision standard floating-point format is used for the conversion. Refer to the *3-1-42 FLTB* on page 3-60 and *3-1-22 DBLB* on page 3-30 real-valued functions for details of the IEEE floating

Examples

This example demonstrates how to convert a V+ transformation variable in Euler ZYZ format (X, Y, Z, Yaw, Pitch, Roll) into its corresponding 3×4 rotation matrix representation using the $TRANSB keyword.
The matrix is encoded as 12 floating-point values (3 rows × 4 columns), stored as columns, with each element having either float32 (4 bytes) or float64 (8 bytes) precision depending on the type parameter. If double_precise = 1, each matrix element is stored as float64, producing 96 bytes in total (12 × 8 bytes). If double_precise = 0, each element is stored as float32, producing 48 bytes (12 × 4 bytes).

The 12 matrix values represent the internal transformation matrix in column-major order. When the location data is stored, this transformation matrix format gets written.
To reconstruct the original transformation, the TRANSB keyword decodes the 48/96-byte binary matrix back into a TRANS variable in Euler ZYZ format.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
SET trans_val = HERE

double_precise = 1
little.endian = 1

$s = $TRANSB(trans_val,double_precise,little.endian)

SET trans_val = NULL
SET trans_val = TRANSB($s,,double_precise,little.endian)

DECOMPOSE d[] = trans_val
TYPE ">>>Receive TRANS    (TRANSB)", d[0], d[1], d[2], d[3], d[4], d[5]
```

Related Keywords

3-1-41 $FLTB on page 3-59
3-1-42 FLTB on page 3-60
3-1-113 $TRANSB on page 3-146

3-1-114 TRANSB

Transformation function that returns a transformation value represented by a 48- byte or 96-byte string.

Syntax

```
TRANSB (string, first_char, type, littleEndian)
```

Usage Considerations

Use float64 (type = 1) if full numerical accuracy is required, especially for long-distance robot movements or precise tooling.

Parameter

Parameter	Description
string	String expression that contains the 48 or 96 bytes to be converted.

Parameter	Description
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 or 96 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 or 97th bytes are extracted. An error is generated if first_char specifies 48 or 96 bytes that are beyond the end of the input string.
type	Optional real expression which specifies the type of the position coordinates (0 = float, 1 = double).
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

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+.48 or 96 sequential bytes of the given string are interpreted as being a set of twelve single-precision (32-bit) or double-precision (64-bit) floating-point numbers in the IEEE standard format. Refer to the description of the *3-1-22 DBLB* on page 3-30 and *3-1-42 FLTB* on page 3-60 functions 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.

Examples

This example demonstrates how to reconstruct a transformation and decode a 48/96-byte binary matrix back into a TRANS variable in Euler ZYZ format.
Refer to the $TRANSB keyword example for more information.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
SET trans_val = HERE

double_precise = 1
little.endian = 1

$s = $TRANSB(trans_val,double_precise,little.endian)

SET trans_val = NULL
SET trans_val = TRANSB($s,,double_precise,little.endian)

DECOMPOSE d[] = trans_val
TYPE ">>>Receive TRANS    (TRANSB)", d[0], d[1], d[2], d[3], d[4], d[5]
```

Related Keywords

3-1-22 DBLB on page 3-30
3-1-42 FLTB on page 3-60

3-1-112 TRANS on page 3-144
3-1-114 TRANSB on page 3-147

3-1-115 TRUE

Real-valued function that returns the value used by V+ to represent a logical true result.

Syntax

```
... TRUE
```

Details

This named constant is useful for situations where true and false conditions need to be specified. The value returned is –1.

Examples

The following example executes continuously until the subroutine process returns a TRUE value for the real variable error.

```
DO
CALL process(error)
UNTIL error == TRUE
```

The following example will execute indefinitely.

```
WHILE TRUE DO
    CALL move.part()
END
```

Related Keywords

3-1-72 OFF on page 3-98
3-1-73 ON on page 3-98
3-1-38 FALSE on page 3-51

3-1-116 $TRUNCATE

String function that returns all characters in the input string until an ASCII NUL (or the end of the string) is encountered.

Syntax

```
$TRUNCATE (string)
```

Parameter

Parameter	Description
string	String variable, constant, or expression that specifies the string to be truncated.

Details

This function is similar to performing a $DECODE function with an ASCII NUL (^H00) specified as the break character. $TRUNCATE differs from the $DECODE function in the following ways.

- The input can be a string expression.
- The input string is not modified.

Usage Considerations

Because of its simplicity, the $TRUNCATE function executes much faster than the $DECODE function.

Examples

The example below sets the value of the string variable "$substring" equal to "abcdef".This is an artificial situation, since you would not 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 Keywords

3-1-24 $DECODE on page 3-33

● Parameter

Parameter	Description
string	String variable, constant, or expression that specifies the string to be truncated.

3-1-117 $UINTB

String function that returns a two-byte string containing the binary representation of a 16-bit unsigned integer.

Syntax

```
$UINTB (ival,littleEndian)
```

Parameter

Parameter	Description
ival	Real-valued expression, the value of which is converted to its binary representation.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

The unsigned 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.
The primary use of this function is to convert integer values to binary representation within an output record of a data file.

Example

This example demonstrates how to encode a V+ numeric variable (uint16_value) as an unsigned 16-bit integer using the $UINTB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
uint16_value = 65535
TYPE "Original uint16_value:  ", uint16_value

little.endian = 1
$s = $UINTB(uint16_value,little.endian)

TYPE "uint16_value encoded (Little Endian): ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2))
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-21 $DBLB on page 3-29
3-1-41 $FLTB on page 3-59
3-1-55 $INTB on page 3-79
3-1-63 $LNGB on page 3-89
3-1-118 UINTB on page 3-151

3-1-118 UINTB

Real-valued function that returns the value of a two-byte string interpreted as a 16-bit unsigned integer.

Syntax

```
UINTB ($string, first_char, littleEndian)
```

Parameter

Parameter	Description
$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 establishes 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.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

Two sequential bytes of a string are interpreted as being a 2's-complement 16-bit unsigned binary integer. The first byte contains bits 9 to 16, and the second byte contains bits 1 to 8.
The primary use of this function is to convert binary numbers from an input data record to values that can be used internally by V+.

Example

The following example encodes a V+ numeric variable (uint16_value) as an unsigned 16-bit integer using the $UINTB keyword and then decodes the 2-byte binary string ($s) back to a numeric variable (decoded_uint16) using the UINTB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
uint16_value = 65535
TYPE "Original uint16_value:  ", uint16_value

little.endian = 1
$s = $UINTB(uint16_value,little.endian)

TYPE "uint16_value encoded (Little Endian): ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2))
TYPE

TYPE "Simulate receiving 2 bytes:  ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2))

decoded_uint16 = UINTB($s,,little.endian)
```

```
TYPE "Decoded 'decoded_uint16' from bytes: ", decoded_uint16
```

Related Keywords

3-1-117 $UINTB on page 3-150
3-1-16 $CHR on page 3-22
3-1-21 $DBLB on page 3-29
3-1-41 $FLTB on page 3-59
3-1-55 $INTB on page 3-79
3-1-56 INTB on page 3-80
3-1-63 $LNGB on page 3-89

3-1-119 $ULNGB

String function that returns a 4-byte string containing the binary representation of a 32-bit unsigned integer.

Syntax

$ULNGB (uliVal,littleEndian**)**

Parameter

Parameter	Description
uliVal	Real-valued expression, the value of which is converted to its binary representation.
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

The unsigned integer part of a real value is converted into its binary representation. The low 32-bits of that binary representation are assembled into a string as four 8-bit characters. Bits 25 to 32 are assembled into the first byte, followed by bits 17 to 24 in the second byte, and so forth.
The primary use of this function is to convert unsigned integer values to binary representation within an output record of a data file.

Example

This example demonstrates how to encode a V+ numeric variable (uint32_value) as an unsigned 32-bit integer using the $ULNGB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
uint32_value = 4.294967295E+09
TYPE "Original uint32_value:  ", uint32_value
```

```
little.endian = 1
$s = $ULNGB(uint32_value,little.endian)

TYPE "uint32_value encoded (Little Endian): ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-41 $FLTB on page 3-59
3-1-55 $INTB on page 3-79
3-1-63 $LNGB on page 3-89
3-1-117 $UINTB on page 3-150
3-1-64 LNGB on page 3-91
3-1-114 TRANSB on page 3-147
3-1-120 ULNGB on page 3-154

3-1-120 ULNGB

Real-valued function that returns the value of four bytes of a string interpreted as an unsigned 32-bit binary integer.

Syntax

ULNGB ($string, first_char, littleEndian**)**

Parameter

Parameter	Description
$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 unsigned 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).
littleEndian	Optional value to specify the byte formatting. 1 specifies little endian. 0 specifies big endian.

Details

Four sequential characters (bytes) of a string are interpreted as being a 2's-complement 32-bit unsigned 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, and so on.
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 on.

Example

The following example encodes a V+ numeric variable (uint32_value) as an unsigned 32-bit integer using the $ULNGB keyword and then decodes the 4-byte binary string ($s) back to a numeric variable (decoded_uint32) using the ULNGB keyword.
The TYPE keyword is used to display the results of the example in the Monitor Window.

```
uint32_value = 4.294967295E+09
TYPE "Original uint32_value:  ", uint32_value

little.endian = 1
$s = $ULNGB(uint32_value,little.endian)

TYPE "uint32_value encoded (Little Endian): ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))

TYPE "Simulate receiving 4 bytes:  ", /S
TYPE $ENCODE(/H2,ASC($s,1),"-",ASC($s,2),"-",ASC($s,3),"-",ASC($s,4))

decoded_uint32 = ULNGB($s,,little.endian)

TYPE "Decoded 'decoded_uint32' from bytes: ", decoded_uint32
```

Related Keywords

3-1-16 $CHR on page 3-22
3-1-41 $FLTB on page 3-59
3-1-55 $INTB on page 3-79
3-1-63 $LNGB on page 3-89
3-1-117 $UINTB on page 3-150
3-1-119 $ULNGB on page 3-153
3-1-64 LNGB on page 3-91
3-1-114 TRANSB on page 3-147
3-1-118 UINTB on page 3-151

3-1-121 $UNPACK

String function that returns a substring from an array of 128-character string variables.

Syntax

```
$UNPACK (string_array[index], first_char, num_chars)
```

Parameter

Parameter	Description
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 0 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 0.
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 sub- string whose first character is the 127th character in element [3] is 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.
For example, element [2,3] is followed by element [2,4].

Examples

The following example 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 "$list[3]". However, because the first character is to be number 130, the 11-character substring actually consists of the second through 12th characters of "$list[4]".

```
$substring = $UNPACK($list[3], 130, 11)
```

Related Keywords

3-1-66 $MID on page 3-93
3-4-90 PACK on page 3-398

3-1-122 VAL

Real-valued function that returns the real value represented by the characters in the input string.

Syntax

```
VAL (string)
```

Usage Considerations

The input string can be a number in scientific notation.
The input string can contain leading number base indicators (^H).
The input string can contain a + or -sign before the numeric part of the string, but after any optional base indicator.
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

Parameter	Description
string	String constant, variable, or expression.

Examples

The following example returns the real value 123.

```
VAL("123 Elm Street")
```

The following example returns the real value 0.012.

```
VAL("1.2E-2")
```

The following example returns the real value 255 .

```
VAL("^HFF")
```

Related Keywords

3-1-5 ASC on page 3-11
3-1-35 $ENCODE on page 3-45
3-1-42 FLTB on page 3-60
3-1-56 INTB on page 3-80
3-1-64 LNGB on page 3-91

3-1-123 VLOCATION

Transformation function that returns a cartesian transformation result of the execution of the specified vision sequence. The returned value is a transform result as x, y, z, yaw, pitch, and roll.

Syntax

```
VLOCATION($ip, sequence_id, tool_id, instance_id, result_id, index_id, frame_id)
```

Usage Considerations

Refer to *Robot Vision Manager User's Manual (Cat. No. I667)* and *V+ Module Reference Manual (Cat. No. I668)* for additional information.

This keyword involves communication with ACE. Using this keyword as an argument for another function can cause unnecessary delays or consume unnecessary processing power. Avoid using this keyword more often than necessary.

Parameter

Parameter	Description
$ip	IP address of the vision server. The vision server is the PC on which the Robot Vision Manager is running and uses a standard IP address format (192.168.1.120 for example).
sequence_id	Index of the vision sequence. The first sequence is 1.
tool_id	Optional index of the tool in the vision sequence. The first tool is 1. instance_id Optional index of the instance in the specified result frame. If no result frame is specified, it is the index for all instances returned by the tool.
result_id	Optional identifier (ID) of the result. Typically this value is 1311. For gripper offset location, this value can be set to 1400 and incremented by 1 for each additional gripper offset. The maximum value is 1499.
index_id	Reserved for internal use (value is always 1).
frame_id	Optional index of the frame for which you want to retrieve the result contained in the specified instance.

Details

If no value is provided for optional parameters, they default to 1. To retrieve global values, use the following parameter settings.

- sequence_id:-1
- tool_id:-1

To retrieve camera values, use the following parameter settings.

- sequence_id:-1
- tool_id:camera Index

To retrieve camera-relative-to robot values, use the following parameter settings.

- sequence_id:-1
- tool_id:camera Index
- index_id:robot Index

To retrieve sequence values, use the following parameter settings.

- sequence_id:sequence Index
- tool_id:-1

To retrieve belt calibration-related values, use the information in the table below.

Property	sequence_id	tool_id	instance_id	result_id	index_id	frame_id
Frame	-1	camera index	n/a	10000	robot index	n/a
Upstream Limit				10001		
Down- stream Limit				10002		
Nearside Lim-it				10003		
Vision Origin				10050		

To retrieve belt latch calibration offsets, use the values provided below.

- Property: Latch Calibration Offset
- sequence_id:-1
- tool_id: Reference number as defined in keyword mapping parameter of Robot Vision Manager Latch Calibration in Sysmac Studio or ACE software.
- instance_id:n/a
- result_id:10010
- index_id:robot Index
- frame_id:n/a

Examples

The following example retrieves the location of a found instance where the 1311 result_id indicates using the first gripper offset. This is equivalent to using the 1400 result_id.

```
SET location = VLOCATION($ip, 1, 2, 1, 1311)
```

The following example sets 1 to 6 gripper offset locations where the first gripper offset location is 1400.

```
SET location = VLOCATION ($ip, 1, 2, 1, 1400)
SET location = VLOCATION ($ip, 1, 2, 1, 1401)
SET location = VLOCATION ($ip, 1, 2, 1, 1402)
SET location = VLOCATION ($ip, 1, 2, 1, 1403)
SET location = VLOCATION ($ip, 1, 2, 1, 1404)
SET location = VLOCATION ($ip, 1, 2, 1, 1405)
```

The following example retrieves the location of the belt frame where the Belt Calibration Frame index is 1000.

```
VLOCATION ($ip, -1, cameraIndex, , 10000, robotIndex)
```

The following example retrieves the location of the vision origin where the Vision Origin index is 10050.

```
VLOCATION ($ip, -1, cameraIndex, , 10050, robotIndex)
```

Related Keywords

3-1-124 VPARAMETER on page 3-160
3-1-125 VRESULT on page 3-161
3-1-126 VSTATE on page 3-163

3-1-124 VPARAMETER

Transformation function that returns the current value of a vision tool parameter.

Syntax

```
VPARAMETER ($ip, sequence_id, tool_id, parameter_id, index_id, object_id)
```

Usage Considerations

Refer to *Robot Vision Manager User's Manual (Cat. No. I667)* and *V+ Module Reference Manual (Cat. No. I668)* for additional information.
This keyword involves communication with ACE. Using this keyword as an argument for another function can cause unnecessary delays or consume unnecessary processing power. Avoid using this keyword more often than necessary.

Parameter

Parameter	Description
$ip	IP address of the vision server. The vision server is the PC on which the Robot Vision Manager is running and uses a standard IP address format (192.168.1.120 for example).
sequence_id	Index of the vision sequence. The first sequence is 1.
tool_id	Optional index of the tool in the vision sequence. The first tool is 1. instance_id Optional index of the instance in the specified result frame. If no result frame is specified, it is the index for all instances returned by the tool.
index_id	Reserved for internal use. Value is always 1.
object_id	Some parameters require an object index to access specific values in an array.

Details

If no value is provided for optional parameters, they default to 1. To retrieve global values, use the following parameter settings.

- sequence_id:-1
- tool_id:-1

To retrieve camera values, use the following parameter settings.

- sequence_id:-1
- tool_id:camera Index

To retrieve sequence values, use the following parameter settings.

- sequence_id:sequence Index
- tool_id:-1

To retrieve belt calibration-related values for Scale (10004), use the following parameter set- tings.

- sequence_id= -1

- tool_id= camera Index
- index_id= robot Index
- object_id= n/a

To retrieve sequence-related values for Mode (10200), use the following parameter settings.

- sequence_id= sequence Index
- tool_id= -1
- index_id= n/a
- object_id= n/a

Examples

The following example will retrieve the scale value for the belt calibration.

```
scalevalue = VPARAMETER ($ip, -1, cameraIndex, 10004, robotIndex)
```

Related Keywords

3-1-123 VLOCATION on page 3-157
3-1-125 VRESULT on page 3-161
3-1-126 VSTATE on page 3-163

3-1-125 VRESULT

Real-valued function that returns a specified result of a vision tool, or returns the status of a specified tool.

Syntax

```
VRESULT ($ip, sequence_id, tool_id, instance_id, result_id, index_id, frame_id)
```

Usage Considerations

Refer to *Robot Vision Manager User's Manual (Cat. No. I667)* and *V+ Module Reference Manual (Cat. No. I668)* for additional information.

This keyword involves communication with ACE. Using this keyword as an argument for another function can cause unnecessary delays or consume unnecessary processing power. Avoid using this keyword more often than necessary.

Parameter

Parameter	Description
$ip	IP address of the vision server. The vision server is the PC on which the Robot Vision Manager is running and uses a standard IP address format (192.168.1.120 for example).
sequence_id	Index of the vision sequence. The first sequence is 1.

Parameter	Description
tool_id	Optional index of the tool in the vision sequence. The first tool is 1.
instance_id	Optional index of the instance in the specified result frame. If no result frame is specified, it is the index for all instances returned by the tool.
result_id	Optional identifier (ID) of the result.
index_id	Reserved for internal use. Value is always 1.
frame_id	Optional index of the frame for which you want to retrieve the result contained in the specified instance.

Details

If no value is provided for optional parameters, they default to 1.
When a VRESULT function is issued for a specific tool, it checks to see if that tool supports the VRESULT code. If the specified tool does not support the code, the VRESULT function moves to the parent tool to see if it supports the code. This process continues until it finds a tool that supports the code. If no valid tool is found, an invalid vision result error is generated.
For example, suppose an Arc Finder tool is placed relative to a Blob Analyzer tool. In the application, the Blob Analyzer tool locates many blobs objects and adds an Arc Finder tool at each instance. If you make a request for the blob area associated with an arc finder instance,the VRESULT function will recognize that the Arc Finder tool does not support that code, so it moves to the parent tool (the Blob Analyzer tool) and finds the blob instance associated with the specified arc result. It validates that the blob result supports the VRESULT code and returns the data.
Some vision tools are considered Frame Sources. The Blob Analyzer and Locator tool are the most commonly used Frame Sources. When these tools execute, it will mark each result as a separate frame or grouping. Any vision tools relative to a Frame Source will associate each of its results with the frame it is relative to. In this case, you may want to use the frame_id parameter to extract the results.
Using the Arc Finder tool relative to the Blob Analyzer tool described above for example, if the Blob Analyzer locates 5 different results, then the Arc Finder tool will execute 5 different Arc Finder operations, one relative to each result returned by the Blob Analyzer. The Arc Finder will associate each result with a frame number that correlates with the index of the result returned by the Blob Analyzer. If you want to request an Arc Finder result associated with the 4th result of the Blob Analyzer, you would reference index_id =1 in frame_id = 4 (requesting the first instance in result frame 4). In this situation, you can still access all the Arc Finder results using frame_id = -1.

Additional Information

Some child vision tools may have multiple results within each frame and might have no results within a frame.

Examples

The following example retrieves the number of instances found by a Locator tool where instance count = 1310.

```
instance_count = VRESULT($ip, 1, 2, 1, 1310)
```

Related Keywords

3-1-123 VLOCATION on page 3-157
3-1-124 VPARAMETER on page 3-160
3-1-126 VSTATE on page 3-163

3-1-126 VSTATE

Real-valued function that returns the state of the execution of a sequence.

Syntax

```
VSTATE ($ip, sequence_id)
```

Usage Considerations

Refer to *Robot Vision Manager User's Manual (Cat. No. I667)* and *V+ Module Reference Manual (Cat. No. I668)* for additional information.
This keyword involves communication with ACE. Using this keyword as an argument for another function can cause unnecessary delays or consume unnecessary processing power. Avoid using this keyword more often than necessary.

Parameter

Parameter	Description
$ip	IP address of the vision server. The vision server is the PC on which the Robot Vision Manager software is running and uses a standard IP address format (192.168.1.120 for example).
sequence_id	Index of the vision sequence. The first sequence is 1.

Details

Details for the values returned are provided below.

Value	Description
0	Idle
1	Running
2	Paused
3	Done
4	Error
5	Starting

Examples

The following example will wait until the vision sequence has completed.

```
DO
    WAIT
UNTIL VSTATE($ip, 1) == 3
```

Related Keywords

3-1-123 VLOCATION on page 3-157
3-1-124 VPARAMETER on page 3-160
3-1-125 VRESULT on page 3-161

3-1-127 WINDOW

Real-valued function that returns a value to indicate 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.

Syntax

WINDOW (transformation, time, mode**)**

Usage Considerations

The belt variable referenced in the compound transformation must have already been defined using a DEFBELT keyword.

Parameter

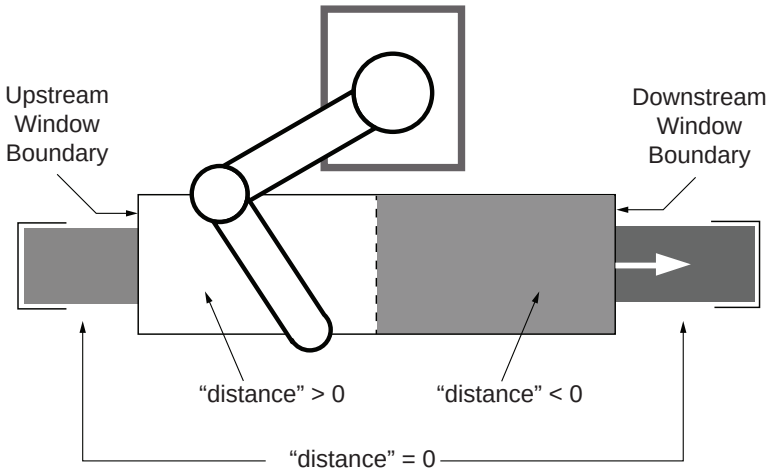
Parameter	Description
transformation	Compound transformation value that is defined relative to a belt. The compound transformation must begin with a belt variable.
time	Optional real-valued expression that specifies the time to anticipate when the transformation is evaluated. The result of the function is the value predicted to apply time seconds in the future, based on the current belt position and speed. This parameter is used to evaluate if a motion can be correctly completed within an anticipated time period. A time of zero (default) evaluates 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.

Details

The value returned is a distance in millimeters and should be interpreted as described below.

Value	Interpretation
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.

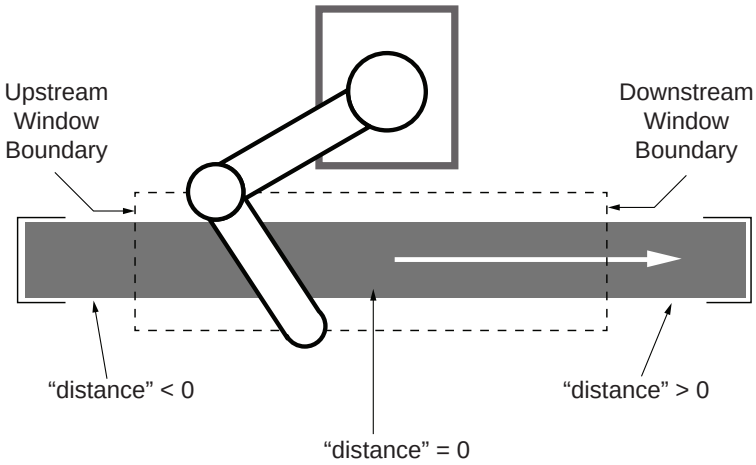
The definitions of upstream and downstream depend on the value of the BELT.MODE system parameter.



If the value of the mode expression is greater than zero, the value returned is interpreted as described below.

Value	Interpretation
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.

If the value of the mode expression is less than or equal to zero (default), the value returned is interpreted as described below.



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.

Examples

The following example sets the variable "distance" to a nonzero value if the location will be outside the operating window for "%belt1" in 2 seconds. The variable "distance" will be 0 if the location is within the window.

```
distance = WINDOW(%belt1:pick.up, 2, 1)
```

The distance is nonzero if, in two seconds, the location will be outside the operating window for %belt1. Otherwise, distance is zero if the location is within the window.

Related Keywords

3-1-11 BELT on page 3-16 (real-valued function)
3-5-1 BELT.MODE on page 3-468
3-1-14 BSTATUS on page 3-20
3-4-33 DEFBELT on page 3-314
3-4-118 SETBELT on page 3-434

3-2 Monitor Command Keywords

Use the information in this section to understand monitor command keywords and their use with the V + system.

3-2-1 ABORT

Monitor command that terminates execution of an executable program.

Syntax

```
ABORT task
```

Usage Considerations

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

Parameters

Parameter	Description
task	Optional real value, variable, or expression interpreted as an integer that specifies which program task is to be terminated.

Details

Terminates execution of the specified active 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.
If the task number is not specified, the ABORT command accesses task number 0.
If the task being aborted was initiated with a monitor command keyword, a completion message in the following form is displayed:

```
Program task # stopped at program_name, step step_number date time
```

However, if the task was initiated from another task with an EXECUTE program command keyword, the completion message is not displayed.

Related Keywords

3-4-1 ABORT on page 3-268 (program command)
3-2-9 CYCLE.END on page 3-177 (monitor command)
3-4-30 CYCLE.END on page 3-310 (program command)
3-2-21 ESTOP on page 3-194 (monitor command)

3-4-45 ESTOP on page 3-330 (program command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)
3-2-44 PANIC on page 3-228 (monitor command)
3-4-91 PANIC on page 3-400 (program command)
3-4-100 PROCEED on page 3-410
3-4-110 RETRY on page 3-426
3-2-58 STATUS on page 3-244

3-2-2 BASE

Monitor command that translates and rotates the world reference frame relative to the robot.

Syntax

```
BASE X_shift, Y_shift, Y_rotation1, Z_shift, Z_rotation1, Z_rotation2
```

Usage Considerations

The BASE command applies to the robot selected with the SELECT keyword. This command can be used while programs are executing. An error will result if the robot is attached by any executing program.
If the V+ system is not configured to control a robot, use of the BASE command causes an error.

Parameters

Parameter	Description
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. A zero value is assumed if no value is provided.
Y_shift	Optional real-valued expression describing the Y component in the normal world coordinate system of the origin point for the new coordinate system. A zero value is assumed if no value is provided.
Y_rotation1	Optional real-valued expression describing the rotation about the Y-axis component in the normal world coordinate system of the origin point for the new coordinate system. A zero value is assumed if no value is provided.
Z_shift	Optional real-valued expression describing the Z component in the normal world coordinate system of the origin point for the new coordinate system. A zero value is assumed if no value is provided.

Parameter	Description
Z_rotation1	Optional real-valued expression describing the rotation about the Z-axis component in the normal world coordinate system of the origin point for the new coordinate system. A zero value is assumed if no value is provided.
Z_rotation2	Optional real-valued expression describing a second rotation about the Z-axis component in the normal world coordinate system of the origin point for the new coordinate system. A zero value is assumed if no value is provided.

Details

When the V+ system is initialized, the origin of the reference frame of the robot is defined in the kinematic model. For SCARA robots, 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. Refer to the Robot User's Guide for the default location of the reference frame for your robot.
The BASE command offsets and rotates the reference frame as specified above. This is useful if the robot is moved after the locations have been defined for an application. This command can be used to compensate for the location differences that occur when a robot is moved to locations that have been defined by transformations relative to a robot reference frame (to a point translated by dX, dY, dZ, and rotated by Z_degrees). This ensures motions to the previously defined locations will still work properly. Additionally, the BASE command can be used to realign the X and Y coordinate axes so that SHIFT functions cause displacements in desired, nonstandard directions.
The BASE command has no effect on locations defined as precision points. The parameters for the BASE command 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 (with the statement "LISTL BASE").

Examples

The following example will redefine the world reference frame because the robot has been shifted "xbase" millimeters in the positive X direction, 50.5 millimeters in the negative Z direction, and rotated 30 degrees about the Z axis.

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

The following example will redefine 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. The arguments for this statement describe movement of the robot reference frame relative to the robot and have an opposite effect on locations relative to the robot.

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

Related Keywords

3-1-8 BASE on page 3-14 (transformation function)
3-4-13 BASE on page 3-286 (program command)
3-2-53 SELECT on page 3-237 (monitor command)

3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-3 BASE.TRANS

Monitor command that translates and rotates the reference frame relative to the robot base.

Syntax

```
BASE.TRANS transformation
```

Usage Considerations

The BASE.TRANS monitor command applies to the robot selected with the SELECT keyword. This command can be used while programs are executing. An error will result if the robot is attached by any executing program.
If the V+ system is not configured to control a robot, use of the BASE.TRANS monitor command causes an error.

Parameter

Parameter	Description
transformation	Optional transformation variable, function, or compound transformation that defines the robot base location. Zero is assumed if no value is provided.

Examples

The following example redefines the world reference frame using the given coordinates and three rotations. The first rotation executed is 20 degrees around the Z axis, the second rotation executed is 180 degrees about the Y axis, and the third executed is -70 degrees around the Z axis. The TRANS function is used to return a value computed from the given X, Y, Z position displacements and y, p, r orientation rotations. This value is used as the BASE.TRANS transformation parameter.

```
BASE.TRANS TRANS(250,-300,175,20,180,-70)
```

The following example redefines the world reference frame because the robot has been shifted xbase millimeters in the positive X direction, 50.5 millimeters in the negative Z direction, and has been rotated 30 degrees about the Z-axis. The TRANS function is used to return a value computed from the given X, Y, Z position displacements and y, p, r orientation rotations. This value is used as the BASE.TRANS transformation parameter.

```
BASE.TRANS TRANS(xbase,, -50.5, 30)
```

The following example 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 base, and thus have an opposite effect on locations relative to the robot. The TRANS function is used to return a value computed from the given X, Y, Z position displacements and y, p, r orientation rotations. This value is used as the BASE.TRANS transformation parameter.

```
BASE.TRANS TRANS(100,, -50)
```

Related Keywords

3-2-2 BASE on page 3-168 (monitor command)
3-1-8 BASE on page 3-14 (transformation function)
3-4-13 BASE on page 3-286 (program command)
3-4-14 BASE.TRANS on page 3-287 (program command)
3-2-53 SELECT on page 3-237 (monitor command)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-4 BITS

Monitor command that sets or clears a group of digital signals based on a value.

Syntax

BITS first_sig, num_sigs **= value**

Usage Considerations

External, digital output, or internal software signals can be referenced. The specified signals must not include any that are configured for input.
No more than 32 signals can be set at one time.
Any group of up to 32 signals can be set, providing that all the signals in the group are configured for use by the system.

Parameters

Parameter	Description
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 32.
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 signal, or internal software signal based on the value on the right of the equal sign. The effect of this operation 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

The following example sets external output signals 1-4 (4 bits) to the binary representation of the BCD digit "7".

```
BITS 1,4 = BCD(7)
```

The following example sets external output signals 9-16 (8 bits) to the binary representation of the current monitor speed setting. If the monitor speed were currently set to 50% (110010 binary), then signals 9-16 would be set as shown after the command:

```
BITS 9,8 = SPEED(1)
9 → 0 (off)     13 → 1 (ON)
10 → 1 (on)     14 → 1 (ON)
11 → 0 (off)     15 → 0 (OFF)
12 → 0 (off)     16 → 0 (OFF)
```

The following example sets external output signals 1-8 (8 bits) to the binary representation of the constant 255, which is 11111111. Signals 1-8 will all be turned ON.

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

Related Keywords

3-4-16 BITS on page 3-290 (program command)
3-1-12 BITS on page 3-17 (real-valued function)
3-2-50 RESET on page 3-235
3-1-93 SIG on page 3-115 (real-valued function)
3-1-91 SIG.INS on page 3-113 (real-valued function)
3-2-54 SIGNAL on page 3-238 (monitor command)
3-4-120 SIGNAL on page 3-437 (program command)

3-2-5 CALIBRATE

Monitor command that initializes the robot positioning system.

Syntax

CALIBRATE mode, robot

Usage Considerations

This key word is typically issued with no mode specified.
The CALIBRATE operation has no effect if the DRY.RUN system switch is enabled.
If the robot is to be moved, the CALIBRATE program command or monitor command keywords must be processed every time system power is turned ON and the V+ system is booted from disk. If a robot has the option to execute CALIBRATE at boot enabled, then it will perform the CALIBRATION operation upon boot up. If this option is not enabled (or not supported for that robot), then this keyword must be executed after high power is enabled.

Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information about enabling the option to execute CALIBRATE at boot.
Robots cannot be moved with the manual control pendant or under program control if the robot is not calibrated until the CALIBRATE program command or monitor command keyword has been processed.
If multiple robots are connected to the system controller and the "robot" parameter is omitted (or 0), this keyword attempts to calibrate all the robots in sequence unless they are disabled with the ROBOT system switch. All of the enabled robots must be calibrated before any of them can be moved under pro- gram control.
If the optional front panel or a remote front panel is installed, the controller key switch must be set to AUTOMATIC mode for this command to be processed.
The CALIBRATE keyword may operate differently for each type of robot. The CALIBRATE keyword generally causes all the robot joints to move. The positions from which the CALIBRATE keyword can be issued depend on the type of robot being controlled. The only restriction is that 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.

Parameters

Parameter	Description	
mode	A real-valued expression that indicates what part of calibration is to be performed:	
	Value of Mode	Interpretation
	0 (or omitted)	Perform a normal calibration of all the robots controlled by the system. The following operations are performed. • Load the main calibration program if it is not already in memory. • Execute the main calibration program with the load, execute, delete, and monitor flags set, which causes the robot- specific routines to be loaded, the robots to be calibrated, and the robot routines to be deleted. • Delete the main calibration program if it was loaded. Regardless of whether or not the main calibration program is deleted at the end of the process, the robot-specific routines will have been deleted by the main program.
	1	• Load the main calibration program if it is not already in memory, and execute the main calibration program with the load and monitor flags set. This causes the main program to load the applicable robot-specific routines. Note that the actual calibration process is not performed.
	2	• Execute the main calibration program which must already be in memory with the execute and monitor flags set. This causes the system robot(s) to be calibrated, and all the calibration programs to be left in memory.
	3	• Execute the main calibration program (which must already be in memory) with the delete and monitor flags set. This causes all the robot-specific calibration routines to be deleted from memory. Then the main calibration program is deleted from memory. Note that the actual calibration process is not performed.

Details

When the calibration operation is started, the V+ system proceeds as if the robot is not calibrated and does not let you execute a robot-control program. The pendant COMP mode light does not come on when the robot is not calibrated.
The robot becomes uncalibrated whenever system power is switched OFF. As a safety measure, robots also become uncalibrated whenever certain servo errors occur.
In the cases that involve loading the main calibration program, the CALIBRATE operation loads the disk file "cal_util.v2". For convenience, the loading operation searches for the file in the following directories and in the following order:

1. The current default directory
2. \CALIB\ on the local disk from which the V+ system was booted (skipped when the system was booted from the TFTP device)
3. DISK>D:\CALIB\

The calibration program is executed in task 0. If task 0 is already active, the CALIBRATE operation fails.

Related Keywords

3-4-20 CALIBRATE on page 3-294 (program command)
3-5-4 NOT.CALIBRATED on page 3-471

3-2-6 CD

Monitor command that displays or changes the default path for disk access.

Syntax

CD `path`

Parameters

Parameter	Description
path	Optional string specifying the disk-directory path of interest. Normally, this parameter contains directory names and backslash (\) characters. The V+ system adds a backslash if one is not included at the end of a path specification. If the path parameter is omitted, the current directory path is displayed.

Details

This command is a synonym for typing the following.

```
default disk = path
```

Refer to the *3-2-10 DEFUALT* on page 3-179 command for information about specifying the path for a disk directory.

Examples

Type the following to display the default path.

```
cd
```

Type the following to change the default path to C:\TEST\JOBS\. The path must already exist to change the default path as specified in this example. The trailing backslash can be omitted.

```
cd c:\test\jobs\
```

Type the following to move up the directory path one level.

```
cd ..
```

Type the following to move back down one level for this example (return to c:\TEST\JOBS from c:\TEST).

```
cd jobs
```

Related Keywords

3-2-10 DEFUALT on page 3-179

3-2-7 COMMANDS

Monitor command that initiates processing of a Monitor Command program.

Syntax

COMMANDS program

Usage Considerations

The COMMANDS command can be issued when program task 0 is executing, but the system keyboard will not respond to input until either the program completes or CTRL+C is pressed to abort the COMMANDS command.

Every command line in a Monitor Command program must begin with "MC".

The Front Panel key switch must be set to AUTOMATIC mode for this command to be processed.

Parameters

Parameter	Description
program	Name of the Monitor Command program to be processed.

Details

COMMANDS initiates processing of the specified Monitor Command program, which must already be in memory. Processing of the program will continue until one of the following occurs.

- The end of the Monitor Command program is reached.
- ACTRL+C sequence is pressed on the system keyboard.
- ACOMMANDS command is encountered in the program.
- An error occurs.

If a COMMANDS command is included in a Monitor Command program, the new Monitor Command program will be invoked and any remaining lines in the first Monitor Command program will be ignored. Monitor Command programs can be linked from one to another, but no return path can be made to occur as with executable programs. Any executable program invoked by a Monitor Command program (with an EXECUTE command) can utilize all of the control keywords available in the V+ language.
The autostart feature provides a method to automatically issue a COMMANDS command when the controller is powered ON and the V+ system is loaded from disk. Refer to *V+ User's Manual (Cat. No. I671)* for more information.

Examples

The following example begins processing of the Monitor Command program named "setup".

```
commands setup
```

Related Keywords

3-2-9 CYCLE.END on page 3-177
3-2-19 DO on page 3-191

3-2-8 COPY

Monitor command that creates a new program as a copy of an existing program.

Syntax

```
COPY new_program = old_program
```

Usage Considerations

The COPY command can be used to copy a program that is executing. COPY does not copy disk files, only programs resident in system memory (refer to the *3-2-23 FCOPY* on page 3-197 command to copy disk files).

Parameters

Parameter	Description
new_program	Name to be given to the program created

Parameter	Description
old_program	Name of the program to be copied.

Details

Creates a new program as an exact copy of an existing program. The new copy of the program is placed in the GLOBAL program module. After the COPY operation, either the new or the old program can be edited as desired.
If there is already a program in the system memory with the specified new name, the COPY operation is not performed and an error message is displayed. In this case, you must first delete or rename the conflicting program before copying, or use a different name for the copy.

Examples

The following example makes a copy of program "test" and assigns the name "test.cpy" to the new copy.

```
COPY test.cpy = test
```

Related Keywords

3-2-23 FCOPY on page 3-197 (monitor command)
3-4-51 FCOPY on page 3-341 (program command)
3-2-49 RENAME on page 3-234

3-2-9 CYCLE.END

Monitor command that terminates the specified executable program the next time it executes a STOP operation or its equivalent. It will suspend processing of a command program until a program completes execution.

Syntax

```
CYCLE.END task, stop_flag
```

Usage Considerations

The CYCLE.END command has no effect if the specified program task is not active.
The CYCLE.END command blocks all keyboard input until the specified task completes execution. Pressing CTRL+C releases the keyboard. In that case the CYCLE.END command will still terminate the program if the "stop_flag" is TRUE.

Parameters

Parameter	Description
task	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 command accesses task number 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. If the parameter has a nonzero value, the selected task will stop 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 operation or its equivalent, regardless of how many program cycles are left to be executed.
CYCLE.END does not terminate a program with continuous internal loops. Such a program must be terminated with the ABORT operation
Regardless of the "stop_flag" parameter, this command waits until the program actually is terminated. If the program being terminated loops internally, so that the current execution cycle never ends, the CYCLE.END command waits forever.
To release the system keyboard from a CYCLE.END command that is waiting for a program to terminate, press CTRL+C.

Examples

The following portion of a command program shows how to make a command program wait for execution of one program to complete before issuing the next command.

```
MC EXECUTE 1 setup    )
MC CYCLE.END 1
MC EXECUTE 1 main.1
MC EXECUTE main
MC CYCLE.END
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-4-66 HALT on page 3-365
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)

3-4-100 PROCEED on page 3-410
3-2-52 RETRY on page 3-236
3-2-58 STATUS on page 3-244 (monitor command)
3-1-100 STATUS on page 3-127 (real-valued function)
3-4-125 STOP on page 3-448

3-2-10 DEFUALT

Monitor command that defines the default relationship between the V+ disk logical device and the physical device to be accessed. This also displays the current default.

Syntax

```
DEFAULT DISK = physical_device>unit:directory_path
```

Usage Considerations

The simpler CD command can be used instead of the DEFAULT command.

Parameters

Parameter*1	Description
physical_device	Optional name of the physical device to be associated with the disk logical device. An acceptable device name is "DISK", which must be specified (without quotes) and can be abbreviated to "DI". The ">" character must be omitted if the physical device is omitted, in which case the previous default physical device is not changed. When the physical device is specified, the default unit is canceled if no unit is specified.
unit	Optional string specifying the desired default unit. • For the DISK physical device (i.e., a controller-based disk), this parameter is the letter name of the disk drive of interest. • For the TFTP device, this is the name or IP address (in dotted decimal format) of the TFTP server that will be accessed. The ":" character must not be entered if the unit is not specified. In that case, the previous default unit is not changed (except as noted above). If the "unit" parameter specifies a unit different from the current default, the directory path specified is always started at the top-level directory.

Parameter*1	Description
directory_path	Optional string specifying the directory path of interest. Normally, this parameter will contain file names and backslash (\) characters. For disk devices, V+ adds a "\" if one is not included at the end of a path specification. When the current or specified physical device is not a disk device, a leading "\" specifies that the directory path starts at the top-level directory. The path replaces any default path currently defined (absolute path). The absence of a leading "\" indicates that the path is to be appended to the current default path (relative path). As a special case, nonstandard directories (for example, "[...]" or "/.../") are accepted to assist with referencing other systems.

*1. If the"unit" and "directory_path" parameters are omitted, the unit and the directory path will be canceled in the default. If all parameters are omitted, the current directory path is displayed.

Details

In the following description, the term "directory specification" is used to refer to the combination of physical device and/or disk unit and/or directory path.
When a disk-related V+ operation (for example, FDIRECTORY , LOAD , STORE , and FOPEN_) is processed, the V+ system automatically combines the current "default" directory specification with the directory specification supplied to the command or function. The DEFAULT command can be used to set the directory specification that is to be used in such situations (see the examples below).
The DEFAULT command does not verify that the specified default device, unit, and directory can actually be accessed.
The DEFAULT command can be entered without any parameters to have the current default directory specification displayed on the Monitor screen.
When the V+ system is booted from disk, the initial default disk relationship is set according to the configuration stored on the system disk.

Additional Information

OMRON delivers V+ system boot disks with the default disk unit set to "DISK>D". The default unit and directory path can be changed using Sysmac Studio or ACE software. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

After a DEFAULT command is processed, subsequent disk operations (and DEFAULT commands) will use the new default directory specification as required. The following rules determine the directory specification that will result from a combination of the default specification and the directory specification that is included in any command or function.

1. If no unit is specified, the current default unit will be used. Any directory path specified is appended to the default directory path if the specified path does not start with a backslash (\). Otherwise, the default directory path is ignored. However, the DEFAULT command cancels both the default unit and directory path if both the unit and directory path are omitted.

2. If the unit specified is the same as the current default unit, the specified directory path (if any) is appended to the default directory path if the specified path does not start with a backslash (\). Otherwise, the default directory path is ignored.
3. If the unit specified is different from the current default unit, any directory path specified is always started at the top-level directory of the specified unit. The default directory path is ignored.

Refer to *V+ User's Manual (Cat. No. I671)* for more information disk directories.

Examples

The following examples illustrate how the DEFAULT command can be used to display or set the default directory specification.

Example	Result
DEFAULT	Displays the current default for DISK.
DEFAULT = A:	Changes the default disk drive to unit "A".
DEFAULT= DI>D:\ROB1	Changes the default subdirectory to be the subdirectory "ROB1" on disk unit "D", on the (local) physical device "DISK".
DEFAULT= TEST\DEMO	When used after the previous command, this changes the default directory path to "\ROB1\TEST\DEMO\" (on disk unit "D").

The following examples show how the default directory specification is applied in certain situations. In each case, the current default directory specification is assumed to be "DISK>D:\ROB1\".
In the following example, the whole default directory specification is used.

- Command as entered by user: FDIRECTORY *.V2
- Command as processed: FDIRECTORY DISK>D:\ROB1*.V2

The following example, the device, disk, and root directory are taken from the default directory specification.

- Command as entered by user: FDIRECTORY JOB1\FEED*.*
- Command as processed: FDIRECTORY DISK>D:\ROB1\JOB1\FEED*.*

In the following example, the device and disk are taken from the default directory specification and the default root directory is ignored because the specified path starts with a backslash.

- Command as entered by user: FDIRECTORY \JOB1\FEED*.*
- Command as processed: FDIRECTORY DISK>D:\JOB1\FEED*.*

Related Keywords

3-2-6 CD on page 3-174
3-1-25 $DEFAULT on page 3-35
3-2-25 FDIRECTORY on page 3-200
3-2-29 FSET on page 3-206
3-2-40 LOAD on page 3-222
3-2-59 STORE on page 3-247
3-4-56 FOPEN on page 3-348
3-4-57 FOPENA on page 3-349
3-4-58 FOPEND on page 3-352
3-4-59 FOPENR on page 3-354

3-4-60 FOPENW on page 3-356

3-2-11 DELETE

Monitor command that deletes the specified programs from the system memory.

Syntax

```
DELETE program, ..., program
```

Usage Considerations

A program cannot be deleted while it is executing or is present on an execution stack, as shown by the STATUS function.
DELETE does not delete disk files, but removes programs from system memory. Deleted programs can be reloaded with a LOAD command if the programs have previously been stored to disk. The FDELETE command deletes disk files from a storage disk.
It is good programming practice to group programs into modules, so the DELETEM command would normally be used instead of DELETE.

Parameters

Parameter	Description
program	Name of a program to be deleted.

Details

The DELETE command completely deletes the named programs. This command deletes the programs themselves (like the DELETEP command) and it also deletes all the following items that are used exclusively by the named programs.
Programs and their referenced items are not deleted if they are in an active program execution stack as shown by the STATUS function. A KILL command should be used to clear the appropriate program execution stack before deleting programs referenced in the stack.

Examples

The following example deletes the program named assembly.

```
DELETE assembly
```

Related Keywords

3-2-12 DELETEL on page 3-183
3-2-13 DELETEM on page 3-184
3-2-14 DELETEP on page 3-185

3-2-15 DELETER on page 3-186
3-2-16 DELETES on page 3-188
3-2-24 FDELETE on page 3-199 (monitor command)
3-4-52 FDELETE on page 3-342 (program command)

3-2-12 DELETEL

Monitor command that deletes the named location variables from the system memory.

Syntax

```
DELETEL @task:program loc_variable, ..., loc_variable
```

Usage Considerations

If a fan array element is specified, that element is deleted.
If an array name is specified without explicit index(es) (for example, "DELETEL a[ ]"), the entire array is deleted.
If one or more of the right-most indexes of a multiple-dimension array are omitted, all the elements defined for those indexes are deleted. The following examples demonstrate this.
This example deletes the elements a[3,2,0] to a[3,2,last].

```
DELETEL a[3,2,]
```

This example deletes all the elements a[3,i,j] for all i and j.

```
DELETEL a[3,,]
```

This example deletes the entire array.

```
DELETEL a[,,]
```

Parameters

Parameter	Description
@task:program	These optional parameters specify the context for the location variables. The location variables will be treated as though they are referenced from the specified context. If no context is specified, the location variables will be considered global. Refer to *V+ User's Manual (Cat. No. I671)* for information about variable context.
loc_variable	Name of a location variable to be deleted.

Details

Deletes an arbitrary number of location variables (transformations and/or precision points). This operation can be used to recover the memory storage space occupied by location variables that are no longer needed.

Once a location variable is deleted, it cannot be referenced by any V+ operation or function. An attempt to reference a deleted variable will result in an error message just as if the variable had never been defined.

Examples

The following example deletes the transformation "pick" and the precision point "#park" from memory.

```
DELETEL pick, #park
```

The following example deletes the transformation variable "temp" which is a local variable in the program "main".

```
DELETEL @main temp
```

Related Keywords

3-2-11 DELETE on page 3-182
3-2-15 DELETER on page 3-186
3-2-16 DELETES on page 3-188
3-2-24 FDELETE on page 3-199

3-2-13 DELETEM

Monitor command that deletes the named program module from the system memory.

Syntax

DELETEM module

Usage Considerations

A module will not be deleted if any of its programs are interlocked (see below).
The programs in the module are deleted even if they are referenced by other programs in memory.
DELETEM removes program modules from system memory. It does not erase the associated disk file.

Parameters

Parameter	Description
module	Name of a program module to be deleted.

Details

Deletes a program module and all of its programs from the system memory.
This operation can be used to recover the memory storage space occupied by programs that are no longer needed.

Unlike the DELETE command, DELETEM does not delete subroutines that are referenced by the programs deleted, except when the subroutines are also contained in the specified module. Variables referenced by the deleted programs are not deleted.
If any of the programs in the module are interlocked (see the STATUS real-valued function), those programs are not deleted and the module is not deleted. A KILL command should be used to clear the appropriate program execution stack before deleting a module containing programs referenced in an execution stack.
Refer to *V+ User's Manual (Cat. No. I671)* for information about program modules.

Examples

The following example deletes the program module named "main.package" and all the programs it contains (assuming that none of the programs are interlocked).

```
DELETEM main.package
```

Related Keywords

3-2-11 DELETE on page 3-182
3-2-14 DELETEP on page 3-185
3-2-24 FDELETE on page 3-199
3-2-41 MDIRECTORY on page 3-224
3-2-42 MODULE on page 3-225
3-2-61 STOREM on page 3-250

3-2-14 DELETEP

Monitor command that deletes the named programs from the system memory.

Syntax

```
DELETEP program, ..., program
```

Usage Considerations

A program cannot be deleted while it is executing or is present on an active execution stack, as shown by the STATUS function.
Subroutines and variables referenced by the deleted programs are not deleted. Refer to the *3-2-11 DELETE* on page 3-182 command for more information.
DELETEP removes a program from memory. It does not erase the associated disk file.
It is good programming practice to group programs into modules, so the DELETEM command would normally be used instead of DELETEP.

Parameters

Parameter	Description
program	Name of a program to be deleted.

Details

Deletes an arbitrary number of programs from the system memory. This operation can be used to recover the memory storage space occupied by programs that are no longer needed.
Unlike the DELETE command, DELETEP does not delete subroutines or variables referenced by the named programs.
Programs are not deleted if they are in an active program execution stack as shown by the STATUS function. A KILL command should be used to clear the appropriate program execution stack before deleting programs referenced in the stack.

Examples

The following example deletes the program named "test.one".

```
DELETEP test.one
```

Related Keywords

3-2-11 DELETE on page 3-182
3-2-13 DELETEM on page 3-184
3-2-24 FDELETE on page 3-199
3-2-34 KILL on page 3-215
3-2-58 STATUS on page 3-244

3-2-15 DELETER

Monitor command that deletes the named real-valued variables from the system memory.

Syntax

```
DELETER @task:program real_variable, ..., real_variable
```

Usage Considerations

DELETER does not delete External variables.

Parameters

Parameter	Description
@task:program	These optional parameters specify the context for the real variables. The real variables will be treated as though they are referenced from the specified context. If no context is specified, the real variables will be considered global. Refer to *V+ User's Manual (Cat. No. I671)* for information about variable context. .
real_variable	Name of a real-valued variable to be deleted

Details

Deletes an arbitrary number of real-valued variables. This operation can be used to recover the memory storage space occupied by real-valued variables that are no longer needed.
Once a real-valued variable is deleted, it cannot be referenced by any V+ operation or function. An attempt to reference a deleted variable will result in an error message just as if the variable had never been defined.
If an array element is specified, that element is deleted. The entire array is deleted, however, if an array name is specified without explicit index(es) (for example, "DELETER a[ ]").
If one or more of the right-most indexes of a multiple-dimension array are omitted, all the elements defined for those indexes are deleted. The following examples demonstrate this.
This example deletes the elements a[3,2,0] to a[3,2,last].

```
DELETER a[3,2,]
```

This example deletes all the elements a[3, i, j] for all i and j.

```
DELETER a[3, , ]
```

This example deletes the entire array.

```
DELETER a[, , ]
```

Examples

The following example deletes the real variable "count" and the real array element "x[2]".

```
DELETER count, x[2]
```

The following example deletes the real array "part", which is a local variable in the program "insert".

```
DELETER @insert part[]
```

Related Keywords

3-2-11 DELETE on page 3-182
3-2-12 DELETEL on page 3-183
3-2-16 DELETES on page 3-188
3-2-24 FDELETE on page 3-199

3-2-16 DELETES

Monitor command that deletes the named string variables from the system memory.

Syntax

```
DELETES @task:program string_var, ..., string_var
```

Parameters

Parameter	Description
@task:program	These optional parameters specify the context for the location variables. The location variables will be treated as though they are referenced from the specified context. If no context is specified, the location variables will be considered global. Refer to *V+ User's Manual (Cat. No. I671)* for information about variable context.
string_var	Name of a string variable to be deleted.

Details

Deletes an arbitrary number of string variables. This operation can be used to recover the memory storage space occupied by string variables that are no longer needed.
Once a string variable is deleted, it cannot be referenced by any V+ operation or function. An attempt to reference a deleted variable will result in an error message just as if the variable had never been defined.
If an array element is specified, that element is deleted. The entire array is deleted, however, if an array name is specified without explicit index(es) (for example, "DELETES $a[ ]").
If one or more of the right-most indexes of a multiple-dimension array are omitted, all the elements defined for those indexes are deleted. The following examples demonstrate this.
(For example, the command "DELETES $a[3,2,]" deletes the elements "$a[3,2,0]"to "$a[3,2,last]". The command "DELETES $a[3, ,]" deletes all the elements "$a[3, i, j]" for all "i" and "j". The command "DELETES $a[, ,]" deletes the entire array).
This example deletes the elements $a[3,2,0] to $a[3,2,last].

```
DELETES $a[3,2,]
```

This example deletes all the elements $a[3, i, j] for all i and j.

```
DELETES $a[3, , ]
```

This example deletes the entire array.

```
DELETES $a[, , ]
```

Examples

The following example deletes the string variable "$input".

```
DELETES $input
```

The following example deletes the string variable "$response", which is a local variable in the program "menu".

```
DELETES @menu $response
```

Related Keywords

3-2-11 DELETE on page 3-182
3-2-12 DELETEL on page 3-183
3-2-15 DELETER on page 3-186
3-2-24 FDELETE on page 3-199

3-2-17 DIRECTORY

Monitor command that displays the names of some or all of the programs in the system memory.

Syntax

```
DIRECTORY /switch wildcard_spec
```

Usage Considerations

This command lists the programs resident in system memory. It does not list the files on a disk drive (refer to the *3-2-25 FDIRECTORY* on page 3-200 command).

Parameters

Parameter	Description	
switch	Switch Value	Purpose
	/S	Suppress protected programs
	/?	Display only non-executable programs
	/M	Display only modified programs
wildcard_spec	Optional character string that can include wild cards using either the "?" or "*" character. Both wild cards operate identically and can match 0, 1, or multiple characters.	

Details

The following information can be displayed for each program listed:

- A question mark ("?") is shown at the left if the program cannot be executed. This usually indicates that the program contains a programming error.
- For any program name that is displayed, if the copy in memory has been modified since last being loaded or stored, an "M" is displayed before the program name.
- If the program has restricted access, a code letter for the type of restriction is shown at the left.
  - A "P" is shown if the program is protected. That means the program cannot be displayed, edited, or stored.
  - An "R" is shown if the program is read-only. That means the program cannot be edited or stored.

- The name of the program is displayed.
- If the program is not protected, the entire .PROGRAM command for the program is displayed. Any parameters required by the program are displayed, as is any comment on the .PROGRAM line.
- If the program is protected, the .PROGRAM command is truncated after the program name.

Examples

The following example will display all unexecutable programs in the controller memory including protected programs

```
DIRECTORY /?
```

Related Keywords

3-2-25 FDIRECTORY on page 3-200
3-2-41 MDIRECTORY on page 3-224

3-2-18 DISABLE

Monitor command that turns OFF one or more system switches.

Syntax

```
DISABLE switch, ..., switch
```

Usage Considerations

The DISABLE command can be used while a program is executing.
If a specified switch accepts an index qualifier and the index is 0 or omitted with or without the brackets, all the elements of the switch array are disabled.

Parameters

Parameter	Description
switch	Name of a system switch to be turned OFF. The name can be abbreviated to the minimum length that uniquely identifies the switch. For example, the MESSAGES switch can be referred to as "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. Refer to *V+ User's Manual (Cat. No. I671)* for available system switches and their functions.

When a switch is turned OFF, the feature it controls is no longer functional or available for use. Turning a switch ON with the ENABLE command or program command makes the associated feature functional or available for use.
The system switches are shared by all the program tasks. Consideration 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 monitor command or program command is processed for task 0, an ATTACH program command is executed for the robot, or a CALIBRATE monitor command or program command is processed.
The SWITCH monitor command or the SWITCH function can be used to determine the status of a switch at any time. The SWITCH program command can be used like the DISABLE program command to disable a switch.

Examples

The following example will turn OFF the CP (Continuous Path) switch.

```
DISABLE CP
```

Related Keywords

3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-2-19 DO

Monitor command that executes a keyword(s) as though it were the next step in an executable program or the next step in the specified task / program context.

Syntax

```
DO @task:program instruction
```

Usage Considerations

The specified program task cannot be currently executing.
Refer to *V+ User's Manual (Cat. No. I671)* for information about keywords that can be used with the DO command.

⚠ WARNING

Typing a DO command with no program instruction specified can result in unexpected motion of the robot, because the previous DO operation is executed again.

Parameters

Parameter	Description
program instruction	Optional V+ keyword(s) to be executed. If no keyword is specified, the last keyword executed with a DO command is repeated (regardless of the context specified or assumed).
task	Optional integer that specifies the program task that is to execute the operation. This parameter is also used to determine the context for any variables referenced in the operation. If "task" is omitted, the colon (":") must also be omitted. Then, the main program task (0) or the current debug task is used.
program	Optional program name that specifies the context for any variables or statement labels referenced in the operation. If "program<" is omitted, the colon (":") must also be omitted. Then, the program on top of the stack specified by "task" (or the current debug program) is used. Refer to *V+ User's Manual (Cat. No. I671)* for information about variable context. The last context specified will be used if all the command parameters are omitted. Global context or the current debug program is the initial default.

Details

V+ language keywords can be processed only if they are included in a program and that program is executed. The DO command provides the ability to execute a single keyword without creating a program.

The DO monitor command executes a single operation as though it were contained within a program. This command can be used to move the robot (for example, "DO READY") or to alter the sequence of program step execution (for example, "DO @ 2 GOTO 100").

A new global variable can be created with a DO command only if the program is null. That occurs when no "@" is present or when there is no program on the top of the stack for the specified task.

When a DO command is processed, the following behavior can occur.

- Any temporary robot-configuration or trajectory-control parameter settings for the referenced program task are canceled by a motion performed with a DO command.
- If any REACT, REACTE, REACTI, or RUNSIG commands were active in a program that has been interrupted with a PAUSE command, the commands(s) are re-enabled during execution of the operation in the DO command.

Examples

The following example performs a straight-line motion to the location defined by the transformation "safe.location".

```
DO MOVES safe.location
```

The following example executes an operation in program task 1 to assign the value 5 to the variable i which is a local variable in the program "io.check".

```
DO @1:io.check i = 5
```

Related Keywords

3-4-39 DOS on page 3-322
3-4-38 DO on page 3-321
3-2-22 EXECUTE on page 3-195
3-4-46 EXECUTE on page 3-331
3-2-56 SSTEP on page 3-241
3-2-70 XSTEP on page 3-261

3-2-20 ENABLE

Monitor command that turns ON one or more system switches.

Syntax

ENABLE switch, ..., switch

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 0 or omitted with or without the brackets, all the elements of the switch array are enabled.

Parameters

Parameter	Description
switch	Name of a system switch to be turned ON. The name can be abbreviated to the minimum length that uniquely identifies the switch. For example, the MESSAGES switch can be referred to with "ME", because there is no other switch name that begins with the letters "ME".

Details

System switches control various aspects of the operation of the V+ system. When the high-power enable operation is issued, all robots are checked for out-of-range errors. A message is displayed on the Monitor Window for each robot in error. Refer to *V+ User's Manual (Cat. No. I671)* for available system switches and their functions.
When a switch is turned ON, the feature it controls is functional and available for use. Turning a switch OFF with the DISABLE monitor command or program command makes the associated feature not functional or available for use.

The system switches are shared by all the program tasks. Consideration 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 monitor command or program command is processed for task 0, an ATTACH program command is executed for the robot, or a CALIBRATE monitor command or program command is processed.
The SWITCH monitor command or the SWITCH function can be used to determine the status of a switch at any time.

Examples

The following example will turn ON the MESSAGES system switch.

```
ENABLE MESSAGES
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-2-21 ESTOP

Monitor command that stops the robot in the same manner as if an emergency stop signal was received.

Syntax

ESTOP robot_num

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the operation for all robots are altered. Otherwise, only the operation for the specified robot is affected.

Details

This command immediately initiates an emergency-stop, power-down sequence. Depending on the system configuration, this may or may not include a con- trolled stop before engaging the brakes and de-asserting high power.
No error will be generated if the specified robot_num is not connected.

Examples

The following example initiates an emergency-stop, power down sequence for robot 2.

```
ESTOP 2
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-4-17 BRAKE on page 3-291
3-2-21 ESTOP on page 3-194 (program command)
3-2-44 PANIC on page 3-228 (monitor command)
3-4-91 PANIC on page 3-400 (program command)
3-2-57 STACK on page 3-242

3-2-22 EXECUTE

Monitor command that begins execution of a control program.

Syntax

EXECUTE /C task program(param_list), cycles, step

Usage Considerations

No program can already be active as the specified program task.

⚠ WARNING

Entering an EXECUTE command with no program specified could result in unexpected motion of the robot, since the previous program is executed again.

Parameters

Parameter	Description
/C	Optional qualifier that conditionally attaches the selected robot. The qualifier has an effect only when starting the execution of task 0.
task	Optional integer specifying which program task is to be activated.
program	Optional name of the program to be executed. If the name is omitted, the last program name specified in an EXECUTE monitor command, program command, or PRIME monitor command for the selected task is used.

Parameter	Description
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 and the parentheses may be omitted. 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 function.
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 32767.
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 (after the initial blank and comment lines and all the AUTO, GLOBAL, and LOCAL statements).

Details

This command initiates execution of the specified control program. The program will be executed the number of times specified with the cycles parameter, starting at the specified program step. If no program is specified, the system re-executes the last program executed by the selected program task.
A program can initiate execution of other related programs with the EXECUTE program command. After a program initiates execution of another program, the initiating program can use the STATUS and ERROR functions to monitor the status of the other program.
If the task number is not specified, the EXECUTE command accesses task number 0.
The optional /C qualifier has an effect only when starting execution of task 0. When /C is not specified, an EXECUTE command for task 0 fails if the robot cannot be attached. Attachment requires that the robot is calibrated and that arm power is enabled or that the DRY.RUN system switch is enabled.
When /C is specified, an EXECUTE command 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. This is equivalent to the following program commands.

```
CPON ALWAYS DURATION 0 ALWAYS
FINE 100 ALWAYS
LOCK 0 MULTIPLE ALWAYS NULL ALWAYS
OVERLAP ALWAYS
SPEED 100,100 ALWAYS SELECT ROBOT = 1
```

The robot configuration is saved for subsequent motions.
An execution cycle is terminated when a STOP program command is executed, a RETURN program command is executed in the top-level program, or the last defined step of the program is encountered.

The loop count value can range from -32768 to 32767. 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 program command is executed, an error occurs, or the user (or another program) aborts execution of the program.
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. The robot currently selected is not changed.
If step is specified, the program begins execution at that step. Subsequent cycles always begin at the first executable step of the program.

Examples

The following example initiates execution (as task 0) of the program named "assembly" with execution to continue indefinitely until execution is aborted, a HALT program command is executed, or a run-time error occurs.

```
EXECUTE assembly,-1
```

The following example initiates 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)
```

The following example initiates execution of the last program executed by program task 0 or by the current debug task. No parameters are passed to the program.

```
EXECUTE
```

Related Keywords

3-2-1 ABORT on page 3-167
3-4-1 ABORT on page 3-268 (program command)
3-4-21 CALL on page 3-297
3-2-9 CYCLE.END on page 3-177 (monitor command)
3-4-30 CYCLE.END on page 3-310 (program command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)
3-2-47 PRIME on page 3-232
3-2-48 PROCEED on page 3-233
3-2-52 RETRY on page 3-236 (monitor command)
3-2-56 SSTEP on page 3-241
3-2-58 STATUS on page 3-244
3-1-99 STATE on page 3-121
3-2-70 XSTEP on page 3-261

3-2-23 FCOPY

Monitor command that copies the information in an existing disk file to a new disk file.

Syntax

```
FCOPY new_file = old_file
```

Parameters

Parameter	Description
new_file	File specification for the new disk file to be created. If the period (".") and filename extension are omitted, the default is a blank extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information.
old_file	Specification of an existing disk file. If the period (".") and filename extension are omitted, the default is a blank extension. The current default device, unit, and directory path are considered as appropriate.

Details

If the new file already exists or the old file does not exist, an error is reported and no copying takes place. You cannot overwrite an existing file. The existing file must first be deleted with an FDELETE command.

If the file to be copied has the read-only attribute, the new file will also have that attribute. Files with the protected attribute cannot be copied. Refer to *3-2-25 FDIRECTORY* on page 3-200 for a description of file protection attributes.

When a file is copied, the file creation date and time are preserved along with the standard file attributes. The only attribute that is affected is the archived bit, which is cleared to indicate that the file is not archived.

In general, a file specification consists of the following six elements.

1. An optional physical device (for example, DISK>)
2. An optional disk unit (for example, D:)
3. An optional directory path (for example, DEMO\)
4. A file name (for example, NEWFILE)
5. A period character (".")
6. A file extension (for example, V2)

Examples

The following example creates a file named "newfile.v2" on disk device "D" that is an exact copy of the existing file named "oldfile.v2" on disk device "D".

```
FCOPY D:\newfile.v2 = D:\oldfile.v2
```

Related Keywords

3-2-8 COPY on page 3-176
3-2-10 DEFUALT on page 3-179
3-4-51 FCOPY on page 3-341 (program command)
3-2-28 FRENAME on page 3-205

3-2-24 FDELETE

Monitor command that deletes one or more disk files matching the given file specification.

Syntax

```
FDELETE file_spec
```

Usage Considerations

If a file is deleted, the information in it cannot be recovered.
This command can also delete subdirectory files. Subdirectory files can also be deleted with the FDIRECTORY command.

Parameters

Parameter	Description
file_spec	File specification for the file(s) to be deleted. This may contain an optional physical device, an optional disk unit, and an optional directory path. A file name and a file extension must be specified. The file name or extension may contain wild card matching characters (see below). The current default disk unit and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command. If the file specification does not include a period and a file extension, a blank name extension is assumed.

Details

Wild card characters (asterisks, "*") can be used in file names and extensions. A wild card character within a name or extension indicates that any character should be accepted in that position. A wild card character at the end of a name or extension indicates that any trailing characters are acceptable. All files that match a wild card specification will be deleted. When using the wildcard feature, it is a good idea to issue an FDIRECTORY command with the same file specification first. After verifying that the files listed are the ones you intend to delete, you can issue an FDELETE command with the same file specification.

Examples

The following examples will require the user to respond "Y" to the verification prompt.
The following example deletes the disk file named "f3.lc" from the default device.

```
FDELETE f3.lc
```

The following example deletes all disk files with the extension "V2" from disk "D".

```
FDELETE D:*.v2
```

The following example deletes all disk files with a file name starting with the letters "abc" and file extension starting with the letter "b" from disk "A".

```
FDELETE A:abc*.b*
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-11 DELETE on page 3-182
3-4-52 FDELETE on page 3-342 (program command)
3-2-25 FDIRECTORY on page 3-200

3-2-25 FDIRECTORY

Monitor command that displays information about the files on a disk and the amount of space remaining for storage as well as creates and delete subdirectories on disks.

Syntax

```
FDIRECTORY /qualifier file_spec
```

Usage Considerations

When the parameter /qualifier is specified, no space is allowed between the keyword and the slash (/).
Subdirectories can be nested to a maximum depth of 16. The total length of a directory path specification cannot exceed 80 characters including any defaults.
Subdirectories cannot be deleted if they contain files or if they are being accessed (for example, after an FOPEND operation).

Precautions for Correct Use

The statement "FDELETE *.*" can be used to delete all files in a subdirectory. Use the DEFAULT command to make sure you are in the correct subdirectory before issuing this command.

Parameters

Parameter	Description
/qualifier	Optional qualifier "/C" or "/D", which specifies that a sub-directory is to be created or deleted, respectively. If omitted, a directory listing is generated.
file_spec	Optional file specification string that selects the file(s) to be displayed, or the subdirectory to be created or deleted. If a directory is being displayed, the file specification may contain a physical device, a disk unit, a directory path, a file name, and a file extension. The file name or extension can be omitted or can contain wild card matching characters (see item 6 under details below). If a subdirectory is being created or deleted, the file name and extension must be omitted. A directory must be specified, and it must be terminated with a "\" character. For either function of the command, the default directory specification (set with the DEFAULT monitor command) is used to supply any missing portion of the file specification.

Details

The directory information for the entire default directory is displayed if no file specification or qualifier is entered. The optional file specification can be used to specify the physical device, disk drive, and directory path, and to select the files to be displayed.

When displaying directory information, the command initially displays the directory path used, including any portion obtained from the default directory specification. Then the following information is displayed for each file that satisfies the given file specification.

- The file name
- The file extension
- The number of kilobytes occupied by the file
- Codes for any special attributes for the file
- The date and time the file was written (this may not be present) The display can be aborted by typing CTRL+C at the Monitor Window.

The qualifier "/C" (create) or "/D" (delete) can be appended to the keyword to create or delete a subdirectory . The specific subdirectory to be considered is the last subdirectory that appears in the directory specification resulting from a combination of the current default and the input on the command line. The user is asked for confirmation when deleting a subdirectory.

When creating and deleting subdirectories, all the intermediate subdirectories in the directory specification must already exist. They are not created or deleted. Subdirectories cannot be deleted if they contain any disk files.

A complete file specification consists of the following elements.

1. An optional physical device name followed by a ">" character. An acceptable device name is "DISK" which can be abbreviated to "D". The ">" character must not be entered if the physical device is not specified.

2. An optional disk unit designation followed by a colon (":"). If no disk unit is specified, the current default disk is assumed. For normal V+ disk files, the unit is the letter name of the disk drive of interest such as "A" or "C".
   A colon (":") must terminate the unit if it is specified.
3. An optional directory path which specifies the subdirectory of interest. This parameter should contain file names and backslash (\) characters. A leading slash (\) specifies that the directory path starts at the top-level directory. Any default path currently defined is not used. A directory path specified as a single slash (\) character indicates that the top-level directory is to be accessed. The absence of a leading slash (\) usually indicates that the path is to be appended to the current default path. I the unit specified is different from the current default unit, the directory path is assumed to start at the top-level directory-even if no leading slash (\) is specified.
4. An optional file name with one to eight characters.
5. A period character (.). This can be omitted if no extension is entered. Omitting the period is equivalent to specifying "*" for the file extension.
6. An optional file extension with up to three characters.

File names and extensions can include wild card characters (*). A wild card character within a file name or extension indicates that any character should be accepted in that position. A wild card character at the end of a file name or extension indicates that any trailing characters are acceptable. wild card characters cannot be used in specifications of directory paths.
Disk files can have special attributes to restrict their use. The attributes listed below are used with V+ program packages. When an attribute has been applied to a file, the corresponding letter is displayed by the FDIRECTORY command.

Attribute	Description
P	Protected file. The file can be loaded into the system memory and the programs contained in the file can be executed. The programs cannot be edited, displayed, or traced during execution. The programs cannot be stored from memory onto a disk. Protected files cannot be copied from one disk to another with the FCOPY monitor command nor can they be displayed with the FLIST monitor command or accessed by application programs.
R	Read-only file. The file can be loaded into the system memory and the programs contained in the file can be executed and displayed. The programs cannot be edited or stored from memory onto a disk. Read-only files can be copied from one disk to another with the FCOPY monitor command and they can be displayed with the FLIST monitor command. Application programs cannot overwrite them.

Examples

The following example will display directory information for all the files on the default disk in the current default directory.

```
FDIRECTORY
```

The following example will display information for all the files with the name "demo" in subdirectory "v1" on disk "A".

```
FDIRECTORY A:\v1\demo
```

The following example will display all the files in the default directory that have three-character names beginning with "f" and ending with "n".

```
FDIRECTORY f*n
```

The following example will display all the files in the default directory with names beginning with "f".

```
FDIRECTORY f*
```

The following example will display all the files in the default directory with the extension "lc".

```
FDIRECTORY .lc
```

The following example will display all the files with the extension "v2" in the top-level directory on the default disk.

FDIRECTORY\.v2

The following example will create subdirectory "v1" in the top-level directory on disk "C".

```
FDIRECTORY/C C:\v1\
```

The following example will create subdirectory "t" within subdirectory "v1" on disk "C".

```
FDIRECTORY/C C:\v1\t\
```

The following example will delete subdirectory "t" from the current default directory path on disk "A" (or from the top-level directory if the current default unit is not "A").

```
FDIRECTORY/D A:t\
```

Related Keywords

3-2-23 FCOPY on page 3-197
3-2-26 FLIST on page 3-203
3-2-24 FDELETE on page 3-199
3-4-56 FOPEN on page 3-348
3-2-10 DEFUALT on page 3-179

3-2-26 FLIST

Monitor command that lists the contents of the specified disk file on the Monitor Window.

Syntax

FLIST file_spec

Usage Considerations

To abort the listing operation, press CTRL+C.
The listing operation does not affect programs and data in the system memory.

Parameters

Parameter	Description
file_spec	File specification for the file(s) to be listed. This may contain an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information

Details

This command lists the contents of any disk file on the Monitor Window that contains standard ASCII text. This command is useful for examining an V+ application program stored on the disk without loading it into memory. All disk files generated by the V+ "STORE_" commands can be read using this command.
Disk files with the protected attribute cannot be listed. Protected files are indicated by a "P" in the output from the FDIRECTORY command.

Examples

The following example lists the contents of the disk file "test.v2" on the Monitor Window.

```
FLIST test.v2
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-25 FDIRECTORY on page 3-200
3-2-37 LISTP on page 3-218

3-2-27 FREE

Monitor command that displays the percentage of available system memory not currently in use.

Syntax

FREE

Details

This command displays information about the status of system memory usage. This information provides the percentage of the memory available for application programs and variables that is not currently utilized.
The information returned from the FREE command appears as follows where "nn.nn" represents the percentage value.

```
% unused program memory = nn.nn
```

All system program memory that does not contain V+ system software is available to store application programs and data.If vision or servo tasks are running on the CPU, their programs are considered program memory.

Some operations may result in the error message, "Not enough storage area" even though the FREE operation indicates that a small percentage of program memory is available. This can happen because the unused memory is fragmented into pieces that are too small to store application information. If this happens, store the entire contents of memory onto disk, ZERO memory, and reload your programs and variables from disk.

If you receive any other error messages, memory may be corrupted. Save your programs and issue another FREE command. If the error persists, restart your controller. If the error persists after a restart, contact your local Omron representative.

Additional Information

Refer to *V+ User's Manual (Cat. No. I671)* for error information.

Examples

The following example will return the percentage of available system memory not currently in use.

```
FREE
```

The following information will be returned (as an example).

```
% unused program memory = 67.03
```

Related Keywords

3-2-27 FREE on page 3-204
3-2-71 ZERO on page 3-263

3-2-28 FRENAME

Monitor command that changes the name of a disk file.

Syntax

```
FRENAME new_file = old_file
```

Parameters

Parameter	Description
new_file	New name of the disk file. If it is necessary to locate the file, this parameter can include an optional physical device, an optional disk unit, and an optional directory path in addition to the file name and extension. The current default device, unit, and directory path are considered as appropriate (refer to the *3-2-10 DEFUALT* on page 3-179 command for more information). An error will occur if this name is already in use.
old_file	Nam of the file that is to be renamed. If the file does not exist, an error will occur. No device, disk unit, or directory path may be specified.

Details

Only the name of the file is changed. The file contents and size are not affected. The disk unit (if specified) must be on the left as shown in the example below.

Examples

The following example changes the name of the existing file "data.v2" on disk "D" to the name "parts.v2".

```
FRENAME D:\parts.v2 = data.v2
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197
3-2-49 RENAME on page 3-234

3-2-29 FSET

Monitor command that sets or modifies attributes of a network device.

Syntax

FSET device attribute_list

Parameters

Parameter	Description
device	Name of the physical device whose attributes are to be changed. The name can be abbreviated. Refer to the *3-4-11 ATTACH* on page 3-279 program command for a description of unit numbers and names.
attribute_list	List of keywords and arguments which specify attributes for this device. Refer to the description of the *3-4-63 FSET* on page 3-361 program command for detailed information on valid keywords.

Details

● Using FSET with TCP

You may define new nodes on the network using the FSET command to access the TCP device. You can use the attributes listed below when accessing this device with the FSET command.

- /ADDRESS
- IP Address
- /NODE Node name

Examples

User the following example to define a new network node called "SERVER2" with the IP address 192.9.200.22.

```
fset tcp /node 'SERVER2' /address 192 9 200 22
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-63 FSET on page 3-361(program command)

3-2-30 HERE

Monitor command that defines the value of a transformation or precision-point variable to be equal to the current robot location.

Syntax

```
HERE @task:program loc_variable
```

Usage Considerations

If no task is specified, the HERE command returns information for the robot selected by the V+ monitor (with the SELECT monitor command). If a task is specified, the command returns the location of the robot selected by that task (with the SELECT program command).
If the V+ system is not configured to control a robot, use of the HERE command will cause an error.

Parameters

Parameter	Description
@task:program	These optional parameters specify the context for the location variable. The location variable will be treated as though it is referenced from the specified context. If no context is specified, the location variable will be considered global.
loc_variable	Transformation, precision point, or a compound transformation that ends with a transformation variable.

Details

This command defines the value of a transformation or precision-point variable to be equal to the current robot location.
Normally, the robot location is determined by reading the instantaneous values of the joint encoders. If the robot has either backlash or linearity compensation enabled, the commanded robot location is used instead..
If a compound transformation is specified, only the rightmost element of the compound transformation will be given a value by this command. An error message results if any other transformation in the compound transformation is not already defined.

Examples

The following example defines the transformation "place" to be equal to the current robot location.

```
HERE place
```

The following example assigns the current location of the robot to the precision point "#pick", which is treated as a local variable in the program "test".

```
HERE @test #pick
```

Related Keywords

3-4-67 HERE on page 3-366 (program command)
3-1-48 HERE on page 3-68 (transformation function)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-69 WHERE on page 3-260

3-2-31 ID

Monitor command that displays identity information about components of the system.

Syntax

```
ID
```

Usage Considerations

If this command is issued when there is no robot connected, the robot section is not displayed.

Details

The option words contain coded information about the system (formatted as hexadecimal values). This information is useful to support personnel when troubleshooting the system.
If a user-defined startup message has been defined, that message is displayed in the Monitor Window when an ID monitor command is issued.
Use the example below to understand the information that is returned when the ID monitor command is issued.

```
Software:    3.0 87-1000 (Edit C8, 14-Jun-2019, Production Release)
Controller: 000-0000000
eV+ Emulator Security ID: 0000-0000-0000
Robot 1:    571-0 1000-0-0 6
Adept eCobra 800 Robot (Professional).
```

The software information returned is described below.

```
Software: <version>.<revision> <opt1>-<opt2> (<ID message>)
```

Variable	Description
version	the V+ version number, which is the same value that is returned by the real-valued function statement ID (3).
revision	the V+ revision number, which is the same value that is returned by the real-valued function statement ID (4).
opt1	the system option word 1, which is the same value that is returned by the real-valued function statement ID(5).
opt2	the system option word 2, which is the same value that is returned by the real-valued function statement ID(6).
ID message	the identifying string, which is the same value that is returned by the real-valued function statement $ID(-1).

The controller information returned is described below.

```
Controller: <model>-<serial><revision> <opt>
```

Variable	Description
model	the controller model number, which is the same value that is returned by the function ID(1).

Variable	Description
serial	the controller serial number, which is the same value that is returned by the function ID(2).
revision	the controller hardware revision/ID, displayed as a hexadecimal value. (This value is returned by the function ID(10).)
options	the controller option word, which is the same value that is returned by the function ID(8).

The Security ID information returned is described below.

```
SecurityID: <aaaa-bbbb-cccc>
```

Variable	Description
aaaa-bbbb-cccc	the 12-character hexadecimal card security code used to associate option licenses with specific controllers.

The Processor information returned is described below.

```
Processor n: <version>.<revision> <type>-<opt> <Memory>Mb
```

Variable	Description
n	the CPU number, which is the same value that is returned by the function ID(1,4, *n*).
version	the CPU hardware version code, which is the same value that is returned by the function ID(3,4, *n*).
revision	the CPU hardware revision code, which is the same value that is returned by the function ID(4,4, *n).*
type	the CPU type, which is the same value that is returned by the function ID(5,4, *n*).
opt	indicates active system software components on this processor, which is the value that is returned by ID (6,4, n). Bit 1 = V+ Bit 2 = Vision Bit 3 = Servo
memory	the RAM size, in megabytes, which is the same value that is returned by the expression ID(7,4, *n*) / 1024.

```
Robot n: <model>-<serial> <opt1-opt2> <cat> <module> <module name>
```

Variable	Description
model	the robot model number, which is the value that is returned by the function ID(1,10+n).
serial	the robot serial number, which is the value that is returned by the function ID(2,10+n).
opt1	robot option word 1, which is the value that is returned by the function ID(8,10+n).
opt2	robot option word 2, which is the value that is returned by the function ID(11,10+n).
cat	a single digit indicating the robot safety level that is returned by the function ID(13,10+n):
module	These values are the kinematic device module number
module name	[returned by the function ID(5,10+n)] and name (which is not available from a function).

Examples

The following information is displayed in the Monitor Window when the ID monitor command is issued under the following conditions.

- The system is operating in Emulation Mode.
- There is one eCobra 800 configured in the system.
- The V+ software version is 3 and the revision is 0.
- SystemOption 1 = 87^H (10000111)

```
Software:    3.0 87-1000 (Edit C8, 14-Jun-2019, Production Release)
Controller: 000-0000000
eV+ Emulator Security ID: 0000-0000-0000
Robot 1:    571-0 1000-0-0 6
Adept eCobra 800 Robot (Professional).
```

Related Keywords

3-1-51 ID on page 3-70ID (real-valued function)

3-2-32 IO

Monitor command that displays the current states of external digital input / output signals or internal software signals.

Syntax

```
IO signal_group
```

Usage Considerations

This monitor command will return the value of signals that are mapped in the host and returns "-" for signals that are not mapped.

Parameters

Parameter	Description
signal_group	Optional integer value that selects which digital signals are to be displayed.

Details

The IO command can be used to monitor the system digital signals. If no signal group is specified, all the input and output signals are displayed.
If a signal group is specified, the value must be a value from 0 to 3. Displaying a single group is useful when the system has many signals installed.

Signal group	Signals displayed
0	Digital output signals
1	Digital input signals
2	System software signals
3	The 3000 series of digital output signals
4	Signals mapped in the Host

A "1" is displayed for each signal that is ON, a "0" is displayed for each signal that is OFF, and a "-" is displayed for each signal that does not have a hardware configuration.

Examples

The following example is a sample display from an IO 0 command.

```
0032-0001 ---- ---- ---- ---- ---- ---- 0000 0110
0064-0033 0000 0000 0000 0000 0100 0000 0000 0000
```

This display indicates that signals 2, 3, and 47 are ON. All others are OFF or not installed.

Related Keywords

3-2-50 RESET on page 3-235
3-1-93 SIG on page 3-115
3-1-91 SIG.INS on page 3-113
3-2-54 SIGNAL on page 3-238 (monitor command)
3-4-121 SINGLE on page 3-438 (program command)

3-2-33 JOG

Monitor command that moves the specified joint of the robot, or moves the robot tool along the specified Cartesian axis. Each time JOG is executed, the robot moves for up to 300 ms.

Syntax

```
JOG (status) robot, mode, axis, speed, location, appro_dist
```

Usage Considerations

Each time JOG is executed, the robot moves for the time specified with the JOG.TIME system parameter
The specified robot cannot be attached by any task when using a mode other than COMP. Otherwise, the error message *Robot interlocked* is generated.
After the robot is moved with the JOG command, the system is left in MANUAL mode (i.e., as though a manual mode had been selected on the pendant). JOG mode 5 (or the pendant) can be used to restore COMP mode. Otherwise, an error
COMP mode disabled will be returned when a task attempts to attach the robot.

If a joint is out of range, the JOG command can be used to return the joint back into range. Refer to the details section below for more information.

Parameters

Parameter	Description	
status	An optional status variable. Returns 1 for success; otherwise, contains a V+ error code.	
robot	Specifies the robot number.	
mode	Specifies the jog mode as described below.	
	-1	Keep-alive mode. Continues the previous operation for the time specified using the JOG.TIME system parameter.
	1	Free joint mode. A positive speed will put the specified joint(s) in Free mode. A negative speed will take the specified joint(s) out of Free mode.
	2	Individual joint control.
	3	World coordinates control.
	4	Tool coordinates control.
	5	Restore COMP mode.
	6	unused.
	7	Jog toward the specified location using the specified speed.
	8	Jog toward alignment of the robot tool-Z axis with the nearest World axis.
	9	Cartesian control relative to a frame defined by the specified location.
axis	Specifies the joint number or Cartesian coordinate (X=1, Y=2, ...), depending on the specified jog mode (see above), for the desired motion. This parameter is ignored for modes 7 and 8, but a value must always be specified.	
speed	Specifies the speed and direction of the motion. This is interpreted as a percentage of the speed in manual mode. Values above 100 are interpreted as 100%, values below - 100 are interpreted as -100%. If Free mode is specified, a positive speed will put the given joint in free mode and a negative speed will put the joint out of free mode.	
location	Optional transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move. This parameter is ignored and can be omitted for all modes except 7 and 9.	
appro_dist	Optional real-valued expression that specifies the distance. The position of the destination location is offset from the given location by the distance given, measured along the Z-axis of the specified location in the negative direction. A positive distance sets the tool back (negative tool-Z) from the specified location. A negative distance offsets the tool forward (positive tool-Z). This parameter is used only for mode 7.	

Details

When the status variable is supplied and there is an error, the JOG command does not directly return an error. The error is simply returned in the status variable.

Each time the JOG command is executed, the robot moves for the time specified with the JOG.TIME system parameter. Another JOG can be executed before the previous motion is completed. For extended smooth motion, subsequent JOG commands should be executed within the JOG.TIME value of the previous JOG command. The keep-alive mode can be used for that purpose. The keep-alive mode will have no effect after the timeout of the JOG.TIME value. It has an effect only before the robot stops.

The following error conditions can be reported when the command is processed:

- Mode 1: The error *Illegal joint number* (-609) is returned if FREE mode is not permitted for the specified joint.
- Mode 2: The error *Joint control of robot not possible* (-938) is returned if the robot does not support joint control.
- Modes 3, 4, 8, 9: The error *Cartesian control of robot not possible* (-635) is returned if the robot does not support Cartesian control.
- Mode 7: If the location cannot be reached, the motion stops at the limit of possible motion and the error *Location out of range* (-610) is returned when the motion stops. If any other motion error occurs during the motion (e.g., an obstacle is encountered), the associated error is reported.
- Modes 7 and 9: The error *Missing argument* (-454) is returned if a location is not specified. For mode 7, a straight-line motion is performed toward the specified location if the location is specified with a transformation. A joint-interpolated motion is performed if the location is specified with a precision point. However, if the robot does not permit the type of motion associated with how the location is specified (e.g., the robot does not permit joint-interpolated motion), the motion is performed in the manner that is permitted by the robot.When a robot joint is out of range, it can be driven into range in either of these methods:
- Enter MAN mode on the pendant and manually control the joint.
- Put the pendant in COMP mode and use the JOG command to move the joint back into range (JOG is allowed only in pendant COMP mode).

Use of COMP mode when a joint is out of range is very restricted. All motion commands (except JOG) return a *Position out of range* error in that situation. In addition, JOG can move the joint only in the direction that moves the joint back into range.

Additional Information

Refer to *V+ User's Manual (Cat. No. I671)* for error information.

Examples

The following example will jog joint 3 in a negative direction in joint mode.

```
JOG 1, 2, 3, -10
```

The following example will jog the X-axis in World mode.

```
JOG 1, 4, 2, 10
```

The following example will jog towards the location defined as "loc1".

```
JOG 1, 7, 1, 10,loc1
```

The following example will jog to a position 50 mm above the location defined as "loc1".

```
JOG 1, 7, 1, 10, loc1, 50
```

Related Keywords

3-4-40 DRIVE on page 3-324
3-4-71 JMOVE on page 3-370
3-4-72 JOG on page 3-371 (program command)
3-5-3 JOG.TIME on page 3-470
3-4-81 MOVE on page 3-384 (program command)

3-2-34 KILL

Monitor command that clears a program execution stack and detaches any I/O devices that are attached.

Syntax

```
KILL task
```

Usage Considerations

The KILL monitor command cannot be used while the specified program task is executing.
The KILL monitor command has no effect if the specified task execution stack is empty.

Parameters

Parameter	Description
task	Optional real value, variable, or expression interpreted as an integer that specifies which program task is to be cleared.

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 pro- gram termination.
This situation can occur if a program executes a PAUSE program command or is terminated by an ABORT monitor command or program command, or an error condition while an I/O device is attached or a file is open. If a limited access I/O device is left attached, no other program task can use that device until it is detached.
When the task number is not specified, the KILL command accesses task number 0.

Examples

The following example will remove any program from task 10 if the program is not executing.

```
KILL 10
```

Related Keywords

3-4-1 ABORT on page 3-268
3-4-46 EXECUTE on page 3-331
3-2-58 STATUS on page 3-244

3-2-35 LIST

Monitor command that displays the value of the expression.

Syntax

```
LIST @task:program expression, ..., expression
```

Parameters

Parameter	Description
@task:program	These optional parameters specify the context for any location variables specified. The location variables are treated as though they are referenced from the specified context. If no context is specified, the location variables are considered global. Refer to *V+ User's Manual (Cat. No. I671)* for more information about variable context. If global context is used and the BASE , DEST , HERE ,or TOOL keywords are referenced, the functions return information for the robot selected by the V + monitor (refer to the *3-2-53 SELECT* on page 3-237 monitor command for more information).
expression	Optional real-valued constant, function, variable,or expression whose value is to be displayed.

Details

This command allows expressions of any type to be displayed. Unlike the LISTx commands, it does not display multiple array elements if the right-hand index is left blank.
If an error occurs, a line containing the error message is displayed for the corresponding expression. If no error occurs, the value of the expression is displayed in the following format where type is one of the data types shown below, and the value format depends on the type.

```
type = value
```

Data type	Value
REAL	A single numeric value
TRANS	Six numeric values for X, Y, Z, y, p, r
PPOINT	N numeric values for J1 to Jn of the precision point
STRING	A quoted string
UNKNOWN	An error string that begins with *

Related Keywords

3-2-26 FLIST on page 3-203 (monitor command)
3-2-36 LISTL on page 3-217 (monitor command)
3-2-37 LISTP on page 3-218
3-2-38 LISTR on page 3-219 (monitor command)
3-2-39 LISTS on page 3-220 (monitor command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-36 LISTL

Monitor command that displays the values of the listed locations.

Syntax

```
LISTL /N @task:program location, ..., location
```

Parameters

Parameter	Description
/N	If /N is specified, only the names of the variables are displayed and not the values. Only a single line is displayed for each array with empty brackets to show the array dimensions. If the argument list contains a location parameter, the /N is ignored.
@task:program	These optional parameters specify the context for any location variables specified. The location variables are treated as though they are referenced from the specified context. If no context is specified, the location variables are considered global. Refer to *V+ User's Manual (Cat. No. I671)* for more information about variable context. If global context is used and the BASE , DEST , HERE ,or TOOL functions are referenced, the functions return information for the robot selected by the V + monitor (refer to the *3-2-53 SELECT* on page 3-237 monitor command for more information)..
location	Optional transformation, precision point, location function, or compound transformation whose value is to be displayed.

Details

If no parameters are specified, the values of all global location variables are displayed in alphabetical order. If a program context is specified but no variables are listed, all the location variables local to that program are displayed.

Each location parameter can include any number of wild card characters and each wild card can match 0, 1, or multiple characters.
If an array element is specified, that element is displayed. The entire array is displayed if an array name is specified without explicit index(es). If one or more of the right-most indexes of a multiple-dimension array are omitted, all the elements defined for those indexes are displayed. For example, the statement"LISTL a[3,2,]" displays the elements a[3,2,0] to a[3,2,last].

Examples

The following example will display the values of transformation "pick", compound transformation "hold:part", and the current tool transformation on the Monitor Window.

```
LISTL pick,hold:part,TOOL
```

The following example will display the values of all location variables defined as local to program "test".

```
LISTL @test
```

Related Keywords

3-2-36 LISTL on page 3-217 (monitor command)
3-2-37 LISTP on page 3-218
3-2-38 LISTR on page 3-219 (monitor command)
3-2-39 LISTS on page 3-220 (monitor command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-37 LISTP

Monitor command that displays all the steps of the listed user programs.

Syntax

LISTP program, ..., program

Usage Considerations

Protected programs cannot be displayed.
The programs must be resident in system memory.

Parameters

Parameter	Description
program	Optional name of an application program to be displayed.

Details

If one or more programs are specified on the command line, those programs are displayed on the Monitor Window. If no program names are specified, this command displays all programs in the system memory that are not protected from access.
When a program reports that it is not executable and no errors are displayed in the V+ Editor, executing this monitor command will expose program lines with problems by displaying a question mark (?) in the left margin.

Related Keywords

3-2-26 FLIST on page 3-203
3-2-35 LIST on page 3-216
3-2-36 LISTL on page 3-217
3-2-38 LISTR on page 3-219
3-2-39 LISTS on page 3-220

3-2-38 LISTR

Monitor command that displays the values of the real expressions specified.

Syntax

```
LISTR /N @task:program expression, ..., expression
```

Parameters

Parameter	Description
/N	If /N is specified, only the names of the variables are displayed and not the values. Only a single line is displayed for each array with empty brackets to show the array dimensions. If the argument list contains an expression parameter, the /N is ignored.
@task:program	These optional parameters specify the context for any real-valued variables or task-specific functions specified. The real-valued variables and functions are treated as though they are referenced from the specified context. If no context is specified, global context is used. Refer to *V+ User's Manual (Cat. No. I671)* for more information about variable context. If global context is used and the BASE , DEST , HERE ,or TOOL functions are referenced, the functions return information for the robot selected by the V + monitor (refer to the *3-2-53 SELECT* on page 3-237 monitor command for more information).
expression	Optional real-valued constant, function, variable, or expression whose value is to be displayed.

Details

If no parameters are specified, the values of all global real-valued variables are displayed. If a program context is specified, but no expressions are listed, all of the real-valued variables local to that program are displayed.
Each expression parameter can include any number of wild card characters and each wild card can match 0, 1, or multiple characters.
If an array element is specified, that element is displayed. The entire array is displayed if an array name is specified without explicit index(es). If one or more of the right-most indexes of a multiple-dimension array are omitted, all the elements defined for those indexes are displayed. For example, the statement "LISTR b[3,2,]" displays the elements b[3,2,0] to b[3,2,last]. The statement "LISTR b [, ,]" displays the entire array.
Some functions return information associated with a specific V+ program task (for example, IOSTAT and PRIORITY).
When referenced by a LISTR command, such functions return values for the program task specified by the context parameter (@task). If no task context is specified, such functions return values for task 0.

Examples

The following example displays the value of the real variable "loop.count" and the current value of system TIMER number 2 on the Monitor Window.

```
LISTR loop.count, TIMER(2)
```

Related Keywords

3-2-35 LIST on page 3-216 (monitor command)
3-2-36 LISTL on page 3-217 (monitor command)
3-2-37 LISTP on page 3-218
3-2-36 LISTL on page 3-217 (monitor command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-39 LISTS

Monitor command that displays the values of the specified strings.

Syntax

```
LISTS /N @task:program string, ..., string
```

Parameters

Parameter	Description
/N	If /N is specified, only the names of the variables are displayed and not the values. Only a single line is displayed for each array without index(es), with brackets and 0 to 2 commas displayed to show the array dimensions. The/N option has no effect on a string argument that is specified as a string constant, function, or expression.
@task:program	These optional parameters specify the context for any string variables specified. The string variables are treated as though they are referenced from the specified context. If no context is specified, the string variables are considered global. Refer to *V+ User's Manual (Cat. No. I671)* for more information about variable context.
string	Optional string constant, function, variable, or expression whose value is to be displayed.

Details

If no parameters are specified, the values of all global string variables are displayed in alphabetical order. If a program context is specified, but no variables are listed, all the string variables local to that program are displayed.

Each string parameter can include any number of wild card characters and each wild card can match 0, 1, or multiple characters.

If an array element is specified, that element is displayed. The entire array is displayed if an array name is specified without explicit index(es) (for example,"$line[ ]"). If one or more of the right-most indexes of a multiple-dimension array are omitted, all the elements defined for those indexes are displayed. For example, the command "LISTS $c[3,2,]" displays the elements $c[3,2,0] to $c [3,2,last].

The LISTS command uses the following special methods to display certain characters in string values.

- ASCII control characters (values 0 to 31 decimal) are displayed as two-character sequences, each consisting of a circumflex character ("^", 94 decimal) followed by the character with ASCII value equal to the actual control character plus 96 (decimal). For example, a carriage-return character (13 decimal) is converted to "^m" (13 + 96 = 109, which is the ASCII value for "m").
- A double quote character ("", 34 decimal) is displayed as "^".
- A circumflex character ("^", 94 decimal) is displayed as "^^".
- A byte with the parity bit set (high-order bit of the 8-bit byte) is not distinguished by the LISTS command. LISTS will display both $CHR (^B11000001) and $CHR(^B01000001) as "A".

Examples

The following example displays the value of the string variable "$message" and the text of the last error message in the Monitor Window.

```
LISTS $message, $ERROR(ERROR(0))
```

Related Keywords

3-2-35 LIST on page 3-216 (monitor command)
3-2-36 LISTL on page 3-217 (monitor command)
3-2-37 LISTP on page 3-218
3-2-38 LISTR on page 3-219 (monitor command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-40 LOAD

Monitor command that loads the contents of the specified disk file into the system memory.

Syntax

```
LOAD /qualifier file_spec
```

Usage Considerations

When a file is loaded in read-only mode, only the first program in the file is listed on the monitor screen (if /Q was not specified to suppress all program names).

⚠ CAUTION

When a file is loaded with the /S switch, you should also use /R to prevent others from mistakenly editing the squeezed version of a file and possibly overwriting the un-squeezed version when the changes are saved.

Parameters

Parameter	Description	
/qualifier	Swith Value	Purpose
	/Q	Suppress the listing of program names to the monitor screen when the file is loaded.
	/S	Squeeze programs as they are loaded into memory in much the same way that the SQUEEZE utility program operates on program files. Squeeze programs as they are loaded into memory in much the same way that the SQUEEZE utility program operates on program files.
	/R	Force all of the loaded programs to be in read-only mode except when other considerations dictate that a more secure mode be utilized.

Parameter	Description
file_spec	File specification for the disk file from which programs and variables are to be loaded. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information. If no filename extension is specified, the extension ".V2" is appended to the name given if a local disk is to be accessed.

Details

This command loads the contents of the specified disk file into the system memory. The disk file can contain programs and/or variables, but it must have the format produced by the V+ STORE_ commands.

If an attempt is made to load a program that has the same name as a program already in memory, an error message is displayed and the new program is not loaded. The currently loaded program must be deleted with the DELETE command or memory must be zeroed with the ZERO command before a new pro- gram of the same name can be loaded with the LOAD command.

If a location variable, real variable, or string variable already exists in memory that has the same name as one contained in the disk file, the previous variable is deleted and replaced by the information on the diskette without warning.

If a program is being loaded into the system and there is a program line that V+ cannot process, an error message appears on the monitor screen and the line is marked with a question mark. You can then use the V+ editor to modify the program after it is completely read into memory. This is useful, for example, when you load a program that was composed off-line.

When a LOAD command loads programs into memory, all the programs are entered in a program module with the same name as the first program read from the disk file. The program module is created if it does not already exist. The programs loaded are entered into the module in the order that they are read from disk. Refer to the description of the *3-2-42 MODULE* on page 3-225 command for an explanation of program modules.

The autostart feature available with V+ systems allows you to automatically issue a LOAD command when the robot system is powered on and V+ is loaded from disk. Refer to *V+ User's Manual (Cat. No. I671)* for more information.

If the file is in special, binary program format, the LOAD command automatically applies the special manipulation the file requires.

Examples

The following example loads the contents of the file PALLET.V2 from disk D (the SD Card).

```
LOAD D:pallet
```

The following example loads all the programs contained in file F3.PG into the system memory.

```
LOAD f3.pg
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-42 MODULE on page 3-225
3-2-59 STORE on page 3-247
3-2-60 STOREL on page 3-249
3-2-61 STOREM on page 3-250
3-2-62 STOREP on page 3-252
3-2-63 STORER on page 3-253
3-2-64 STORES on page 3-254

3-2-41 MDIRECTORY

Monitor command that displays the names of all the program modules in the system memory or the names of the programs in a specified program module.

Syntax

```
MDIRECTORY /M module
```

Parameters

Parameter	Description
/M	Optional qualifier that specifies that only modified modules or modified programs within a module are listed.
module	Optional name of a program module in memory. All the modules in memory are listed if this para- meter is omitted. If this parameter is specified, all the programs in the named module are listed.

Details

This command can be used to obtain information about the program modules currently defined in the system memory. Program modules are automatically created by the LOAD command. The MODULE command can be used to create, expand, or rearrange a program module.
When the command parameters are omitted, the MDIRECTORY command lists the names of all program modules currently in memory.
For any module or program name that is displayed, if the copy in memory has been modified since last being loaded or stored, an "M" appears before the program or module name.
When the module parameter is specified, the MDIRECTORY command functions like a DIRECTORY command, except only programs in the specified module are listed in the sequence they follow in the module. Refer to the *3-2-17 DIRECTORY* on page 3-189 command for details about the information displayed.
The order of programs in a module can be changed with the MODULE monitor command.

Examples

The following example displays the names of all of the program modules in memory.

```
MDIRECTORY main.package
```

Related Keywords

3-2-13 DELETEM on page 3-184
3-2-17 DIRECTORY on page 3-189
3-2-40 LOAD on page 3-222
3-2-42 MODULE on page 3-225
3-2-61 STOREM on page 3-250

3-2-42 MODULE

Monitor command that creates a new program module, or modifies the contents of an existing module.

Syntax

```
MODULE module = program, ..., program
```

Parameters

Parameter	Description
module	Name of a program module.
program	Name of a program in memory.

Details

A program module is a group of programs that can be referred to by a single name. The following monitor commands can be used to access program modules:

- LOAD creates a new module if necessary and enters programs in the module as they are read from a disk file.
- DELETEM deletes all the programs contained in a module and deletes the module.
- MDIRECTORY lists either all the modules currently in memory or all the programs in a named module.
- MODULE either creates a new module or modifies the contents of an existing module.
- STOREM stores in a disk file all the programs in a module.

Program modules are created automatically by the LOAD monitor command when a program file is read from disk. The MODULE command can be used to create new program modules, or to expand or rearrange the contents of modules already defined in the system memory.

If the specified program module does not already exist, the MODULE command will create the module. In that case, all the listed programs will be placed in the new module.

If the specified module does exist, the MODULE command will add the listed programs at the end of the module. If any of the programs are already in the specified module, they will be moved to the end of the module.
Each program in memory may belong to one module at most. Whenever a program is added to a module and the program is already in another module, the program will be removed from its previous module.

Examples

If there is no program module named "system.1" in memory, the example below will create that module and put three programs into it. If there is a program module named "system.1" in memory, this command will add the three programs to the module.

```
MODULE system.1 = main.program, subroutine.1, subroutine.2
```

Related Keywords

3-2-13 DELETEM on page 3-184
3-2-40 LOAD on page 3-222
3-2-41 MDIRECTORY on page 3-224
3-2-42 MODULE on page 3-225
3-2-61 STOREM on page 3-250

3-2-43 NET

Monitor command that displays status information about the network. Also displays details about the remote mounts that are currently defined in the V+ system.

Syntax

NET mode

Parameters

The mode parameter is an optional value that indicates what part of the status information is to be displayed. The values for the mode parameter are described in the table below.
Omitting the optional mode parameter will have the same function as specifying a 0 value.

Value	Description
0	Display network status information as shown below. .NET 0 TCP/IP: Up FTP: Option not installed Local IP address: 192.9.222.252 Packets transmitted: 105007 Packets received: 577717 Transmission errors: 0 Reception errors: 6 Missed packets: 0 Available TCP connections: 32 Mount Node Path XC ASERVER C:/ADEPT/DISKS/DISK_C
1	Display additional information about errors to the output as shown below. .NET 1 TCP/IP: Up FTP: Option not installed Local IP address: 192.9.222.252 Packets transmitted: 105007 Packets received: 577875 Transmission errors: 0 Reception errors: 6 Missed packets: 0 Reception framing error: 0 Reception CRC error: 3 Reception overflow: 3 Reception buffer error: 0 Transmission late collision: 0 Transmission loss of carrier: 0 Transmission retry error: 0 Transmission buffer error/underflow: 0 Available TCP connections: 32 Mount Node Path XC ASERVER C:/ADEPT/DISKS/DISK_C
2	Display network status and IP addresses of the defined nodes as shown below. .NET 2 TCP/IP: Up FTP: Option not installed Name IP address ASERVER 192.9.222.252

Details

This command displays information about the TCP/IP protocol.

There can be any one of three network states as described in the table below.

Network	Description
Up	This indicates that the network has initialized successfully and is running.
Option installed	The software option is installed,but the protocol is not currently active.
Hardware not installed	The Ethernet cable is not connected.

If TCP/IP is in the Up state, additional information is also displayed as shown in the following example.

Examples

If the TCP/IP network is in the Up state, issuing a NET command will return information as shown in the example below (your values may be different).

```
Local IP address        192.9.222.252
Packets transmitted Packets received        105007
577717    Trans-
mission errors    0
Reception errors        0
Missed packets        0    Avail-
able TCP Connections    32
```

The following additional output is displayed only if mode is specified with a nonzero value.

```
Reception framing error    0
Reception CRC error    0
Reception overflow    0
Reception buffer error    0
Transmission late collision    0
Transmission loss of carrier    0
Transmission retry error    0
Transmission buffer error/underflow    0
```

Related Keywords

3-1-68 NETWORK on page 3-95

3-2-44 PANIC

Monitor command that simulates an external E-stop button press, stops all robots immediately but does not turn OFF robot high power.

Syntax

PANIC robot_num

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the operation for all robots are altered. Otherwise, only the operation for the specified robot is affected.

Details

This command performs the following actions:

- Immediately stops robot motion.
- Stops execution of the robot control program if the robot is attached and no REACTE operation has been executed to enable program processing of error.

Unlike pressing the emergency stop button on the manual control pendant, high power remains enabled after a PANIC operation is processed.

Examples

The following monitor command example will simulate an external E-stop button press, stop robot 2 immediately, but does not turn OFF robot high power.

```
PANIC 2
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-21 ESTOP on page 3-194 (monitor command)
3-4-45 ESTOP on page 3-330 (program command)
3-4-91 PANIC on page 3-400 (program command)

3-2-45 PARAMETER

Monitor command that sets or displays the values of system parameters.

Syntax

```
PARAMETER parameter[index] = value
```

Usage Considerations

If the equal sign and value are omitted, the system parameter is not changed and its current value is displayed in the Monitor Window.

If no system parameter is specified, the current values of all system parameters are displayed in the Monitor Window.
Refer to *V+ User's Manual (Cat. No. I671)* for more information about system parameters.

Parameters

Parameter	Description
parameter	Name of the system parameter whose value is to be displayed or modified. The name must be specified when the equal sign and a value are included in the command to modify the system parameter value. The name can be omitted when the command is being used to display system parameter values. When specified, the system parameter name can be abbreviated.
index	For system 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.
value	Optional real value, variable, or expression defining the value to be assigned to the system parameter. The equal sign must be omitted if no value is specified.

Details

If a value is specified, the specified system parameter is set to the value on the right-hand side of the equal sign. The parameter name can be abbreviated to the minimum length that identifies it uniquely. The PARAMETER command can be used without any arguments to see a list of all the system parameters available with your V+ system. A subset of the complete list can be requested by providing an abbreviation for the system parameter name and no input value. Then, all the system parameters with names beginning with the specified root will be displayed with their cur- rent values (see examples below).
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 modified or displayed.
If the system parameter name is omitted but an index is specified, the values of all parameters without indexes are displayed along with the specified element of all parameter arrays.

Examples

The following example sets the BELT.MODE system parameter to 4.

```
PARAMETER BELT.MODE = 4
```

The following example displays the current settings of the system parameters with names that begin with "B".

```
PARAMETER B
```

Related Keywords

3-5-1 BELT.MODE on page 3-468

3-5-4 NOT.CALIBRATED on page 3-471
3-4-92 PARAMETER on page 3-400 (program command)
3-1-75 PARAMETER on page 3-100 (real-valued function)

3-2-46 PING

Monitor command that tests the network connection to a node.

Syntax

```
PING node
```

Parameters

Parameter	Description
node	Name or IP address of the network node with which communication will be attempted. If a node name is used, it must have been defined in the V+ configuration file or by an FSET monitor command keyword or program command keyword.

Details

This command tests the network connection to an addressed node. If the node responds, the command displays "Success". If the node does not respond within 5 seconds, the command displays "Node not reachable".

Examples

The following example will return a message of "Success" if a connection to the node name "server2" was successful. If the connection was unsuccessful, the message "Node not reachable" will be returned.

```
ping server2
```

The following example will return a message of "Success" if a connection to the node with an IP address of 172.16.200.1 was successful. If the connection was unsuccessful, the message "Node not reachable" will be returned.

```
ping 172.16.200.1
```

Related Keywords

3-2-29 FSET on page 3-206 (monitor command)
3-4-63 FSET on page 3-361 (program command)
3-2-43 NET on page 3-226

3-2-47 PRIME

Monitor command that prepares a program for execution but does not start execution.

Syntax

```
PRIME task program(param_list), cycles, step
```

Usage Considerations

PRIME resets the execution stack for the selected program execution task and cancels the context of any program that is paused for that task.
A PRIME command cannot be processed while the selected program task is already active.
This command can be used only when the external Front Panel key switch is set to AUTOMATIC mode.

Parameters

Parameter	Description
task	Optiona linteger number that specifies which system pro- gram task is to be activated.
program	Optional name of the program to be executed. If the name is omitted, the last program name specified in an EXECUTE monitor command, program command, or PRIME monitor command for the selected task is used.
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 and the parentheses may be omitted. 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 pro- gram as undefined and can be detected with the DE- FINED function.
cycles	Optional real value, variable, or expression interpreted as an integer that specifies the number of program ex- ecution 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 al- lowed is 32767.
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 after the initial blank and comment lines and all the AUTO, GLOBAL, and LOCAL statements.

Details

This command prepares a program for execution. It can be considered as being the same as the EXECUTE monitor command, except that PRIME does not actually start program execution.
After a program is primed, execution can be started with the PROCEED monitor command.

Examples

The following example will prepare the "rob.move" program on task 1.

```
PRIME 1 rob.move()
```

Related Keywords

3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-62 STOREP on page 3-252
3-2-70 XSTEP on page 3-261

3-2-48 PROCEED

Monitor command that resumes execution of an application program.

Syntax

```
PROCEED task
```

Usage Considerations

A program cannot resume if it has completed execution normally or has stopped due to a HALT program command.

Parameters

Parameter	Description
task	Real value, variable, or expression interpreted as an integer that specifies which program task is to be executed.If no task number is specified, task number 0 is assumed.

Details

This command resumes execution of the specified program task at the step following the one where execution was halted due to a PAUSE program command , an ABORT monitor command, a breakpoint, single-step execution, or a runtime error.

In addition to continuing execution of a suspended program, this command can be used to initiate execution of a program that has been prepared for execution with the PRIME monitor command.
If the specified task is executing and the program is at a WAIT or WAIT.EVENT program command (for example, waiting for an external signal condition to be satisfied), entering PROCEED in the Monitor Window has the effect of skipping over the or WAIT.EVENT program commands.
This command has no effect if the specified task is executing and the program is not at a WAIT or WAIT.EVENT program command.
PROCEED differs from RETRY. If a program statement generated an error, RETRY attempts to re-execute that statement, but PROCEED resumes execution at the statement that follows. If a robot motion was in progress when the program stopped, RETRY attempts to complete that motion where PROCEED advances to the next motion

Examples

The following example will continue execution of a paused program on task 2.

```
PROCEED 2
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-47 PRIME on page 3-232
3-2-52 RETRY on page 3-236
3-2-56 SSTEP on page 3-241
3-2-58 STATUS on page 3-244
3-2-70 XSTEP on page 3-261

3-2-49 RENAME

Monitor command that changes the name of a user program in memory to the new name provided.

Syntax

```
RENAME new_program = old_program
```

Usage Considerations

RENAME can be used on any program in memory. If the program is currently executing, an error "Program Interlocked" will be returned. If a program is stopped, but still on the stack, the program will be removed from the stack (similar to KILL) and renamed in memory. RENAME does not change the name of a disk file. It changes the name of a program resident in system memory.

Parameters

Parameter	Description
new_program	New name for the program.
old_program	Current name of the program.

Details

If there is already a program in the system memory with the specified new name, the RENAME operation is not performed and an error message is dis- played. In this case, you must first delete the existing program with the new name to perform the RENAME operation.
If the user program being renamed is currently assigned to a program module, the program is removed from the module and assigned with the new name to the global module.
Refer to the *3-2-42 MODULE* on page 3-225 command for information about program modules.

Examples

The following example changes the name of program "oldprog" to "newprog".

```
RENAME newprog=oldprog
```

Related Keywords

3-2-8 COPY on page 3-176
3-2-28 FRENAME on page 3-205

3-2-50 RESET

Monitor command that will turn OFF all the digital output signals.

Syntax

RESET

Usage Considerations

This monitor command has no effect on Host I/O (external) signal numbers 4001 to 4999.
The RESET program command is useful in the initialization portion of a program to ensure that all the external output signals are in a known state.

⚠ WARNING

Do not issue this command unless you are sure all output signals can be safely turned OFF. Be particularly careful of devices that are activated when a signal is turned OFF.

Related Keywords

3-2-4 BITS on page 3-171 (monitor command)
3-4-16 BITS on page 3-290 (program command)
3-1-12 BITS on page 3-17 (real-valued function)
3-2-32 IO on page 3-211
3-1-93 SIG on page 3-115
3-1-91 SIG.INS on page 3-113
3-2-54 SIGNAL on page 3-238 (monitor command)
3-4-120 SIGNAL on page 3-437 (program command)

3-2-51 RESET.LOCK

Monitor command that detaches a robot from the application program.

Syntax

```
RESET.LOCK
```

Usage Considerations

This command will detach all robots that are currently attached.

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318

3-2-52 RETRY

Monitor command that repeats execution of the last interrupted statement and continues execution of the program.

Syntax

```
RETRY task
```

Usage Considerations

RETRY cannot be processed when the specified task is executing.
A program cannot be resumed if it has completed execution normally or has stopped due to a HALT program command.

Parameters

Parameter	Description
task	Real value, variable, or expression interpreted as an integer that specifies which program task is to be executed. If no task number is specified, task number 0 is assumed.

Details

This command restarts execution of the specified task similar to the PROCEED monitor command. After a RETRY command, the point at which execution resumes depends on the status at the time execution was interrupted. If a pro- gram step or robot motion was interrupted before its completion, use of a RETRY command causes the interrupted operation to be completed before execution continues normally. This allows you to retry a step that has been aborted or that caused an error.
If no program step or robot motion was interrupted, the RETRY command has the same effect as the PROCEED command.
When a RETRY command is used to resume an interrupted motion, all motion parameters are restored to the settings in effect before the motion was interrupted.

Examples

The following example will resume from where the task number defined by "task.num" was interrupted (if the task is not executing or has not been stopped from the HALT operation).

```
RETRY task.num
```

Related Keywords

3-2-48 PROCEED on page 3-233
3-2-56 SSTEP on page 3-241
3-2-58 STATUS on page 3-244
3-2-70 XSTEP on page 3-261

3-2-53 SELECT

Monitor command that selects a robot for subsequent Monitor Window operations.

Syntax

SELECTdevice_type = unit

Usage Considerations

The SELECT monitor command can only be used with Robot Integrated Control systems.

The SELECT command can be used only if there are multiple devices of the same type connected to your system.
If the SELECT command is not issued, automatic selection will occur for the first robot of the system, starting from the lowest robot identifier (1).
The SELECT command affects only the robot selection for the Monitor Window (i.e., for subsequent monitor commands, such as HERE and WHERE). If you want to change the selection for a program task, you must execute the SELECT program command in that program task, either within a program, or by using the DO monitor command. For example, DO @1 SELECT ROBOT = 2 changes the robot selection for program task 1.

Parameters

Parameter	Description
device_type	Keyword that identifies the type of device that is to be selected. The only valid device type is ROBOT.
unit	Real value, variable, or expression interpreted as an integer that specifies the particular robot to be selected. The values that are accepted depend on the configuration of the system.

Details

In a multiple-robot system, the SELECT monitor command specifies which robot the Monitor Window will access.
If the robot number specified for the unit parameter is not valid, a (-407) *Invalid argument* error occurs.
If the robot number specified for the unit parameter is valid, but the robot is not configured in the system, a (-622) *No robot connected to system* error occurs.
If the robot number specified for the unit parameter is valid and the robot is configured in the system, a (1) *Success* information message occurs.

Examples

The following example will select robot 2 for subsequent Monitor Window operations:

```
SELECT ROBOT = 2
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-54 SIGNAL

Monitor command that turns ON or OFF digital output signals, internal software signals, or host signals.

Syntax

```
SIGNAL signal, ..., signal
```

Usage Considerations

To control host signals with the SIGNAL command, they must be assigned in the range of 4001 to 4999 in the NJ-series Robot Integrated CPU Unit using Sysmac Studio or ACE software. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Parameters

Parameter	Description
signal	Real-valued expression that evaluates to a digital output or internal signal number. A positive value indicates ON, a negative value indicates OFF. SIGNAL ignores parameters with a zero value.

Details

The value of the signal parameter determines which signal is to be controlled. The following values for the signal parameter are available.

- Digital outputs connected to XIO and IO Blox of each robot: 1 to 999
- Robot outputs for end effectors such as a gripper: 3001 to 3004
- Internal software signals: 2001 to 2999
- Host signals: 4001 to 4999

Refer to *V+ User's Manual (Cat. No. I671)* for more information about default signal allocation when multiple robots are present.

In emulation mode, digital inputs (numbered from 1001 to 1999) can also be con- trolled with the SIGNAL command.

Executing this monitor command keyword for an IOBlox signal that does not physically exist will cause a -508 *Device not ready* error.

Examples

The following example will turn OFF the digital output signal specified by the value of the variable "reset" (assuming the value of "reset" is positive), and turn ON the digital output signal 4.

```
SIGNAL -reset, 4
```

The following example will turn digital output signal 1 OFF, digital output signal 4 ON, and internal software signal 2010 ON.

```
SIGNAL -1, 4, 2010
```

Related Keywords

3-2-4 BITS on page 3-171 (monitor command)
3-4-16 BITS on page 3-290 (program command)
3-1-12 BITS on page 3-17 (real-valued function)
3-2-32 IO on page 3-211
3-2-50 RESET on page 3-235
3-4-114 RUNSIG on page 3-429
3-1-93 SIG on page 3-115
3-1-91 SIG.INS on page 3-113

3-2-55 SPEED

Monitor command that specifies monitor speed.

Syntax

```
SPEED speed_factor
```

Usage Considerations

Monitor speed is limited to 100 or less. If you specify a faster speed, 100 will be assumed. A value of 100 is considered normal, full speed and 50 is 1/2 of full speed.

- Motion speed has different affects for joint-interpolated motions and straight-line motions. Refer to *V + User's Manual (Cat. No. I671)*for more information.
- SPEED settings takes effect immediately including the speed of any currently executing motions.

The speed of robot motions is determined by a combination of the monitor speed setting set with the SPEED monitor command and the program speed setting set by an executing program with a SPEED program command.

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

Parameters

Parameter	Description
speed_factor	Real-valued expression whose value is used as a new speed factor.

Details

The speed at which robot motion occurs is a function of both the speed set by this command and the speed set by a SPEED program command. During a continuous path motion, when the program speed is changed, the path followed is altered to maintain the specified speed and acceleration. However, when the monitor speed is changed, the path is unaffected but the accelerations will be modified.

Monitor speed is applied as a percentage of set program speeds. This allows convenient adjustments to the speed of the robot motions without modification to programs or variables. The relationship of the monitor SPEED, the program SPEED, and the accelerations can be explained as follows.
When the monitor speed is 100%, V+ generates motions that attempt to achieve the specified program speed and acceleration. During continuous path motions, this will result in the path being rounded near intermediate destination locations to prevent excessive accelerations. If the program speed is increased and the accelerations remain constant, the rounding at intermediate points is increased to maintain the acceleration specifications.
If the monitor speed is set below 100%, V+ generates the same path that would have been planned for a monitor speed of 100% and the rounding is unchanged. The duration of each part of the motion (acceleration segments, constant velocity segments, and deceleration segments) are proportionally scaled to slow down the entire motion.
The monitor speed is set to 50 when V+ is initialized. The speed cannot be set lower than 0.000001 [1.0E-6].

Examples

The following example sets the monitor speed to 30%.

```
SPEED 30
```

Related Keywords

3-6-10 SCALE.ACCEL on page 3-484
3-4-124 SPEED on page 3-445 (program command)
3-1-96 SPEED on page 3-118 (real-valued function)

3-2-56 SSTEP

Monitor command that executes a single step or an entire subroutine of a control program.

Syntax

SSTEP task

Usage Considerations

SSTEP (subroutine-step) can be used to single-step any of the available system program tasks independent of the execution status of other system tasks.

Parameters

Parameter	Description
task	Optional integer number that specifies which system program task is to be executed. If no task number is specified, task number 0 is assumed.

Details

If the next program step to be executed is not a CALL or CALLS program command, the SSTEP command operates the same as an XSTEP command without program arguments.
If the program step to be executed is a CALL or CALLS program command, the SSTEP command causes the entire called subroutine to be executed before pro- gram execution stops at the step following the CALL or CALLS program command.
As with the PROCEED and RETRY commands, an SSTEP command can be executed only after single-step execution of the preceding program statement, a PAUSE program command, a breakpoint, or a nonfatal error during program execution.
During single-step execution, the next statement to be executed is displayed in the Monitor Window and the manual control pendant.
The SSTEP status is retained by the V+ system even if program execution stops within the called subroutine and is restarted at that point with additional XSTEP SSTEP command, or with a PROCEED command. Program execution will stop when the original subroutine finally executes a RETURN program command.

Examples

The following example will execute the next step of the program that was executing as task 0.

```
SSTEP
```

The following example will execute the next step of program task 1.

```
SSTEP 1
```

Related Keywords

3-4-21 CALL on page 3-297
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-47 PRIME on page 3-232
3-2-48 PROCEED on page 3-233
3-2-52 RETRY on page 3-236
3-2-58 STATUS on page 3-244
3-2-70 XSTEP on page 3-261

3-2-57 STACK

Monitor command that specifies the amount of system memory reserved for a program task to use for subroutine calls and automatic variables.

Syntax

```
STACK task = size
```

Usage Considerations

This command cannot be executed while any program task is active.
This command cannot be issued using the Monitor Window while the Sysmac Studio or ACE software is connected. This command is typically used with the MC keyword in a Monitor Command program.
This command is not executable while operating in Emulation Mode.
The RETRY monitor command can be used to continue task execution after issuing the STACK command.
If an attempt is made to set the stack size smaller than the amount of stack memory currently in use by the task, the stack size will be set to the size currently in use.
If you do not want to use the default stack size (128 kB), it must be set every time the V+ system is booted from disk. An initialization command program can be used to set the stack sizes in this case.

Parameters

Parameter	Description
task	Real-valued constant, variable, or expression interpreted as an integer that specifies the program task whose stack size is to be changed.
size	Real-valued constant, variable, or expression that specifies the amount of stack space to be reserved in kilobytes. The number of bytes to be reserved is computed by multiplying the size parameter value by 1024..

Details

When subroutine calls are made, V+ uses an internal storage area called a stack to save information required by the subroutine that begins executing. This information includes the following items.

- The name and step number of the calling program.
- Data necessary to access subroutine arguments.
- The values of any automatic variables specified in the called program.

The STACK command allows users to explicitly allocate storage space to the stack for each program task. To optimize the use of system memory, the amount of stack space can be adjusted for a particular application. Stacks can be made arbitrarily large, limited only by the amount of memory available in your system.
If a STACK command cannot allocate the amount of storage requested, it will fail with the error message *Not enough storage area*. If this occurs, take the following steps. These steps will compact the program storage area and may permit a larger stack to be allocated.

1. Reduce the sizes of other program task stacks if possible.
2. Issue a ZERO monitor command to delete all programs in memory
   1) Issue the desired STACK command.
   2) Reload the programs.

If a program task runs out of stack space, it will stop with the error message"*Not enough program stack space*". If this happens, use the STACK monitor command to increase the stack size and then

issue the RETRY monitor command to continue program execution. All the other program tasks must be stopped.
The STATUS command can be used to display the stack statistics for a single program task. The maximum stack value indicates how much stack space was requested by the task that generated the error.

Examples

The following statement will reserve 64 kilobytes of memory on the stack for task 0.

```
STACK 0 = 64
```

Related Keywords

3-4-12 AUTO on page 3-283
3-2-58 STATUS on page 3-244

3-2-58 STATUS

Monitor command that returns status information for the system and the programs being executed.

Syntax

```
STATUS select
```

Usage Considerations

The STATUS command can be used at any time to determine the status of the system.

Parameters

Parameter	Description
select	Optional real value, variable, or expression interpreted as an integer that selects the information to be returned.

Details

If the select parameter is omitted, the status of all the program tasks is returned. The information returned is not updated continuously.
If the value of the select parameter is 0 or positive, it must correspond to one of the program tasks.
The following example provides descriptions of the information returned from the STATUS command when the select parameter is omitted.

```
ROBOT:  Robot power off        Monitor speed:  100
TASK    STATE            MAIN              CURRENT PROGRAM   STEP   CYCLES   STACK
0       Program Running  ai.monitor.jobs    ai.check.job        25        0     2.5
1       Program Running  ai.monitor.jobs    ai.check.jobs.cmp    7       10     2.0
2       Program Wait     ai.monitor.jobs    rn.get.msg.wait     35        0     1.5
3       Not Active       ai.monitor.jobs    ai.help.screen      35        0     0.0
```

● ROBOT

This appears only in systems with a robot present. The following messages are possible for the "ROBOT:" field shown above.

■ Fatal Error

A fatal hardware error has occurred. Robot power is OFF and cannot be turned ON until the error is resolved.

■ Robot power off

Robot power is OFF and can be turned ON if necessary.

■ Not calibrated

Robot power is ON, but the robot is not calibrated and cannot be moved until a CALIBRATE monitor command or program command is issued.

■ COMP mode

Robot power is ON and the robot is enabled for control by an application program.

■ Manual mode

Robot power is ON and the robot is being controlled by the manual control pendant.

● Monitor Speed

The "Monitor speed:" field of the information returned shows the current monitor speed factor.

● TASK

The "TASK" column lists all running tasks

● STATE

The "STATE" field contains messages indicating the current state of each program task as described below.

■ Not active

The task is currently inactive.

■ Program running

The task is executing the program indicated at the right.

■ Program input

The executing program is waiting for input from some I/O device.

■ Program WAIT

The executing program is waiting at a WAIT program command statement.

- **MAIN**

The "MAIN" column indicates the main program that is being executed that is the program that was invoked with an EXECUTE monitor command or program command, or a PRIME or XSTEP monitor command.

- **CURRENT PROGRAM**

The "CURRENT PROGRAM" column is the program that is currently on the top of the stack and is the program that is currently executing. It may be either the main program or a program that was subsequently invoked with a CALL or CALLS program command (or a reaction).

- **STEP**

The "STEP" column indicates the number of the next step to be executed within the current program.

- **CYCLES**

The "CYCLES" column indicates the number of execution cycles of the main program that have been completed when the STATUS command was issued.

- **STACK**

The "STACK" column indicates the present size of the execution stack in kilo-bytes.
The following example provides descriptions of the information returned from the STATUS command when the select parameter has a program task specified.

```
STATE           PROGRAM         STEP        CYCLES       STACK    MAX  LIMIT
Not active      program0        82          0 of 1         0.0    0.0  128.0
```

The "STATE", "PROGRAM", and "STEP" fields in this format are similar to the "STATE","CURRENT PROGRAM", and "STEP" fields in the description above.
If the selected program task is active, only the program on the top of the stack is shown. If the task state is "Not active", then the entire execution stack is shown, beginning with the main program and ending with the top of the stack.

- **CYCLES**

The "CYCLES" column returns the total number of cycles that have been completed, followed by the total number of cycles to be completed. The value -1 indicates that cycles are to be executed indefinitely.

- **STACK**

The "STACK" column indicates the size of the stack in kilobytes currently in use.

- **MAX**

The "MAX" column shows the maximum amount of the stack that has been used since the last time the program was executed. If a program has failed with the "*Not enough program stack space*" error, the MAX field indicates how much stack space was requested by the system. This will return a value to use to real- locate stack space to the task.

- **LIMIT**

  The "LIMIT" field shows the limit on the stack size. This limit may be changed with the STACK monitor command.

Examples

In the example shown below, tasks 0 and 1 are running, task 2 has completed one cycle and is no longer running, and task 3 is inactive and has an empty stack.

```
ROBOT:  Robot power off        Monitor speed:  100

TASK    STATE            MAIN              CURRENT PROGRAM   STEP   CYCLES   STACK
0       Program Running  ai.monitor.jobs   ai.check.job        25        0     2.5
1       Program Running  ai.monitor.jobs   ai.check.jobs.cmp    7       10     2.0
2       Program Wait     ai.monitor.jobs   rn.get.msg.wait     35        0     1.5
3       Not Active       ai.monitor.jobs   ai.help.screen      35        0     0.0
```

Task 1 was started by the request "EXECUTE 1 ai.monitor.jobs" (see the "MAIN" column). The next step to execute is step 25 of program "ai.check.job" which was called either directly or indirectly by the main program "ai.monitor.jobs". The task has not completed any cycles and is using a stack that is currently 2.5 kilo- bytes in size.

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)
3-2-48 PROCEED on page 3-233 (monitor command)
3-2-52 RETRY on page 3-236
3-2-57 STACK on page 3-242
3-1-100 STATUS on page 3-127 (real-valued function)

3-2-59 STORE

Monitor command that stores programs and variables in a disk file.

Syntax

```
STORE /levels file_spec = program, ..., program
```

Usage Considerations

STORE can be used while a program is executing. An executing program can be stored.
There must be sufficient room on the disk to store the new disk file. Otherwise, the store operation will fail.

Loading and storing precision points on a system with less axes than the one which defined them will result in components being lost.
Protected and read-only programs in memory cannot be stored. This monitor command does not store External variables.

Parameters

Parameter	Description
/levels	Optional qualifier that determines the level of program references to consider if a program parameter is specified. If /levels is omitted, all program references are processed as described below. If the qualifier is specified as "/2", for example, only the first two levels of program references are processed.
file_spec	Specification of the disk file into which the programs and variables should be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information. If no file name extension is specified, the extension ".V2" will be appended to the name given (for disk files only.
program	Optional name of a program in memory.

Details

This command creates the specified disk file and stores the following information in the file:

- The specified programs.
- All the subroutines referenced directly and indirectly by the specified programs (unless limited by a "/levels" qualifier).
- All the global (location, real-valued, and string) variables referenced by the programs and subroutines that are stored.

If no program names are specified, all the programs, subroutines, and global variables in memory are saved in the disk file.
This command stores the same information as the separate commands STOREP , STOREL , STORER , and STORES , but the STORE command creates only one file rather than four.
As the programs are stored on the disk, their names are displayed on the Monitor Window. You may see names other than those given on the command line since referenced subroutines are automatically stored. Programs are stored in alphabetical order regardless of the order used in the command.

Examples

The following example creates a file named "F3.V2" on the default disk unit and stores the two programs named "cycle" and "motor" along with all the sub- routines and global variables they reference.

```
STORE f3=cycle,motor
```

The following example creates a file named "F3.V2" on the default disk unit and stores only the program "cycle" and the subroutines it calls directly (but no subroutines called by those subroutines), along with all the global variables referenced by those programs.

```
STORE /2 f3=cycle
```

the following example creates a file named "DEMO.V2" on disk unit "D" and stores all the programs and global variables that are in memory.

```
STORE D:demo
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197
3-2-40 LOAD on page 3-222
3-2-60 STOREL on page 3-249
3-2-61 STOREM on page 3-250
3-2-62 STOREP on page 3-252
3-2-63 STORER on page 3-253
3-2-64 STORES on page 3-254

3-2-60 STOREL

Monitor command that stores location variables in a disk file.

Syntax

```
STOREL /levels file_spec = program, ..., program
```

Usage Considerations

STOREL can be used while a program is executing.
There must be sufficient room on the disk to store the new disk file. Otherwise, the store operation will fail.
Loading and storing precision points on a system with fewer axes than the one which defined them will result in components being lost.

Parameters

Parameter	Description
/levels	Optional qualifier that determines the level of program references to consider if a program parameter is specified. If /levels is omitted, all program references are processed as described below. If the qualifier is specified as "/2", for example, only the first two levels of program references are processed.

Parameter	Description
file_spec	Specification of the disk file into which the programs and variables should be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information. If no file name extension is specified, the extension ".LC" will be appended to the name given (for disk files only).
program	Optional name of a program in memory.

Details

This command stores the names and values of all the global location variables referenced in the specified programs, and in any subroutines referenced by those programs unless limited by the /levels parameter. If no programs are specified,all global location variables with defined values are stored in the disk file.

Examples

The following example will store all the global location variables referenced by the program named "motor" and by all the subroutines referenced by "motor" into a disk file named "F2.LC".

```
STOREL f2=motor
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197
3-2-40 LOAD on page 3-222
3-2-59 STORE on page 3-247
3-2-61 STOREM on page 3-250
3-2-62 STOREP on page 3-252
3-2-63 STORER on page 3-253
3-2-64 STORES on page 3-254

3-2-61 STOREM

Monitor command that stores a specified program module to a disk file.

Syntax

STOREM /qualifiers **file_spec = module**

Usage Considerations

STOREM can be used while a program is executing.
There must be sufficient room on the disk to store the new disk file. Otherwise, the store operation will fail.
Protected and read-only programs in memory cannot be stored.

Parameters

Parameter	Description
/qualifiers	Any combination of up to three qualifiers may be specified to store multiple data types in addition to the module programs. If all the qualifiers are omitted, only the module programs are stored in the specified file. If /L is specified, the global location and precision point variables referenced directly by the module programs are stored in a .LOCATIONS section at the end of the file. If /R is specified, the global real and double variables referenced directly by the module programs are stored in .REAL and .DOUBLE sections at the end of the file. If /S is specified, the global string variables referenced directly by the module programs are stored in a .STRINGS section at the end of the file.
file_spec	Specification of the disk file into which the programs should be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information. If no file name extension is specified, the extension ".PG" will be appended to the name given (for disk files only).
module	Name of a program module in memory.

Details

This command stores in the indicated disk file all the unrestricted programs in the specified program module. As the programs are stored on the disk, their names are displayed on the Monitor Window. Programs are stored in the sequence they follow in the module.

Examples

The following example stores all the programs in the program module named "main" into a disk file named "LINE23.PG".

```
STOREM line23=main
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197
3-2-40 LOAD on page 3-222
3-2-41 MDIRECTORY on page 3-224
3-2-42 MODULE on page 3-225
3-2-59 STORE on page 3-247
3-2-62 STOREP on page 3-252

3-2-62 STOREP

Monitor command that stores program files to a disk file.

Syntax

```
STOREP /levels file_spec = program, ..., program
```

Usage Considerations

STOREP can be used while a program is executing.
There must be sufficient room on the disk to store the new disk file. Otherwise, the store operation will fail.
Protected and read-only programs in memory cannot be stored.
In general, it is good programming practice to group programs into modules, so STOREM should normally be used instead of STOREP.

Parameters

Parameter	Description
/levels	Optional qualifier that determines the level of program references to consider if a program parameter is specified. If /levels is omitted, all program references are processed as described below. If the qualifier is specified as "/2", for example, only the first two levels of program references are processed.
file_spec	Specification of the disk file into which the programs should be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information. If no file name extension is specified, the extension ".PG" will be appended to the name given (for disk files only).
program	Optional name of a program in memory.

Details

This command stores the specified programs in the indicated disk file. In addition to the programs specified, any subroutines referenced by those programs and any subroutines referenced by the subroutines are also automatically stored, unless limited by the /levels parameter. If no program names are given, all the programs in memory are saved in the disk file.
As the programs are stored on the disk, their names are displayed on the Monitor Window. You may see names other than those given on the command line since referenced subroutines are automatically stored. Programs are stored in alphabetical order regardless of the order used in the command.

Examples

The following example stores the program named "test" and all the subroutines referenced by it into a disk file named "F1.NEW".

```
STOREP f1.new=test
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197
3-2-40 LOAD on page 3-222L
3-2-58 STATUS on page 3-244
3-2-60 STOREL on page 3-249
3-2-61 STOREM on page 3-250
3-2-64 STORES on page 3-254

3-2-63 STORER

Monitor command that stores real variables in a disk file.

Syntax

```
STORER /levels file_spec = program, ..., program
```

Usage Considerations

STORER can be used while a program is executing.
There must be sufficient room on the disk to store the new disk file. Otherwise, the store operation will fail.

Parameters

Parameter	Description
/levels	Optional qualifier that determines the level of program references to consider if a program parameter is specified. If /levels is omitted, all program references are processed as described below. If the qualifier is specified as "/2", for example, only the first two levels of program references are processed.
file_spec	Specification of the disk file into which the programs should be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information. If no file name extension is specified, the extension ".PG" will be appended to the name given (for disk files only).
program	Optional name of a program in memory.

Details

This command stores the names and values of all the global real variables referenced in the specified programs, and in any subroutines referenced by those programs unless limited by a the /levels parameter. If no programs are specified, all defined global real variables are stored in the disk file.
Although they can have real values, system parameters are not stored with the STORER command.

Examples

The following example stores all the global real variables referenced by the pro- gram named "cycle" and by all the subroutines referenced by "cycle" into a disk file named "F2.RV".

```
STORER f2=cycle
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197
3-2-40 LOAD on page 3-222
3-2-59 STORE on page 3-247
3-2-60 STOREL on page 3-249
3-2-61 STOREM on page 3-250
3-2-62 STOREP on page 3-252

3-2-64 STORES

Monitor command that stores a string variable in a disk file.

Syntax

```
STORES /levels file_spec = program, ..., program
```

Usage Considerations

STORES can be used while a program is executing.
There must be sufficient room on the disk to store the new disk file. Otherwise, the store operation will fail.

Parameters

Parameter	Description
/levels	Optional qualifier that determines the level of program references to consider if a program parameter is specified. If /levels is omitted, all program references are processed as described below. If the qualifier is specified as "/2", for example, only the first two levels of program references are processed.
file_spec	Specification of the disk file into which the programs should be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information. If no file name extension is specified, the extension ".PG" will be appended to the name given (for disk files only).
program	Optional name of a program in memory.

Details

This command stores the names and values of all global string variables referenced in the specified programs, and in any subroutines referenced by those programs unless limited by the /levels parameter. If no programs are specified, all global defined string variables are stored in the disk file.
Refer to the *3-2-39 LISTS* on page 3-220 command for the format used to store certain special characters.

Examples

The following example stores all the global string variables in system memory into a disk file named "F3.ST".

```
STORES f3
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197
3-2-40 LOAD on page 3-222
3-2-59 STORE on page 3-247
3-2-60 STOREL on page 3-249
3-2-61 STOREM on page 3-250
3-2-62 STOREP on page 3-252
3-2-63 STORER on page 3-253

3-2-65 SWITCH

Monitor command that displays the settings of system switches in the Monitor Window.

Syntax

```
SWITCH switch[index]
```

Usage Considerations

If no switch parameter is provided, the current settings of all switches are displayed.
Refer to *V+ User's Manual (Cat. No. I671)* for more information about system switches.

Parameters

Parameter	Description
switch	Optional name of a switch to be displayed. The switch name can be abbreviated as described below.
index	For switches that can be qualified by an index, this is an optional real value, variable, or expression that designates the specific switch element of interest.

Details

This command displays the settings of the specified switch as "On" (enabled) or "Off" (disabled). If no switch name is specified, the status of all switches is displayed.
A subset of the complete list can be displayed by providing an abbreviation for the switch name. All the switches with names beginning with the specified root will be displayed with their current settings.
If the specified switch accepts an index qualifier and the index is 0 or omitted (with or without the brackets), all the elements of the switch array are displayed
If the switch name is omitted but an index is specified, the values of all switches without indexes are displayed along with the specified element of all switch arrays.

Examples

The following example displays the current settings of the CP switch.

```
SWITCH CP
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-2-66 TESTP

Monitor command that tests for the presence of the named program in the system memory.

Syntax

TESTP program

Parameters

Parameter	Description
program	Name of the program to search for.

Details

This command is primarily useful in Monitor Command programs. A success message is output if the program is found. Otherwise, an error response is returned.

Examples

The following example will generate an error message if the program "move" is not in memory.

```
TESTP move
```

Related Keywords

3-2-17 DIRECTORY on page 3-189
3-2-58 STATUS on page 3-244

3-2-67 TOOL

Monitor command that sets the internal transformation used to represent the location and orientation of the tool tip relative to the tool-mounting flange of the robot.

Syntax

```
TOOL @task:program transform_value
```

Usage Considerations

The TOOL command applies to the robot selected by the V+ monitor with the SELECT monitor command.
The command can be used while programs are executing. However, an error will result if the robot is attached by any executing program.
If the V+ system is not configured to control a robot, use of the TOOL command will cause an error.

Parameters

Parameter	Description
@task:program	These optional parameters specify the context for any variables referenced by the command. The variables will be treated as though they are referenced from the specified context. If no context is specified, the variables will be considered global. Refer to *V+ User's Manual (Cat. No. I671)* for more information about variable context.
transform_value	Optional transformation variable, 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

If no transformation value is specified, the tool transformation is set equal to the null tool. The null tool has its center at the surface of the tool mounting flange and its coordinate axes parallel to those of the last joint of the robot represented by the transformation [0,0,0,0,0,0]. The tool transformation is automatically set equal to the null tool when the system is turned ON and after a ZERO monitor command. The relative tool transformation is automatically taken into consideration each time the location of the robot is requested, when a command is issued to move the robot to a location defined by a transformation, and when manually controlled motions are performed in world or tool mode. Refer to *V+ User's Manual (Cat. No. I671)* for information about how to define a tool transformation.
If the transformation value specified as the argument to this command is modified after the TOOL command is issued, the change does not affect motions of the robot until another TOOL command is issued. For example, if the transformation value is specified by a transformation variable, changes to the value of that variable will not affect the tool transformation until another TOOL command is issued with the variable.

The monitor command statement "LISTL TOOL" can be used to display the current tool setting.

Examples

The following example replaces the current tool transformation with the value of compound transformation "grip:extension".

```
TOOL grip:extension
```

The following example cancels any tool transformation that may be in effect.

```
TOOL
```

Related Keywords

3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-128 TOOL on page 3-451 (program command)
3-1-110 TOOL on page 3-143 (transformation function)

3-2-68 WAIT.START

Monitor command that puts a Monitor Command program into a wait state until a condition is satisfied.

Syntax

WAIT.START condition

Usage Considerations

This command is not intended to be used in the Monitor Window. It is normally used as a step in a Monitor Command program.
You can cancel an activated WAIT.START command by pressing the Emergency Stop button on the external Front Panel or by pressing the Emergency Stop button on the pendant.
Aborting an activated WAIT.START command causes termination of the Monitor Command program containing the command.
Refer to *V+ User's Manual (Cat. No. I671)* for more information about Monitor Command programs.

Parameters

Parameter	Description
condition	Optional real value, variable, or expression that is evaluated for a TRUE (nonzero) or FALSE (0) value.

Details

This command can be used to suspend processing of a Monitor

Command program until a desired condition exists. For example, the state of one or more external signals can be used as the condition for continuation.
If the condition parameter is included in a WAIT.START command, the Monitor Command program is suspended until the condition value makes a transition from FALSE (0) to TRUE (nonzero).
The Monitor Command program is suspended if the condition being tested is already TRUE when the WAIT.START command is executed. A transition from FALSE to TRUE must occur to resume.
The WAIT.START command checks for the specified condition only once every system cycle. There can be a delay of 1 system cycle between satisfaction of the condition and resumption of program execution.

Examples

The following example stops processing of the Monitor Command program until the state of digital input signal 1001 changes from OFF to ON.

```
WAIT.START SIG(1001)
```

Related Keywords

3-4-136 WAIT on page 3-460

3-2-69 WHERE

Monitor command that displays the current location of the robot and the hand opening.

Syntax

WHERE

Usage Considerations

The WHERE command applies to the robot selected by the V+ monitor with the SELECT command. If the V+ system is not configured to control a robot, use of the WHERE command will cause an error.

Details

The location of the robot tool point is displayed in cartesian world coordinates and as joint positions together with the current hand opening.

Examples

The following example shows the output displayed when the WHERE command is issued for a 4-axis robot.

```
.WHERE
Robot 1:
      X mm        Y mm        Z mm         y           p           r          Hand
    550.001       0.003     384.000       0.000     180.000     180.000      1.00
0
       J1          J2          J3          J4          J5          J6
   -42.893      93.372      10.000      50.479
```

Related Keywords

3-2-30 HERE on page 3-207
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-2-70 XSTEP

Monitor command that executes a single step of a program.

Syntax

XSTEP task program (param_list), cycles, step

Usage Considerations

XSTEP can be used to single-step any of the system program tasks, independent of the execution status of other system tasks.

Parameters

Parameter	Description
task	Optional integer that specifies which system program task is to be executed. If no task number is specified, task 0 is assumed.
program*1	Optional name of the application program to be executed.
param_list	Optional parameter list for the program. Refer to the description of the *3-2-22 EXECUTE* on page 3-195 monitor command and program command for details.
cycles	Optional real value, variable, or expression interpreted as an integer that specifies the number of program execution cycles to be performed. Refer to the description of the *3-2-22 EXECUTE* on page 3-195 monitor command and program command for details.
step	Optional real value, variable, or expression interpreted as an integer that specifies the step at which program execution is to begin.

*1. STEP commands that do not include a program name do not affect the temporary time slice and priority parameters.

Details

The XSTEP command can be used to execute an application program one step at a time. This is frequently useful while a program is being developed and pro- gram errors are detected and corrected.
The following aspects of program execution must be considered when using this command.

- The system program task that is to be utilized
- The program that is to be executed
- Program execution is stopped after one command.

The optional task parameter specifies which of the system program tasks is to be activated.
If any of the program arguments (program, param_list, cycles, or step) are specified, program execution is initiated in the same manner as for the EXECUTE monitor command. Unlike the EXECUTE monitor command, the first executable program statement is displayed in the Monitor Window but is not executed.
XSTEP must then be entered again without any program arguments to execute that statement.
If all the command arguments are omitted, the following operations are performed.

1. The displayed program statement is executed.
2. The next statement to be executed is displayed in the Monitor Window.
3. The program is stopped again.

As with the PROCEED and RETRY monitor commands, an XSTEP command with no arguments can be executed only after execution has stopped due to one of the following events.

- An ABORT keyword operation is processed.
- Single-step execution of the preceding program statement.
- A PAUSE program command is executed.
- A breakpoint is encountered.
- Occurrence of a nonfatal error during program execution.

Examples

The following example initiate execution of program "pack" for three cycles as task 0. The parameters "p2" and "17" are passed to the program. The first executable step of "pack" is displayed in anticipation of its execution with a subsequent XSTEP command (without parameters).

```
XSTEP pack(p2,17),3
```

The following example prepares the program "assembly" for execution as program task 0 or the current debug task starting at step number 23. If "XSTEP" is then typed, step 23 will be executed.

```
XSTEP assembly,,23
```

The following example executes the next step of the program executing as program task 2.

```
XSTEP 2
```

The following example executes the next step of the program that was executing as task 0 (or the current debug task).

```
XSTEP
```

The following example moves the execution pointer to step number 45 in the program that was executing as task 0 (or the current debug task).

```
XSTEP ,,45
```

Related Keywords

3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-47 PRIME on page 3-232
3-2-48 PROCEED on page 3-233
3-2-56 SSTEP on page 3-241
3-2-58 STATUS on page 3-244

3-2-71 ZERO

Monitor command that initializes the V+ system and deletes all the programs and data in system memory.

Syntax

```
ZERO
```

Usage Considerations

This command cannot be used when any program task is executing or when a robot is attached to a task.
Any currently selected robot will not be affected by this command. This command cannot be used while online with a controller. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Details

This command initializes the V+ system and deletes all the programs and variables in memory.
The following changes occur when the command is processed.

- All the programs and variables in memory are deleted.
- The program execution stacks are cleared.
- The status line is cleared.
- Any latch data buffers are cleared.
- Any tool transformations are automatically set equal to the null tool when the system is turned ON after a ZERO operation.
- The BASE transformation is automatically set equal to the null transform when the system is turned ON after a ZERO operation.
- All logical units are detached (except the robot).
- Movement parameters such as SPEED, FINE, etc. are reset to default. Other parameters and switches are not affected.

Examples

The following example deletes all programs and data in system memory.

```
ZERO
```

Related Keywords

3-2-11 DELETE on page 3-182
3-2-12 DELETEL on page 3-183
3-2-13 DELETEM on page 3-184
3-2-14 DELETEP on page 3-185
3-2-15 DELETER on page 3-186
3-2-16 DELETES on page 3-188

3-3 Other Keywords

Use the information in this section to understand other keywords and their use with the V+ system.

3-3-1 .END

Keyword that marks the end of an V+ program.

Syntax

```
.END
```

Usage Considerations

The V+ editors automatically add this line to the end of every program.
This keyword must exist at the end of every V+ program. Programs missing the .END keyword do not load correctly into the V+ system.

Example

The following example will display the string "Valid Program" in the Monitor Window.

```
.PROGRAM program1()
    TYPE "Valid Program"
.END
```

Related Keywords

3-4-101 .PROGRAM on page 3-411

3-3-2 IPS

Specify the units for a SPEED program command as inches per second.

Syntax

```
SPEED value IPS ALWAYS
```

Usage Considerations

IPS can be used only as a parameter for a SPEED program command.
The speed setting specified is scaled by the monitor speed in effect when the robot motion occurs.
Speeds specified with the IPS keyword apply to straight-line motions. Joint-interpolated motions do not maintain the specified tool speed.

To specify speed in millimeters per second, use the MMPS keyword.

Details

IPS is an optional parameter for the SPEED program command which specifies the units to be used for the speed value. When IPS is specified in a SPEED program command, the speed value is interpreted as inches per second for straight-line motions.
Refer to the *3-4-124 SPEED* on page 3-445 program command for further details on setting motion speeds with the IPS keyword.

Example

The following example sets the robot tool tip speed to 20 inches per second for the next straight-line robot motion (assuming the monitor speed is set to 100).

```
SPEED 20 IPS
```

Related Keywords

3-3-3 MMPS on page 3-266
3-2-55 SPEED on page 3-240 (monitor command)
3-4-124 SPEED on page 3-445 (program command)
3-1-96 SPEED on page 3-118 (real-valued function)

3-3-3 MMPS

Specify the units for a SPEED program command as millimeters per second.

Syntax

SPEED value MMPS ALWAYS

Usage Considerations

MMP Scan be used only as a parameter for a SPEED program command.
The speed setting specified is scaled by the monitor speed in effect when the robot motion occurs.
Speeds specified with the MMPS keyword apply to straight-line motions. Joint-interpolated motions do not maintain the specified tool speed.
To specify units in inches per second, use the IPS keyword.

Details

MMPS is an optional parameter for the SPEED program command which specifies the units to be used for the speed value. When MMPS is specified in a SPEED program command, the speed value is interpreted as millimeters per second for straight-line motions.

Refer to the *3-4-124 SPEED* on page 3-445 program command for further details on setting motion speeds with the MMPS keyword.

Example

The following example sets 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

3-3-2 IPS on page 3-265
3-2-55 SPEED on page 3-240 (monitor command)
3-4-124 SPEED on page 3-445 (program command)
3-1-96 SPEED on page 3-118 (real-valued function)

3-4 Program Command Keywords

Use the information in this section to understand program command keywords and their use with the V + system.

3-4-1 ABORT

Terminate execution of an executing program task.

Syntax

ABORT task_num

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 logical units. If the program has one or more files open and you decide not to resume execution of the program, use the KILL keyword to close all the files and detach the logical units.

Parameters

Parameter	Description
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

The ABORT program command 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 keyword.

Example

The following example will terminate the execution of the program on task 1 after moving to the "safe.loc" position.

```
MOVE safe.loc BREAK
ABORT 1
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)

3-2-9 CYCLE.END on page 3-177 (monitor command)
3-4-30 CYCLE.END on page 3-310 (program command)
3-2-21 ESTOP on page 3-194 (monitor command)
3-4-45 ESTOP on page 3-330 (program command)
3-2-22 EXECUTE on page 3-195
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)
3-2-44 PANIC on page 3-228 (monitor command)
3-4-91 PANIC on page 3-400 (program command)
3-4-100 PROCEED on page 3-410
3-2-58 STATUS on page 3-244 (monitor command)
3-1-100 STATUS on page 3-127 (real-valued function)

3-4-2 ABOVE

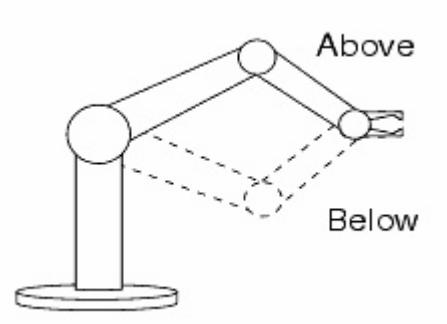
Request a change in the robot configuration during the next motion so that the elbow is above the line from the shoulder to the wrist.

Syntax

ABOVE

Usage Considerations

SCARA and parallel robots cannot have an above configuration.
Configuration changes cannot be made during straight-line motions.
If the selected robot does not support an above configuration, this keyword is ignored by the robot.
The ABOVE program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. If the robot is not attached, this command has no effect.
This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the ABOVE program command causes an error.
The following figure shows the ABOVE and BELOW configurations.



Example

The following example will move the robot to "point1" using the ABOVE configuration.

```
ABOVE
MOVE point1 BREAK
```

Related Keywords

3-4-15 BELOW on page 3-289
3-1-18 CONFIG on page 3-24
3-4-115 SELECT on page 3-431 (program command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-3 ACCEL

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

Syntax

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

Usage Considerations

The ACCEL program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing this command 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).
If an acceleration profile that is not defined is initially specified in the program, the system will default to acceleration profile 1 for eCobra and Viper robots.
If an acceleration profile that is not defined is initially specified in the program, the system will default to acceleration profile 2 for iX3 and iX4 robots.
If an acceleration profile that is defined (2, for example) is specified in the program and then later in the same program, an acceleration profile is specified that does not exist (7, for example), the program will continue to use the previously specified profile (2, for example).
If a parameter is omitted, the current setting remains in effect.

Parameters

Parameter	Description
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.

Parameter	Description
acceleration	Optional real value, variable, or expression considered as a percentage of the maximum possible acceleration.
deceleration	Optional real value, variable, or expression considered as a percentage of the maximum possible deceleration. The value should normally be in the range of 1 to 100. If an out-of-range value is specified, the nearest limit will be used.

Details

If profile 0 is used, a square wave acceleration profile is 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 command define the maximum acceleration and deceleration magnitudes that are achieved.

When the V+ system is initialized, the profile, acceleration, and deceleration values are set to initial values. The initially selected profile may be 0, 1, or 2 depending on the type of robot.The settings are not affected when program execution starts or stops or when a ZERO monitor command is processed.

Default acceleration and deceleration values of 100% are set for use with typical robot payloads and link inertias. However, because the actual attainable values 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. If you specify a higher acceleration or deceleration than is permitted, the default values are used.

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

For a given motion, the maximum attainable acceleration may be less than what is requested. This occurs when a profile with a nonzero acceleration ramp time is used and there is insufficient time to reach the maximum acceleration. A specific time must elapse before the acceleration can be changed from 0 to the specified maximum value. If the maximum acceleration cannot be achieved, the trapezoidal profile is reduced to a triangular shape. This occurs during the following circumstances.

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

If you increase the maximum acceleration and deceleration values but the motion time does not change, try increasing 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. This type of acceleration limiting cannot occur with acceleration profile 0 because a square-wave acceleration instantaneously changes acceleration values without ramping.

Example

The following example sets the default acceleration time to 50% of normal and the deceleration time to 30% of normal.

```
ACCEL 50, 30
```

The following example changes the deceleration time to 60% of normal and does not change the acceleration time.

```
ACCEL ,60
```

The following example reduces the acceleration and deceleration to one half of their current settings.

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

The following example invokes defined profile 2 and sets the acceleration magnitude to 80% of the defined rate.

```
ACCEL (2) 80
```

Related Keywords

3-1-2 ACCEL on page 3-8 (real-valued function)
3-1-30 DURATION on page 3-41
3-6-10 SCALE.ACCEL on page 3-484
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)
3-2-55 SPEED on page 3-240

3-4-4 ALIGN

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

Syntax

ALIGN

Usage Considerations

The ALIGN program command can be executed by any program task as long as the task has attached a robot. The command applies to the robot selected by the task.
If the system is not configured to control a robot, executing this command causes an error.

Details

The ALIGN program Command causes the tool to be rotated so that its Z-axis is aligned parallel to the nearest axis of the world coordinate system. This command is primarily useful for aligning the tool before a series of locations is taught. This typically accomplished with the use of the DO monitor command.

Example

The following example will move the robot to "point1" and align the tool's z-axis to the nearest world axis.

```
MOVE point1
ALIGN
```

Related Keywords

3-4-115 SELECT on page 3-431 (program command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-5 ALTER

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

Syntax

ALTER Dx, Dy, Dz, Rx, Ry, Rz

Usage Considerations

This program command can be executed by the task that is controlling a robot in alter mode or by any other task that has selected the robot (using the SELECT keyword).
Refer to the *3-4-7 ALTON* on page 3-275 program command for more information about enabling alter mode.
This command is ignored if the selected robot is not in alter mode.
When alter mode is enabled, this command should be executed once during each trajectory cycle. If this command is executed more often, only the last set of values defined during each cycle will be used.

Parameters

Parameter	Description
Dx	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 0. In non-cumulative mode, omitted coordinates default to the values specified in the previous ALTER command. Distances are interpreted as millimeters; angles are interpreted as degrees.
Dy	
Dz	
Rx	
Ry	
Rz	

Details

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

Example

The following statements can be embedded in a program loop that uses sensor data to control the motion of the robot.

```
ALTER 0.1*sx, , 0.2*sz
WAIT
```

Related Keywords

3-4-6 ALTOFF on page 3-274
3-4-7 ALTON on page 3-275
3-1-99 STATE on page 3-121

3-4-6 ALTOFF

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

Syntax

ALTOFF

Usage Considerations

A robot must be attached by the program task prior to executing this program command. Turning OFF alter mode causes a break in continuous-path motion.

Details

This program command suspends program execution until any previous robot motion has been completed and then turns OFF real-time path modification mode (alter mode). After alter mode is OFF, the robot position is 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

3-4-5 ALTER on page 3-273
3-4-7 ALTON on page 3-275

3-1-99 STATE on page 3-121

3-4-7 ALTON

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

Syntax

ALTON mode

Usage Considerations

A robot must be attached by the program task prior to executing this program command. Alter mode cannot be active at the time this program command 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.
For V+ , you can modify the robot path by periodically adding the compensation values, which are requested from NJ-series Robot Integrated CPU Unit. Refer to *NJ-series Robot Integrated CPU Unit User's Manual (Cat. No. O037)*.

Parameters

Parameter	Description
mode	Optional real value, variable, or expression that defines how path-modification data specified by subsequent ALTER commands are to be interpreted. The mode value is interpreted as a sequence of bit flags that are detailed below. 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 ON, coordinate values specified by subsequent ALTER commands are interpreted as incremental and are accumulated. If this bit OFF, each set of coordinate values is interpreted as the total (non-cumulative) correction to be applied. **Bit 2: World Coordinates (mask value = 2)** If this bit is ON, coordinate values specified by subsequent ALTER commands are interpreted to be in the world coordinate system. If this bit is OFF, coordinates are interpreted to be in the tool coordinate system. **Bit 3: Period Modification from NJ-series Robot Integrated CPU Unit** If this bit is ON, coordinate values are periodically requested from the NJ-series Robot Integrated CPU Unit. This keyword cannot be used at the same time with the ALTER keyword. This condition will return a -621 *(Robot Interlocked)* message. This bit should not be ON while bit 1 is ON. This condition will return a -407 *(Invalid Argument)* message.

Details

This program command initiates the real-time path-modification (alter) functionality. After this command is executed, the coordinate values specified by ALTER commands will automatically be superimposed on the nominal path computed by the V+ trajectory generator during all subsequent robot motions. 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.

RX, RY, and RZ angles represent an extrinsic rotation and are different than the intrinsic rotation yaw, pitch, and roll angles of transformation variables.

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 actions.

- Executing an ALTOFF program command
- Detaching the robot
- Prematurely terminating a robot motion
- Stopping program execution

Example

The following example will initiate alter mode and interpret subsequent ALTER commands as total corrections (bit 1 is OFF) and world-coordinate (bit 2 is ON) corrections to the nominal path of the robot.

```
ALTON 2
```

Related Keywords

3-4-5 ALTER on page 3-273
3-4-6 ALTOFF on page 3-274
3-1-99 STATE on page 3-121

3-4-8 ANY

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

Syntax

ANY

Usage Considerations

The ANY statement must be within a CASE structure.

Details

Refer to *3-4-23 CASE* on page 3-301 for more information.

Example

Refer to *3-4-23 CASE* on page 3-301 for more information.

Related Keywords

3-4-23 CASE on page 3-301
3-4-132 VALUE on page 3-456

3-4-9 APPRO

Start a robot motion toward a location defined relative to specified location with joint-interpolated motion.

Syntax

```
APPRO location, distance
```

Usage Considerations

The APPRO program command can be executed by any program task as long as the task has attached a robot. The command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing these commands will cause an error.

Parameters

Parameter	Description
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-axis) from the specified location. A negative distance offsets the tool forward (positive tool Z-axis).

Details

This command initiates 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 Z-axis of the specified location in the negative direction.

Example

The following example moves the tool by joint-interpolated motion to a location "offset" millimeters from that defined by the transformation "place".

```
APPRO place,offset
```

Related Keywords

3-4-34 DEPART on page 3-316
3-4-35 DEPARTS on page 3-317
3-4-81 MOVE on page 3-384
3-4-83 MOVES on page 3-390

3-4-10 APPROS

Start a robot motion toward a location defined relative to specified location with straight-line motion.

Syntax

```
APPROS location, distance
```

Usage Considerations

APPROS causes a straight-line motion, during which no changes in configuration are permitted.
The APPROS program command can be executed by any program task as long as the task has attached a robot. The command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing these commands will cause an error.

Parameters

Parameter	Description
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 Z-axis) from the specified location. A negative distance offsets the tool forward (positive Z-axis).

Details

This command initiates 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 Z-axis of the specified location in the negative direction.

Example

The following example 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 to a location beyond the location "place".

```
APPROS place,-50
```

Related Keywords

3-4-34 DEPART on page 3-316
3-4-35 DEPARTS on page 3-317
3-4-81 MOVE on page 3-384
3-4-83 MOVES on page 3-390

3-4-11 ATTACH

Make a device available for use by the application program.

Syntax

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

Usage Considerations

The robot is automatically attached when the EXECUTE monitor command or program command is processed for task 0 except when the DRY.RUN system switch is enabled. All the other logical units are automatically detached when program execution begins.
It is recommended to use the IOSTAT real-valued function with the ATTACH program command to monitor the successful completion of the operation. Refer to the example below for more information.
If the Monitor Window or the 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 keywords.

Parameters

Parameter	Description
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 described below. If bit 3 of the mode parameter is 0, this parameter is optional and defaults to 0 to attach the robot. 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. 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 attaches the device specified by the $device parameter and automatically assigns a logical unit number to this parameter. If all the logical units are in use, the parameter is set to -1 . V+ assigns a value to the lun parameter even if the ATTACH request fails.

Parameter	Description
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. The bits are assumed to be clear if no value is specified. **Bit 1 (mask value = 1, LSB): Queue (0) versus Fail (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 ON. If this bit is OFF and the device is already attached by another control program task, the driver queues this attach request and signals the control program that there is an error in the attachment request. In this case, the operation IOSTAT(lun) returns a negative number. The attachment will complete when the device becomes available. If this bit is ON and the device is already attached by another control program task, the device driver immediately signals that the attach request has failed. **Bit 2 (mask value = 2): Wait (0) versus No-wait (1)** This bit controls whether or not the control program task waits for a response from the device driver. For most applications, this bit should be OFF. If this bit is OFF, program execution waits for the device driver to signal the result of the attach request. If this bit is ON, program execution does not wait for the result of the attach request. The program must then use the function statement IOSTAT(lun) to determine if the attachment has succeeded (see above). 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 (mask value = 4): Specify LUN (0) versus Have LUN Assigned (1)** This bit determines how the lun parameter is processed. If this bit is OFF, the device corresponding to the value of lun is attached. The value of the lun parameter specifies the device that is to be attached except when a different device is specified with the $device parameter (refer to the *Details* on page 3-282 section below). If this bit is ON, 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 turned ON by the ATTACH keyword. V+ assigns a value to lun even if the ATTACH request fails. This mode cannot be used to attach the robot or pendant.

Parameter	Description
$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 (the logical unit 0 is always the robot). The acceptable device names are provided below. • **DISK**-Physical drive in the controller (disk or Secure Digital card) • **MONITOR**-The Monitor Window • **SERIAL:n**-Reserved for future use. • **SYSTEM**-Disk device set with the CD or DEFAULT keywords • **TCP**-TCP protocol device driver • **TFTP**-Reserved for future use. • **UDP**-UDP protocol device driver

Details

The robot must remain attached by a robot control program to execute motion of the robot. When the robot is detached, you can use the manual control pendant to move the robot under directions from the application program. This is useful, for application setup sequences for example.
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 by a different task. The robot that is attached by an ATTACH command is the one that was last specified by a SELECT program command executed by the current task. If no SELECT program command has been executed, then robot 1 is attached. The SELECT program command 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 with the following methods.

- Use the STATE function to determine if the system is in COMP mode before executing an ATTACH command.
- Set bit 1 in the mode value and use the IOSTAT function to determine the success of the Attach command.

When the Monitor Window (logical unit 4) is attached, all keyboard input will be buffered for input requests by the program.
When the Monitor Window is attached, a user is not able to type ABORT to terminate program execution. The program must provide a means for fielding a termination request or you must use the pendant or emergency stop switch to stop program execution.
When a DISK device is attached, it allows a program to read and write data from and to files (DISK refers to the Secure Digital card). One of the FOPEN program commands must be used to specify which file to access. WRITE and READ program commands can then be used to transfer information to and from the file. FCMND program commands can be used to send commands to the file system.
When mode bit 3 = 0 and the $device parameter is omitted, the logical unit number implicitly specifies the corresponding default device from the following table.

Number	Device
0	Robot (default when lun is omitted)
1	Reserved for future use
2, 3, 4	Monitor Window
5, 6, 7, 8	Disk
9	No default device
10, 11, 12, 13	Reserved for future use
14, 15, 16	No default device
17, 18, 19	Disk
20, 21, 22, 23, 24	Reserved for future use
25, 26, 27, 28, 29, 30, 31	No default devices

Example

The following example will attempt to attach the robot for an amount of time specified by timer 3 (5 seconds). The IOSTAT real-valued function will return information to monitor the success or failure of the ATTACH operation during this time interval. If the attach operation is not successful within the 5 second interval, a message will be displayed in the Monitor Window.

```
start = TIMER(-3)
DO
    ATTACH (0,1)
    UNTIL (IOSTAT(0, 0) == 1) OR ((TIMER(-3) - start) > 5)
IF IOSTAT(0 , 0) < 1 THEN
    TYPE "The following error occurred: ", $ERROR(IOSTAT(0,0))
    TYPE "The program will be interrupted!"
HALT
END
```

Related Keywords

3-4-36 DETACH on page 3-318
3-4-63 FSET on page 3-361
3-1-58 IOSTAT on page 3-83
3-4-115 SELECT on page 3-431 (program command)

3-4-12 AUTO

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

Syntax

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

Usage Considerations

AUTO (automatic) variables have an undetermined value when a program is first entered, but they are not considered undefined. They have no value after the program exits.
AUTO commands must appear before any executable keyword in the program. Only the .PROGRAM command, comments, blank lines, GLOBAL and LOCAL commands, and other AUTO commands may precede this command.
If a variable is listed in an AUTO command, any global variables with the same name cannot be accessed directly by the program.
The values of AUTO variables are not saved by the STORE or restored by the LOAD commands.

Parameters

Parameter	Description
type	If the type parameter is specified, all the variables must match that type (REAL, DOUBLE, or LOC). Array variables must have their indexes specified explicitly, indicating the highest valid index for the array. 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. LOC: Location variable (transformation or precision point). REAL: Single-precision real variable. DOUBLE: Double-precisionreal variable. Refer to the *3-4-64 GLOBAL* on page 3-362 command for details on the default type.
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, 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 command is used to declare variables to be defined only within the current program. An AUTO variable can be referenced only by the specific calling instance of a program. The names of AUTO variables can be selected without regard for the names of variables defined in any other programs.
AUTO variables are allocated each time the program is called and their values are not preserved between successive subroutine calls. These values can be displayed with monitor commands only when the program task is inactive, but is on an execution stack. When a program is first entered, AUTO variables have arbitrary, undetermined values but they are not considered undefined. AUTO variables are lost when the program exits.
Unlike a LOCAL variable, a separate copy of an AUTO 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 unpredictable program errors. Therefore, AUTO variables should be used for all temporary local variables to minimize the chance of such errors.
Variables can be defined as GLOBAL, AUTO, or LOCAL. An attempt to define AUTO, 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 (AUTO, LOCAL, or GLOBAL). Attempting to define a variable more than once with a different type will result in the error message "Attempt to redefine variable type".
AUTO 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 following statement 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.

```
AUTOLOC points[3,5]
```

The storage space for AUTO variables is allocated on the program execution stack. If the stack is too small for the number of AUTO variables declared, the task execution will stop with the error message "Too many subroutine calls". If this happens, use the STATUS monitor command to determine how much additional stack space is required. Then, use the STACK monitor command to increase the stack size and then issue the RETRY monitor command to continue program execution.
AUTO variables cannot be deleted with the DELETE_ commands.
AUTO variables can be referenced with monitor commands such as DELETE_, DO , HERE , LIST_, TOOL by using the optional context specifier @ as shown in the example below.

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

Example

The following example statement will declare the variables loc.a, $ans, and i to be AUTO 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
```

The following example will declare the variables i, j, and tmp[ ] to be AUTO, real variables in the current program. Array elements tmp[0] through tmp[10] are defined.

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

The following example will declare the variable "loc" to be an AUTO variable in the current program. The variable type of "loc" must be determined by its use in the program. Since "loc" appears by itself, it is not interpreted as the type-specifying keyword.

```
AUTO loc
```

Related Keywords

3-4-64 GLOBAL on page 3-362
3-4-76 LOCAL on page 3-377
3-2-57 STACK on page 3-242
3-4-38 DO on page 3-321
3-4-67 HERE on page 3-366
3-4-128 TOOL on page 3-451

3-4-13 BASE

Translates and rotates the coordinate reference frame relative to the robot's base.

Syntax

```
BASE X_shift, Y_shift, Z_shift, Z_rotation1, Y_rotation1, Z_rotation2
```

Usage Considerations

The BASE program command causes a break in continuous-path motion.
This command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, use of the BASE command will cause an error.

Parameters

Parameter	Description
X_shift	Optional real-valued expression that gives the displacement, in the world coordinate system, of the X component of the origin point for the translated coordinate system. Zero is assumed if no value is provided.
Y_shift	Optional real-valued expression that gives the displacement, in the world coordinate system, of the Y component of the origin point for the translated coordinate system. Zero is assumed if no value is provided.
Z_shift	Optional real-valued expression that gives the displacement, in the world coordinate system, of the Z component of the origin point for the translated coordinate system. Zero is assumed if no value is provided.
Z_rotation1	Optional real-valued expression that gives the displacement, in the world coordinate system, for the first rotation about the Z-axis for the rotated coordinate system. Zero is assumed if no value is provided.
Y_rotation1	Optional real-valued expression that gives the displacement, in the world coordinate system, for the first rotation about the Y-axis for the rotated coordinate system. Zero is assumed if no value is provided.
Z_rotation2	Optional real-valued expression that gives the displacement, in the world coordinate system, of a second rotation about the Z-axis for the rotated coordinate system. A zero value is assumed if no value is provided.

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's 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 command translates and rotates the reference frame relative to this origin.
If the robot is moved from its initial position a BASE command can be used to compensate the robot motions for the displacement.
The BASE command can also be used to realign the X and Y coordinate axes so that SHIFT functions cause displacements in desired, nonstandard directions.
The BASE program command has no effect on locations defined as precision points. The arguments for the BASE program command describe the displacement of the robot relative to its normal location.
The BASE function can be used with the LISTL monitor command to display the current BASE setting in the Monitor Window.

Example

The following example redefines the world reference frame when the robot has been shifted by "xbase" millimeters in the positive X direction, 50.5 millimeters in the negative Z direction, and has been rotated 30 degrees about the Z-axis.

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

The following example redefines the world reference frame to shift all locations 100 millimeters in the negative X direction and 50 millimeters in the positive Z direction from their nominal location. The arguments for this statement describe movement of the robot reference frame relative to the robot and have an opposite effect on locations relative to the robot.

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

Related Keywords

3-1-8 BASE on page 3-14 (transformation function)
3-2-53 SELECT on page 3-237 (monitor command)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-14 BASE.TRANS

Program command that translates and rotates the reference frame relative to the robot base.

Syntax

BASE.TRANS transformation

Usage Considerations

The BASE.TRANS program command applies to the robot selected with the SELECT keyword. This command can be used while other programs are executing. An error will result if the robot is attached by any executing program.
If the V+ system is not configured to control a robot, use of the BASE.TRANS program command causes an error.

Parameter

Parameter	Description
transformation	Optional transformation variable, function, or compound transformation that defines the robot base location. Zero is assumed if no value is provided.

Examples

The following example redefines the world reference frame using the given coordinates and three rotations. The first rotation executed is 20 degrees around the Z axis, the second rotation executed is 180 degrees about the Y axis, and the third executed is -70 degrees around the Z axis. The transformation function (TRANS) returns a value computed from the given X, Y, Z position displacements and y, p, r orientation rotations and sets this as the loc variable. The loc variable is used as the transformation parameter of the BASE.TRANS program command.

```
SET loc = TRANS(250,-300,175,20,180,-70)
BASE.TRANS loc
```

The following example redefines the world reference frame because the robot has been shifted xbase millimeters in the positive X direction, 50.5 millimeters in the negative Z direction, and has been rotated 30 degrees about the Z-axis. The transformation function (TRANS) returns a value computed from the given X, Y, Z position displacements and y, p, r orientation rotations and sets this as the loc variable. The loc variable is used as the transformation parameter of the BASE.TRANS program command.

```
SET loc = TRANS(xbase,, -50.5, 30)
BASE.TRANS loc
```

The following example 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 base, and thus have an opposite effect on locations relative to the robot. The transformation function (TRANS) returns a value computed from the given X, Y, Z position displacements and y, p, r orientation rotations and sets this as the loc variable. The loc variable is used as the transformation parameter of the BASE.TRANS program command.

```
SET loc = TRANS(100,, -50)
BASE.TRANS loc
```

Related Keywords

3-2-2 BASE on page 3-168 (monitor command)
3-1-8 BASE on page 3-14 (transformation function)

3-4-13 BASE on page 3-286 (program command)
3-2-3 BASE.TRANS on page 3-170 (monitor command)
3-2-53 SELECT on page 3-237 (monitor command)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-15 BELOW

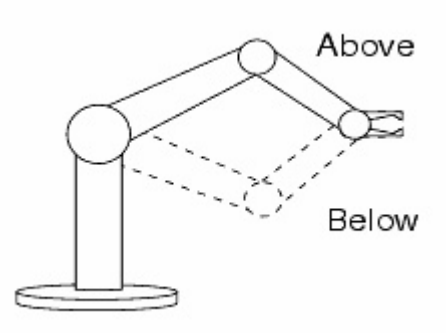
Request a change in the robot configuration during the next motion so that the elbow is below the line from the shoulder to the wrist.

Syntax

BELOW

Usage Considerations

SCARA and parallel robots cannot have a below configuration. If the selected robot does not support a below configuration, this command is ignored by the robot.
Configuration changes cannot be made during straight-line motions.
The BELOW program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. If the robot is not attached, this command has no effect.
This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the BELOW command will cause an error.
The following figure shows the ABOVE and BELOW configurations.



Example

The following example will move the robot to "point1" using the BELOW configuration.

```
BELOW
MOVE point1
BREAK
```

Related Keywords

3-4-2 ABOVE on page 3-269
3-1-18 CONFIG on page 3-24
3-4-115 SELECT on page 3-431 (program command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-16 BITS

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

Syntax

```
BITS first_sig, num_sigs = 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 32 signals can be set at one time.
Any group of up to 32 signals can be set, provided that all the signals in the group are configured for use by the system.
All V+ digital output commands do not wait for a V+ cycle and will turn ON outputs immediately, excluding latch inputs. Digital inputs are checked approximately every 4 ms for Standard Control systems and approximately 8 ms or 16 ms for Robot Integrated systems.

Parameters

Parameter	Description
first_sig	Real-valued expression defining the lowest-numbered signal to be affected.
num_sig	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 32.
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

This program command turns ON or OFF 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 command is to round the value parameter to an integer and then turn ON or OFF a number of signals based on the individual bits of the binary representation of the integer.

Example

The following example sets 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 set to 50% (0011 0010 binary), then signals 1-8 are set as follows after the statement above is issued.

```
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)
```

The following example sets external output signals 5-9 (4 bits) to the binary representation of the BCD digit 7

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

The following example sets external output signals 1-8 (8 bits) to the binary representation of the constant 255, which is 11111111 (binary). Signals 1-8 will all be turned ON.

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

Related Keywords

3-1-12 BITS on page 3-17 (real-valued function)
3-2-32 IO on page 3-211
3-2-50 RESET on page 3-235
3-1-93 SIG on page 3-115
3-1-91 SIG.INS on page 3-113
3-2-54 SIGNAL on page 3-238 (monitor command)
3-4-120 SIGNAL on page 3-437 (program command)

3-4-17 BRAKE

Abort the current robot motion.

Syntax

BRAKE

Usage Considerations

The BRAKE program command can be executed by any program task including a task that is not actively controlling the robot.
This command does not cause a BREAK to occur.
If more than one robot is connected to the controller, this command applies to the robot currently selected (refer to the *3-1-88 SELECT* on page 3-110 keyword).
If the V+ system is not configured to control a robot, the BRAKE command will not generate an error due to the absence of a robot.

Details

The BRAKE program command causes the current robot motion to be aborted immediately. In response to this command, the robot will decelerate to a stop and then begin the next motion without waiting for position errors to null.
Program execution is not suspended until the robot motion stops.

Example

The following example 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 command. Otherwise, the motion is completed normally.

```
MOVES step[1]
DO
    IF SIG(1023) THEN
        BRAKE
        EXIT
    END
UNTIL STATE(2) == 2
MOVES step[2]
```

Related Keywords

3-4-18 BREAK on page 3-292

3-4-18 BREAK

Suspend program execution until the current motion completes.

Syntax

BREAK

Usage Considerations

The BREAK program command is only used to wait for motion by the robot attached to the current task.
If the V+ system is not configured to control a robot, executing the BREAK command will cause an error.

Details

This command has two effects.

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

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

BREAK cannot be used to make one task wait until a motion is completed by another task.
The BREAK command causes continuous-path processing to terminate by blocking V+ program execution until the motion ends. CPOFF program command causes the trajectory generator to terminate continuous path without affecting the forward processing of the V+ program.

Example

The following example will move the robot to "point2" after the move to "point1" is completed without blending or forward processing.

```
MOVE point1 BREAK
MOVE point2
```

Related Keywords

3-4-17 BRAKE on page 3-291
3-4-28 CPOFF on page 3-308
3-4-29 CPON on page 3-309
3-6-2 CP on page 3-475
3-4-115 SELECT on page 3-431 (program command)
3-2-53 SELECT on page 3-237 (monitor command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-19 BY

Completes the syntax of the SCALE and SHIFT functions.

Syntax

```
SCALE(transformation BY value)
SHIFT(transformation BY value, value, value)
```

Example

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

Related Keywords

3-1-87 SCALE on page 3-109
3-1-90 SHIFT on page 3-112

3-4-20 CALIBRATE

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

Syntax

```
CALIBRATE mode, status, robot
```

Usage Considerations

Normally,the keyword is issued with mode equal to zero.
The command has no effect on a robot when the DRY.RUN switch is enabled.
If the robot is to be moved, the CALIBRATE program command or monitor command keywords must be processed every time system power is turned ON and the system is booted from disk. If a robot has the option to execute CALIBRATE at boot enabled, then it will perform the CALIBRATION operation upon boot up. If this option is not enabled (or not supported for that robot), then this keyword must be executed after high power is enabled.
Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information about enabling the option to execute CALIBRATE at boot.
Robots cannot be moved with the manual control pendant or under program control if the robot is not calibrated until the CALIBRATE program command or monitor command keyword has been processed.
The CALIBRATE keyword may operate differently for each type of robot. For robots with non-absolute (e.g., incremental) encoders, this keyword causes the robot to move. In this case, 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. Refer to the *3-2-5 CALIBRATE* on page 3-172 monitor command keyword for details of the robot motion.
If the V+ system is not configured to control a robot, executing the CALIBRATE keyword causes an error.

Parameters

Parameter	Description	
mode	Optional real expression that indicates what part of calibration is to be performed: Value of mode Interpretation	
	0 (or omitted)	Perform a normal calibration. In detail, the following operations are performed: (a) Load the main calibration program if it is not already in memory. (b) Execute the main calibration program with the load, execute, and delete flags set. This causes the robot-specific calibration routines to be loaded, the robots to be calibrated, and the robot routines to be deleted. The main calibration program is left in memory.
	1	Load the main calibration program if it is not already in memory, and execute the main calibration program with the load flag set. This causes the calibration program to load the applicable robot-specific calibration routines. Note that the calibration process is not performed.
	2	Execute the main calibration program (which must already be in memory) with the execute flag set. That causes the system robot(s) to be calibrated, and all the calibration programs to be left in memory.
	3	Execute the main calibration program which must already be in memory with the delete flag set. This causes the calibration program to delete the robot-specific calibration routines from memory. Note that the actual calibration process is not performed and the main calibration program is left in memory.
status	Optional real-valued variable that receives the exit status returned by the calibration program, or (in mode -1) from V+ when trying to enter into the calibrate mode.	
robot	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, all robots are affected. Otherwise,only the setting for the specified robot is affected.	

Details

When the calibration operation is started, V+ assumes that the robot is not calibrated and restricts your ability to move the robot with the pendant or an application program.
The COMP mode light on the pendant does not come on when the robot is not calibrated.
Robots with incremental encoders lose start-up calibration whenever system power is switched OFF. As a safety measure, these 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*.
For the eCobra 600 and 800 robots, this operation causes a small motion of joint 4 (theta).
If this program command attempts to load the main calibration program, the same program, module, and file name, and search algorithm, are employed as for the CALIBRATE monitor command.
If you wish to execute a CALIBRATE operation in task 0, it can be accomplished by using the /C qualifier on the EXECUTE program command keyword in a program. With that qualifier specified, a program to calibrate the robot can run in task 0 even when the DRY.RUN system switch is disabled. A program running in any task other than 0 can execute the CALIBRATE operation without special conditions.
The CALIBRATE operation shall only be executed from one task at a time. If it might be called from multiple tasks (from a REACTE program for example) the call needs to be protected by a mutex using the TAS keyword (see example below).

Example

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

```
DETACH()
DISABLE DRY.RUN
ENABLE POWER

CALIBRATE
ATTACH()
```

The following example must be used if CALIBRATE is called from more than one task.

```
GLOBAL en_po_lock
WHILE TAS(en_po_lock,TRUE) DO
    WAIT
END


ENABLE POWER
CALIBRATE
ATTACH()

en_po_lock = FALSE
```

Related Keywords

3-2-5 CALIBRATE on page 3-172 (monitor command)
3-5-4 NOT.CALIBRATED on page 3-471
3-2-53 SELECT on page 3-237 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-1-104 TAS on page 3-131

3-4-21 CALL

Suspend execution of the current program and continue execution with a new subroutine program.

Syntax

```
CALL program(arg_list)
```

Usage Considerations

External variable types cannot be used for the arg_list parameter.

Parameters

Parameter	Description
program	Name of the new program to be executed.

Parameter	Description
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 as described below. Each argument can be any one of the data types supported by V+ (belt, precision point, real-value, string, and transformation) and can be specified as a constant, a variable, or an expression. 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. If a value is being passed back to the calling program, the parameter must be specified as a variable. External variable types cannot be used. Any argument can be omitted but 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.*1

*1. Refer to *3-4-101 .PROGRAM* on page 3-411 for information on the effect of omitting an argument.

Details

The CALL command 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 command is executed in the subroutine. Execution continues with the keyword immediately following the CALL command.
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 program. 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 program. If the called subroutine changes the value of the variable, the value is also changed for the calling program. 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 automatically changes the other corresponding subroutine argument.
Note that an argument that is 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 shown in the following examples.
For a real variable, passage by value can be forced by enclosing the variable in parentheses as shown below.

```
CALLprog_a((count))
```

For a string variable, an empty string ("") can be added to the variable as shown below.

```
CALLprog_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 as shown below.

```
CALLprog_c(start:NULL)
```

As stated above, the items in the arg_list must match their corresponding items in the called subroutine. In addition to straight forward 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 exa
mple(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[,,])
```

Example

The following example executes the program named "pallet", passing to it a pointer to the variable "count". When a RETURN command is executed in the "pallet" subroutine, control returns to the program containing the CALL command and the "count" variable will contain the current value of the corresponding subroutine argument.

```
CALL pallet(count)
```

The following example executes the program named "cycle". The value 1 is passed to the first parameter of the program "cycle", its second parameter is undefined, and its third parameter receives the value of the expression n+3. If the "cycle" program has more than three parameters, the remaining parameters are all undefined.

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

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

Related Keywords

3-4-22 CALLS on page 3-300
3-4-111 RETURN on page 3-427

3-4-22 CALLS

Suspend execution of the current program and continue execution with a new subroutine program specified with a string value.

Syntax

```
CALLS string(arg_list)
```

Usage Considerations

CALLS takes much longer to execute than the normal CALL command. CALLS should be used only when necessary.
Since the argument list is not specified as part of the string parameter, all the subroutines called by a specific CALLS command must have equivalent argument lists.

Parameters

Parameter	Description
string	String value, variable, or expression defining the name of the new program to be executed. The letters in the name can be lowercase or uppercase and must be within 1 to 15 characters in length.
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. Refer to the *3-4-21 CALL* on page 3-297 command for further information on subroutine arguments.

Details

The CALLS command functions almost exactly the same as the CALL command. The only difference between the two is the way the subroutine name is specified. CALL requires that the name be explicitly entered in the command 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 command, execution automatically returns to the current pro- gram when a RETURN command is executed in the subroutine. Execution continues with the keyword immediately following the CALLS command.

Example

The example below demonstrates how the CALLS command can be used to execute a subroutine whose name is determined when the program is executed.

The program reads a set of four digital inputs (1001 to 1004) to determine which of sixteen different part types applies and is returned to the "type" variable. The part type is considered to be a hexadecimal number which is converted to the corresponding ASCII character as the "$type" variable. Once the character is defined, the appropriate subroutine ("part.0", "part.1", ..., "part.f") is called. 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)
$type = $ENCODE(/H0, type)
CALLS "part."+$type(arguments[type,], status)
```

This example can be made more robust by using the STATUS real-valued function to make sure the proposed subroutine exists before it is called. Using this function avoids possible errors from undefined program names.

Related Keywords

3-4-21 CALL on page 3-297
3-4-22 CALLS on page 3-300
3-4-101 .PROGRAM on page 3-411
3-4-111 RETURN on page 3-427
3-2-58 STATUS on page 3-244

3-4-23 CASE

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

Syntax

```
CASE value OF
```

Usage Considerations

This command must be part of a complete CASE structure.

Parameters

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

Details

The CASE command is a flexible structure available in V+. It provides a means for executing one group of keywords from among any number of groups. The complete syntax is as follows (the blank lines are not required).

```
CASE evaluation OF
    VALUE evaluation_condition_1,...:
        group_of_steps
    VALUE evaluation_condition_2,...:
        group_of_steps
...
    ANY
        group_of_steps
END
```

The three vertical dots indicate that any number of VALUE steps can be used to execute additional groups of keywords.
The ANY step listed above is 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.
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 one is found to meet the evaluation conditions.
3. Keywords following that VALUE step are executed.
4. Execution continues at the first keyword after the END step.

If no VALUE step is found that meets the evaluation conditions and there is an ANY step in the structure, then the keywords following the ANY step will be executed.
If no VALUE evaluation condition is met in the structure, and there is no ANY step, none of the commands in the entire CASE structure are executed.

Example

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

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

The following sample program prompts to enter a test value. If the value is negative, the program displays a message and then exits. Otherwise, a CASE command is used to classify the input value as a member of one of the following three groups.

1. Even integers from 0 to 10.
2. Odd integers from 0 to 10.
3. All other numbers.

```
PROMPT "Enter a value from 0 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 use of the CASE command 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
```

The following example demonstrates how to use the CASE command to evaluate string data.

```
PROMPT "Enter a number", x
CASE TRUE OF
        VALUE ($string1=="hello"):
                TYPE "string=hello"
        VALUE($string1=="world"):
                TYPE "string=world"
        ANY
                TYPE "unknown string"
END
```

The following example demonstrates how to use the CASE command to evaluate string data.

```
CASE TRUE OF
        VALUE ($string1=="hello"):
                TYPE "string=hello"
        VALUE($string1=="world"):
                TYPE "string=world"
        ANY
                TYPE "unknown string"
END
```

Related Keywords

3-4-8 ANY on page 3-277
3-4-44 END on page 3-328
3-1-115 TRUE on page 3-149
3-4-132 VALUE on page 3-456

3-4-24 CLEAR.EVENT

Clear an event associated with the specified task.

Syntax

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

Parameters

Parameter	Description
task	Optional 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. If this parameter is omitted, the number of the current task is used.
flag	Not used. Defaults to 1.

Details

This command clears the event associated with the specified task.
The default event cleared is the input / output completion event for which the statement WAIT.EVENT 1 waits. This event is cleared by the execution (not the completion) of any input / output keyword operation.

Related Keywords

3-1-47 GET.EVENT on page 3-67
3-4-116 SET.EVENT on page 3-432
3-4-137 WAIT.EVENT on page 3-461

3-4-25 CLEAR.LATCHES

Empties the latch buffer for the selected device.

Syntax

```
CLEAR.LATCHES (select)
```

Parameters

Parameter	Description
select	Integer, expression, or real variable that determines whether any latches have occurred since the last time the function was executed. The setting values are described below.
0	Clears latch buffer for currently selected robot
-n (< 0)	Clears latch buffer for belt n

Details

This program command clears the event and all information associated with the specified latch buffer. As opposed to the LATCHED Real-valued function, no latch event data will be made available for retrieval.

Example

The following example is typically used in an initialization program to ensure latches that may have been detected when V+ programs were not executing are cleared from the buffer (first in, first out), so that any new latches are accurate. For the example below, if you stop your V+ programs and belt 1 continues to generate latches, these latches may not be associated with objects still upstream of the robot. It is good practice to clear latch buffers when programs are starting up, except in applications where these latches may still be valid.

```
CLEAR.LATCHES(-1)
```

Related Keywords

3-1-28 DEVICE on page 3-39
3-1-61 LATCHED on page 3-88
3-1-60 LATCH on page 3-87
3-1-79 #PLATCH on page 3-102

3-4-26 COARSE

Enable a low-precision nulling tolerance for the robot.

Syntax

```
COARSE tolerance ALWAYS
```

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 program command executed during the current execution cycle, regardless of whether ALWAYS is specified.
The statement FINE 100 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins. This is the default state of the V+ system.
The COARSE command can be executed by any program task as long as the robot selected by the task is not attached by any other task. The command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the COARSE command causes an error.

Parameters

Parameter	Description
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 cur- rent execution task. Refer to the *Details* on page 3-306 section below for default values.
ALWAYS	Optional qualifier that establishes COARSE as the default condition. COARSE will remain in effect until disabled by a FINE program command. If ALWAYS is not specified, the COARSE command will apply only to the next robot motion.

Details

The COARSE program command enables a low-precision feature in the robot motion servos system 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.
Lower precision is sometimes required in belt tracking applications when the constant motion of the robot prevents the servos from settling to high precision.
If the tolerance parameter is specified, the new setting takes effect at the start of the next motion. The value becomes the default for any subsequent COARSE command 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.

Example

The following example enables the low-precision feature for the next motion operation only.

```
COARSE
```

The following example enables the low-precision feature for the next motion operation with the tolerance settings changed to 150% of the standard tolerance for each joint.

```
COARSE 150
```

The following example enables the low-precision feature until it is explicitly disabled.

```
COARSE ALWAYS
```

Related Keywords

3-4-21 CALL on page 3-297
3-4-22 CALLS on page 3-300
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-47 PRIME on page 3-232
3-2-56 SSTEP on page 3-241

3-2-70 XSTEP on page 3-261

3-4-27 COPY.ARRAY

Copy elements of the source array to the destination array.

Syntax

```
COPY.ARRAY dest_array[dest_start_index] = source_array[source_start_index], copy_length
```

Usage Considerations

The COPY.ARRAY Program Command will raise an error if the size of source array or destination array is lower than the length specified.

Parameters

Parameter	Description
dest_array	Output array where the values are copied to. Global, auto, local or external variables are supported.
dest_start_index	Real value, variable or expression (interpreted as an integer) that identifies the starting element of the output array.
source_array	Input array where the values are copied from. Global, auto, local or external variables are supported.
source_start_index	Real value, variable, or expression (interpreted as an integer) than identifies the starting element of the input array.
copy_length	Real value, variable, or expression (interpreted as an integer) that provides the number of elements to copy.

Example

The following example copies elements from the "source.array" variable to the "dest.array" variable.

```
; Initialize and set values of the source array source.array[0] = 1
source.array[1] = 2
source.array[2] = 3
source.array[3] = 4

; Initialize and set values of the destination array dest.array[0] = 0
dest.array[1] = 0
dest.array[2] = 0
dest.array[3] = 0

; Get the length of the array length = LAST(source.array[])
```

```
; Copy the source array data in the destination array COPY.ARRAY dest.array[0] = so
urce.array[0], length
```

3-4-28 CPOFF

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

Syntax

```
CPOFF ALWAYS
```

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.
If the CP system switch is disabled, continuous-path processing never occurs regardless of any CPOFF commands that are executed.
CPON ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.
The CPOFF program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. The command applies only to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the CPOFF command causes an error.

Parameters

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

Details

When CPOFF is active, the robot will come to a stop at the completion of the next robot motion and any final position errors will be nulled (if required).
Unlike the BREAK program command which is executed after a motion to cause continuous-path processing to terminate, CPON and CPOFF are executed before a motion operation to affect the continuous-path processing of the next motion command. While BREAK applies to only one motion operation, CPOFF can apply to all the motion operations that follow.
The BREAK program command causes continuous-path processing to terminate by blocking V+ program execution until the motion ends. CPOFF causes the trajectory generator to terminate continuous path without affecting the forward processing of the V+ program.

Example

The following example will cause the robot to move to the position defined as "pos1", decelerate its motion, and stop briefly before moving to the position defined as "pos2" (the flow of the program execution will not be stopped). The robot motions between pos1 and pos2 are not blended.

```
CPOFF ALWAYS
MOVES pos1
MOVES pos2
```

Related Keywords

3-4-18 BREAK on page 3-292
3-6-2 CP on page 3-475
3-4-29 CPON on page 3-309
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-29 CPON

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

Syntax

```
CPON ALWAYS
```

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.
If the CP system switch is disabled, continuous-path processing never occurs regardless of any CPON commands that are executed.
CPON ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins. This is the default state of the V+ system.
The CPON program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. The command applies only to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the CPON command causes an error.

Parameters

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

Details

When CPON is active, it is possible to execute a series of motion operations that are blended into a single continuous path. Each motion will be performed in succession without stopping the robot at specified locations.
Unlike the BREAK program command which is executed after a motion operation to cause continuous-path processing to terminate, CPON and CPOFF are executed before a motion operation to affect the continuous-path processing of the next motion operation.
The BREAK program command causes continuous-path processing to terminate by blocking V+ program execution until the motion ends. CPOFF causes the trajectory generator to terminate continuous path without affecting the forward processing of the V+ program.
Executing the CPON command will permit continuous-path processing to occur, but any of the following conditions will break a continuous path and override CPON functionality.

- No subsequent motion operation is executed before completion of the next motion operation.
- The CP system switch is disabled.
- The next motion operation is followed by an operation that explicitly or implicitly causes motion termination (for example, BREAK).

Example

The following example will cause the robot to begin blending moves between positions defined as "pos1" and "pos2" when it was previously commanded to not blend using other methods.

```
CPON ALWAYS
MOVES pos1
MOVES pos2
```

Related Keywords

3-4-18 BREAK on page 3-292
3-6-2 CP on page 3-475
3-4-28 CPOFF on page 3-308
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-30 CYCLE.END

Terminate the executing program in the specified task the next time it executes a STOP program command (or its equivalent).
Suspend processing of an executable program until a program running in the specified task completes execution.

Syntax

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

Usage Considerations

The CYCLE.END program command has no effect if the specified program task is not active.
The CYCLE.END program command suspends execution of the program containing the keyword until the specified program task completes execution. If a program is aborted while its execution is suspended by a CYCLE.END command, the program task specified by the CYCLE.END command will still be terminated if the stop_flag is TRUE.

Parameters

Parameter	Description
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 pro- gram command 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 CYCLE.END has all its other functionality as described 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 command (or its equivalent), regardless of how many program cycles are left to be executed.
CYCLE.END will not terminate a program with continuous internal loops. Such a program must be terminated with the ABORT keyword.
Regardless of the stop_flag parameter, this command will wait until the program is terminated. If the program being terminated loops internally so that the current execution cycle never ends, the CYCLE.END command will wait indefinitely.
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 issuing an ABORT monitor command for the program task that executed the CYCLE.END command.

Example

The following example demonstrates how a program task can be initiated from another program task. The ABORT and CYCLE.END program commands are used to ensure the specified program task is not already active.

```
ABORT 3
CYCLE.END 3
EXECUTE 3 new.program
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-9 CYCLE.END on page 3-177 (monitor command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)
3-2-58 STATUS on page 3-244 (monitor command)
3-1-100 STATUS on page 3-127 (real-valued function)
3-4-125 STOP on page 3-448

3-4-31 DECOMPOSE

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

Syntax

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

Parameters

Parameter	Description
array_name	Name of the real-valued array that has 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 is incremented as the values are assigned.
location	Location value that is 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 program command 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, 6 elements are defined corresponding to X, Y, Z, yaw, pitch, and roll.
If the location is represented as a precision-point value, then 1 to 12 elements are defined depending on the number of robot joints present that correspond to the individual joint positions.

Example

The following example assigns the components of transformation part to elements 0 to 5 of array "x".

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

The following example assigns the components of precision point "#pick" to array element "angles[4]" and those that follow.

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

The following example will decompose a multi-dimension variable array named "point".

```
SET point[0] = TRANS(1,10,2,10,3,10)
SET point[1] = TRANS(1,20,2,20,3,20)
SET point[2] = TRANS(1,30,2,30,3,30)
FOR i = 0 TO 2
    DECOMPOSE a[i,] = point[i]
END

FOR i = 0 TO 2
    FOR j = 0 TO 5
        TYPE "a[", i, ",", j, "]", a[i,j]
    END
END
```

Related Keywords

3-1-81 #PPOINT on page 3-104
3-1-112 TRANS on page 3-144

3-4-32 DEF.DIO

Assign virtual digital I/O to standard V+ signal numbers for use by keywords.

Syntax

```
DEF.DIO signal = count, operation
```

Usage Considerations

This program does not support Host I/O signal numbers 4001 to 4999. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Parameters

Parameter	Description
signal	Value representing a V+ signal number from 17 to 505 for output or from 1017 to 1505 for input.
count	The number of signals desired. Default is 8 and the range is 1 to 32.
operation	An optional integer or real-valued expression representing the action to be taken. If omitted or positive, the signals will be configured. If negative, the signals will be removed.

Details

Each DEF.DIO operation can only map 32 inputs or 32 outputs. However multiple operations can be made to map more signals. The signals mapped this way are equivalent to soft signals and can be read or set with the SIG function or the SIGNAL and BITS program commands.
If the signal is already mapped to a real or virtual digital I/O, an error -405 * Illegal digital signal * is returned.

Example

The following example defines input signals 1033 through 1048 as virtual signals.

```
DEF.DIO 1033 = 16
```

Related Keywords

3-1-12 BITS on page 3-17
3-2-32 IO on page 3-211
3-2-50 RESET on page 3-235
3-4-114 RUNSIG on page 3-429
3-1-93 SIG on page 3-115
3-1-91 SIG.INS on page 3-113
3-4-120 SIGNAL on page 3-437

3-4-33 DEFBELT

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

Syntax

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

Usage Considerations

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

When a belt variable is initialized using this command, its window parameters are set to allow any location in the working volume of the robot and no belt window is set. Refer to the *3-4-139 WINDOW* on page 3-464 program command for more information.

Parameters

Parameter	Description
%belt_var	Name of the belt variable to be defined.
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. Encoders numbered from 101 to 116 can be specified. 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 is 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 for the DEFBELT program command, a belt variable contains the following information

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

Belt variables have the following characteristics.

- Belt variable names must always be preceded by the percent character (%). 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 command. There is no assignment command for them. The following statements are invalid.

```
%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 command can be stored on a mass-storage device.

Example

The following example defines the belt variable "%belt.var". The value of "b.num" must be the number of the encoder to be associated with this belt variable. 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

3-1-11 BELT on page 3-16 (real-valued function)
3-5-1 BELT.MODE on page 3-468
3-1-14 BSTATUS on page 3-20
3-4-118 SETBELT on page 3-434
3-4-139 WINDOW on page 3-464 (program command)
3-1-127 WINDOW on page 3-164 (real-valued function)

3-4-34 DEPART

Start a robot motion away from the current location with joint-interpolated motion.

Syntax

DEPART distance

Usage Considerations

This program command can be executed by any program task as long as the task has attached a robot. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing this command causes an error.

Parameters

Parameter	Description
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

This program command initiates a joint-interpolated robot motion to a new location which is offset from the current location by the distance given, measured along the current tool Z-axis.

Example

The following example moves the robot tool 80 millimeters back from its current location using a joint-interpolated motion.

```
DEPART 80
```

Related Keywords

3-4-9 APPRO on page 3-277
3-4-10 APPROS on page 3-278
3-4-35 DEPARTS on page 3-317
3-4-81 MOVE on page 3-384
3-4-83 MOVES on page 3-390
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-35 DEPARTS

Start a robot motion away from the current location with straight-line motion.

Syntax

```
DEPARTS distance
```

Usage Considerations

The DEPARTS program command causes a straight-line motion during which no changes in configuration are permitted.
This command can be executed by any program task as long as the task has attached a robot. This command applies to the robot selected by the task.

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

Parameters

Parameter	Description
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

This command initiates a straight-line robot motion to a new location which is offset from the current location by the distance given, measured along the cur- rent tool Z-axis.

Example

The following example withdraws the robot tool (2 * offset) millimeters along a straight-line path from its current location.

```
DEPARTS 2*offset
```

Related Keywords

3-4-9 APPRO on page 3-277
3-4-10 APPROS on page 3-278
3-4-34 DEPART on page 3-316
3-4-81 MOVE on page 3-384
3-4-83 MOVES on page 3-390
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-36 DETACH

Release a specified device from the control of the application program.

Syntax

```
DETACH (logical_unit)
```

Usage Considerations

Detaching the robot causes a break in continuous-path motion.
DETACH automatically forces an FCLOSE operation if a disk file is open on the specified device.

The robot is automatically attached to task 0 when the EXECUTE keyword is processed to initiate that task and the DRY.RUN system switch is disabled. All other logical units are automatically detached when program execution begins. Other events that cause automatic detachment are listed below.

Parameters

Parameter	Description
logical_unit	Optional real value, variable, or expression interpreted as an integer that identifies the device to be detached. Refer to the *3-4-11 ATTACH* on page 3-279 program command for a description of logical unit numbers. The parentheses can be omitted if the logical unit number is omitted. This will cause the robot to be detached.

Details

This program command releases the specified device from control by the application program. No error is generated if the device was not previously attached.
Control of the specified device can be returned to the program with the ATTACH program command. When logical_unit is 0 or omitted, the program releases control of the robot. While the robot is detached, robot power can be turned OFF and ON, the 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 occurs when a robot is detached.
This is useful for applications that require defining where the robot should be located for certain operations. A teaching program can DETACH the robot and then output directions in the Monitor Window or the pendant. You can then use the pendant to move the robot to the desired locations. The Monitor Window or the pendant can be used for accepting input.
When a disk logical unit is detached, any device that was specified by the corresponding ATTACH program command is forgotten and subsequent ATTACH command must 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 keyword
- Processing of the KILL keyword
- Processing of the ZERO monitor command
- Normal completion of program execution

If a program terminates execution abnormally, all of its devices remain attached, but the teach pendant is detached. Abnormal termination of program execution refers to any cause other than HALT or STOP program commands. If the task is subsequently resumed, the program automatically reattaches the Monitor Window and pendant if they were attached before the termination.

Additional Information

It is possible that another program task attached the pendant in the meantime. This results in an error message when the stopped task is restarted.

Example

The following example will release program control of the robot.

```
DETACH
```

The following example will discontinue program control of the pendant.

```
DETACH (1)
```

Related Keywords

3-4-11 ATTACH on page 3-279

3-4-37 DISABLE

Turn OFF one or more system switches.

Syntax

```
DISABLE switch, ... , switch
```

Usage Considerations

If a specified system 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.
Disabling the DRY.RUN system switch does not have an effect until the next EXECUTE keyword is processed for task 0, an ATTACH program command is executed for the robot, or a CALIBRATE keyword is processed.

Precautions for Correct Use

The system switches are shared by all the program tasks. 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.

Parameters

Parameter	Description
switch	Name of a system switch to be turned OFF. The name can be abbreviated to the minimum length that uniquely identifies the switch. For example, the MESSAGES system switch can be referred to as "ME" since there is no other switch with a name beginning with the letters ME.

Details

When a system switch is turned OFF, the feature it controls is no longer functional or available for use. Turning a switch ON with the ENABLE keyword makes the associated feature functional or available for use.
The SWITCH monitor command or the SWITCH real-valued function can be used to determine the status of a system switch at any time. The SWITCH program command can be used like the DISABLE program command is used to disable a switch.

Example

The following example turns OFF the MESSAGES system switch.

```
DISABLE MESSAGES
```

Related Keywords

3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-4-38 DO

Introduce a DO program structure.

Syntax

DO

Usage Considerations

The DO program structure must be concluded with an UNTIL command.

Details

The DO structure provides a way to control the execution of a group of keywords 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 command steps is executed.

2. The logical expression is evaluated. If the result is FALSE, return to item. Otherwise, proceed to item 3.
3. Program execution continues at the first command after the UNTIL command.

When this structure is used, it is assumed that some action occurs within the group of enclosed keywords that changes the result of the logical expression from FALSE to TRUE when the structure should be exited. Alternately, a logical_expression can be replaced with an expression that evaluates the state of a digital I/O signal (see example below).
The group of keywords within the DO structure is always executed at least one time. Refer to the *3-4-138 WHILE* on page 3-463 command for a variation of this functionality.
There do not need to be any keywords between the DO and UNTIL commands. When there are no such keywords, the UNTIL criterion is continuously evaluated until it is satisfied, at which time program execution continues with the keywords following the UNTIL command.

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. Then, the robot typically performs a different sequence of motions.

```
DO
    CALL get.part()
    CALL put.part()
UNTIL SIG(buffer.full)
```

Related Keywords

3-4-38 DO on page 3-321
3-4-47 EXIT on page 3-334
3-4-84 NEXT on page 3-391
3-4-131 UNTIL on page 3-455
3-4-138 WHILE on page 3-463

3-4-39 DOS

Execute a keyword defined by a string expression.

Syntax

```
DOS string, error
```

Usage Considerations

Before the command is executed, the string must be translated from ASCII into the internal representation used by V+. This causes the DOS command to execute slower than issuing the keyword directly.
The string cannot define a declaration statement or most of the control structure statements.

The DOS command is ignored if the string defines a comment line or a blank line.
If a variable referenced in the command is not found in the current program context, the variable is assumed to be global. Any new variables that are created by the command (for example, in an assignment statement) are created as global types. 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 multi-line control structures (for example, CASE ... END, IF ... ELSE ... END) cannot be executed by the DOS command.

Parameters

Parameter	Description
string	String constant, variable, or expression that defines the keyword to be executed. The command 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 command. The value is set to 1 if the command succeeds. If the command 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) command provides a method for altering a program during run-time. The embedded keyword that is defined by a string expression is executed as though it had been entered in the program as a normal keyword.
The keyword executes in the context of the current program. Any subroutine argument, automatic variable, or local variable can be accessed.

Example

The following example 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 "var = 123"
```

The following example causes the keyword contained in the string variable $ins to be executed. If an error occurs, an V+ error code is placed in the real variable status and execution continues

```
DOS $ins, status
```

Related Keywords

3-4-38 DO on page 3-321
3-4-79 MCS on page 3-381

3-4-40 DRIVE

Move an individual joint of the robot.

Syntax

```
DRIVE joint, change, speed
```

Usage Considerations

The DRIVE program command can be executed by any program task as long as the task has attached a robot. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the DRIVE command causes an error.

Parameters

Parameter	Description
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. 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 program speed setting. This can be specified by a constant, a variable, or an expression.

Details

The DRIVE program command operates the single specified robot joint, changing its position by the amount specified with the change parameter in degrees or millimeters. The joint parameter 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 command and the monitor speed setting. The regular program speed setting is not used. Refer to the *3-2-55 SPEED* on page 3-240 monitor command and the *3-4-124 SPEED* on page 3-445 program command for information about motion speeds.
The duration setting established by the DURATION program command also affects the execution time of the motion.

Example

The following example changes 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 Keywords

3-1-88 SELECT on page 3-110 (real-valued function)
3-4-115 SELECT on page 3-431 (program command)

3-4-41 DURATION

Set the minimum execution time for subsequent robot motions.

Syntax

```
DURATION time ALWAYS
```

Usage Considerations

Unless the ALWAYS parameter is specified, only the next robot motion is affected.
The statement DURATION 0 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.
The DURATION program command affects the DRIVE program command.
The setting of the SPEED monitor command affects the results of the DURATION setting.
The DURATION command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the DURATION command causes an error.

Parameters

Parameter	Description
time	Real-valued expression that specifies the minimum length of time in seconds that subsequent robot motions take to perform. 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 keywords.
ALWAYS	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 32.

Details

This command sets the minimum execution time for subsequent robot motions. For any motion, the time specified by the DURATION command has no effect if the duration setting is less than the time computed by the V+ robot-motion trajectory generator using the current motion speed and acceleration settings. 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.
The DURATION command 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. Continuous-path motions in which individual motions are blended together get the correct duration, but a single motion takes longer than the specified duration. The time of motion is primarily defined either by the value of DURATION or SPEED, using the value that provides the longer time.
Consider a situation where the value of a periodic, external signal is used to continuously correct the path of the robot while the robot is moving. The DURATION command can be used to match the motion execution time to the sensor sampling rate and processing time. This ensures that the robot is kept in motion while new information is being processed.
Actual motion times may differ slightly from the duration setting due to quantization effects and due to acceleration and deceleration profiling.

Example

The following example evaluates an external sensor and moves to the computed robot location. This sequence is repeated 20 times at intervals of 24 milliseconds (6/TPS seconds). This assumes the default period of 4 milliseconds for the V+ trajectory generator. The motion speed is set to a very large value to ensure motion is paced by the duration setting.

```
DURATION 6/TPS ALWAYS
SPEED 200 ALWAYS
    FOR i = 1 TO 20
        CALL read.signal(loc)
        MOVE loc
END
```

Related Keywords

3-4-3 ACCEL on page 3-270
3-1-30 DURATION on page 3-41 (function keyword
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-55 SPEED on page 3-240 (monitor command)
3-4-124 SPEED on page 3-445 (program command)

3-4-42 ELSE

Separate the alternate group of statements in an IF ... THEN control structure.

Syntax

ELSE

Usage Considerations

ELSE can be used only within an IF ... THEN ... ELSE ... END control structure.

Details

Denotes 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.

Example

The following example determines 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. The outer IF does not include an ELSE clause.

```
IF DEFINED(input.signal) THEN
    IF SIG(input.signal) THEN
        TYPE "The input signal is ON"
    ELSE
        TYPE "The input signal is OFF"
    END
END
```

Related Keywords

3-4-69 IF...THEN on page 3-368

3-4-43 ENABLE

Turn ON one or more system switches.

Syntax

ENABLE switch, ..., switch

Usage Considerations

The ENABLE can be used when a program is executing.
If a specified system 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.

Precautions for Correct Use

The system switches are shared by all the program tasks. 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.

Parameters

Parameter	Description
switch	Name of a system switch to be turned ON. The name can be abbreviated to the minimum length that uniquely identifies the switch. 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

When a system switch is turned ON, the feature it controls is functional and available for use. Turning a switch OFF with the DISABLE keyword makes the associated feature not functional or available for use.

The SWITCH monitor command or the SWITCH real-valued function can be used to determine the status of a system switch at any time. The SWITCH pro- gram command can be used like the ENABLE program command is used to set a switch.

Example

The following example turns ON the MESSAGES system switch.

```
ENABLE MESSAGES
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-4-44 END

Mark the end of a control structure.

Syntax

END

Usage Considerations

Every END program command must be part of a CASE, FOR, IF ... GOTO, IF ...THEN, or WHILE control structure.

Details

Every CASE, FOR, IF ... GOTO, IF ... THEN, or WHILE control structure must have its end marked by this END program command. The V+ editor displays an error message when program editing is exited if the correct number of END commands do not exist in a program.

Example

The following example demonstrates the use of END in a WHILE control structure.

```
WHILE TRUE DO
        ;statements
END
```

The following example demonstrates the use of END in a FOR control structure.

```
FOR i = 0 TO 10
        ;statements
END
```

The following example demonstrates the use of END in an IF ... THEN control structure.

```
IF (condition) THEN
        ;statements
        ELSE
;statements
END
```

The following example demonstrates the use of END in a CASE control structure.

```
CASE condition OF
        VALUE (condition):
                ;statements
        VALUE (condition):
                ;statements
        VALUE (condition):
                ;statements
        ANY
                ;statements
END
```

The following example demonstrates the use of END in an IF ... GOTO control structure.

```
        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)
100 IF IOSTAT(dlun) < 0 THEN
        TYPE $ERROR(IOSTAT(dlun))
END
```

Related Keywords

3-4-23 CASE on page 3-301
3-4-61 FOR on page 3-358
3-4-68 IF...GOTO on page 3-367
3-4-69 IF...THEN on page 3-368
3-4-138 WHILE on page 3-463

3-4-45 ESTOP

Stop the robot in the same manner as if an emergency-stop signal was received.

Syntax

ESTOP robot_num

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

This keyword immediately issues the emergency-stop signal and then proceeds with a normal power-down sequence. It is functionally identical to pressing the E-STOP button on the pendant.
No error will be generated if the specified robot_num is not connected.

Example

The following example initiates an emergency-stop, power down sequence for robot 2.

```
ESTOP 2
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)

3-4-17 BRAKE on page 3-291
3-2-21 ESTOP on page 3-194 (program command)
3-2-44 PANIC on page 3-228 (monitor command)
3-4-91 PANIC on page 3-400 (program command)
3-1-99 STATE on page 3-121

3-4-46 EXECUTE

Begin execution of a control program.

Syntax

```
EXECUTE /C task program (param_list), cycles, step, priority[i]
```

Usage Considerations

A program cannot already be active as the specified program task.

Parameters

Parameter	Description
/C	Optional qualifier that conditionally attaches the selected robot. The qualifier has an effect only when starting the execution of task 0.
task	Real value or expression specifying which program task is to be activated. Refer to *V+ User's Manual (Cat. No. I671)* for more information about program tasks.
program	Name of the program to be executed.
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 command. They must be passed by value (refer to the *3-4-21 CALL* on page 3-297 program command). The parameters are evaluated in the context of the new task that is started.

Parameter	Description
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 (after the initial blank and comment lines and all the AUTO and LOCAL statements).
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. Refer to *V+ User's Manual (Cat. No. I671)* for more information.
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 is executed cycles times, starting at the specified program step.

After a program initiates execution of another program, the initiating program can use the STATUS and ERROR functions to monitor the status of the other program.

The optional /C qualifier has an effect only when starting execution of task 0. When /C is not specified, an EXECUTE command 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 System Switch is enabled. When /C is specified, an EXECUTE command for task 0 attempts to attach the robot, but allows execution of task 0 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 statements.

```
CPON ALWAYS DURATION 0 ALWAYS
FINE 100 ALWAYS
LOCK 0 MULTIPLE ALWAYS NULL ALWAYS OVERLAP ALWAYS
SPEED 100,100 ALWAYS SELECT ROBOT = 1
```

The robot configuration is saved for subsequent motions.

An execution cycle is terminated when a STOP program command is executed, a RETURN program command 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 program command is executed, an error occurs, or execution of the program is aborted.

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. 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.
All the EXECUTE parameters are evaluated in the context of the new task that is started. This can lead to unexpected results when the EXECUTE command is used and an attempt is made to pass a task-dependent value (for example, the TASK 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.

Example

The following example initiates execution (as task 0) of the program named "assembly" with execution to continue until HALT program command is executed, an error occurs, or execution of the program is aborted.

```
EXECUTE 0 assembly, -1
```

The following example initiates 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)
```

The following example demonstrates how an application program can be initiated from another application program. The ABORT and CYCLE.END program commands are used to ensure the specified program task is not already active.

```
ABORT 3
CYCLE.END 3
EXECUTE 3 new.program
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-4-21 CALL on page 3-297 (program command)
3-2-9 CYCLE.END on page 3-177 (monitor command)
3-4-30 CYCLE.END on page 3-310 (program command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)
3-2-47 PRIME on page 3-232
3-2-48 PROCEED on page 3-233
3-2-52 RETRY on page 3-236
3-2-56 SSTEP on page 3-241
3-2-58 STATUS on page 3-244
3-1-100 STATUS on page 3-127 (real-valued function)
3-2-70 XSTEP on page 3-261

3-4-47 EXIT

Branch to the statement following the nth nested loop of a control structure.

Syntax

```
EXIT count
```

Usage Considerations

This command works with the FOR, WHILE, and DO control structures.

Parameters

Parameter	Description
count	Optional integer value (expressions and variables are not acceptable) specifying how many nested structures to exit. The default value is 1.

Details

When an EXIT command is reached, the control structure is terminated and execution continues at the first command following the outermost control structure exited.

Example

In the following example, if input signal 1001 is set, exit one control structure. If 1002 is set, exit three control structures.

```
FOR i = 1 TO 40
    WHILE ctrl.var DO
        DO
            IF SIG(1002) THEN
                EXIT 3
            END
            IF SIG(1001) THEN
                EXIT
            END
        UNTIL FALSE
        count = count+1
    END
END
```

Related Keywords

3-4-38 DO on page 3-321

3-4-61 FOR on page 3-358
3-4-84 NEXT on page 3-391
3-4-138 WHILE on page 3-463

3-4-48 EXTERNAL

Declare a variable that is shared between V+ and a host controller.

Syntax

```
EXTERNAL variable, ..., variable
```

Usage Considerations

The EXTERNAL keyword is only available for use with the Robot Integrated Controller.
EXTERNAL keyword statements must appear before any executable statement in the program.
The external variable must also be declared with the same name in the NJ-series Robot Integrated CPU Unit. Refer to *NJ-series Robot Integrated CPU Unit User's Manual (Cat. No. O037)* for more information.
External variables are automatically declared during the PROFINET or EtherCAT SubDevice configuration process. Data type and size are also automatically established. Do not declare External variables that already exist from a PROFINET or EtherCAT SubDevice configuration procedure using the EXTERNAL keyword. Refer to *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* and *Industrial Robot Fieldbus Configuration User's Guide (Cat. No. M130)* for more information.
External variables can be accessed from V+ programs and the Monitor Window in the same way as global variables. The type of the variable is defined by the host system.
Robot Integrated Control systems support the following data types:

- BOOL (16 bit)
- INT (16 bit)
- UINT (16 bit)
- REAL (32 bit) LREAL (64 bit)
- Fixed Length Array (one dimension) of BOOL, INT, UINT, REAL, LREAL with a maximum number of elements of 100 (range [0...99])

If the value of the external variable is out of range for the data type defined in the NJ-series Robot Integrated CPU Unit, it will have an undefined value in that unit.
External variables cannot be declared with the same name as other variable types (GLOBAL, LOCAL, AUTO) in the same program.
Using any of the following keywords with an external variable is not permitted:

- CALL (when an EXTERNAL variable is used as a subroutine argument)
- FOR (with an EXTERNAL variable counter)
- DELETE, DELETER, and DELETES
- STORE, STORER, and STORES
- ARGLIST.DECODE and ARGLIST.ENCODE
- PROMPT and READ (when an EXTERNAL variable is used as an input argument)
- LOAD and PRIME (when an EXTERNAL variable is used as program arguments)

Parameters

Parameter	Description
variable	Variable name as a double-precision data type. Each variable can be a simple variable or an array. If an array is used, the array size is defined in the NJ-series Robot Integrated CPU Unit declaration and must match the array size of the variable parameter. Array variables must not have their indexes specified, but they must include brackets [] after the variable name. The array size range is 0 to 99 and only one-dimension, fixed length arrays are allowed. Refer to the *Details* on page 3-336 section below for information about variable names.

Details

After declaring the external variable, it will be available for use with any V+ program executed in any V + task.

Once an external variable has been declared, it is available to any executing program until the variable is deleted or all V+ programs that reference it are removed from system memory.

The following conditions will create an error when retrieving the value of an external variable from the NJ-series Robot Integrated CPU Unit.

- The variable does not exist in the NJ-series Robot Integrated CPU Unit (-406, undefined program or variable name).
- The array limits are exceeded (-404, illegal array index)
- The variable type is not supported or the types do not match (-404, illegal array index)

● Variable Names

The name of the external variable must be the same in V+ and the host system.

Variable names must begin with an alphabetic character (a-z) and can include 0-9, a-z, periods, and underscores.

The maximum length of a variable name is 15 bytes.

Example

The following example declares an external variable "myvar", accesses the variable value, and then changes the value for a Robot Integrated Control system.

```
EXTERNAL myvar IF myvar==1 THEN
myvar=myvar+1
END
```

Related Keywords

3-4-12 AUTO on page 3-283
3-4-64 GLOBAL on page 3-362
3-4-76 LOCAL on page 3-377

3-4-49 FCLOSE

Close the disk file currently open on the specified logical unit.

Syntax

```
FCLOSE (logical_unit)
```

Usage Considerations

No error is generated if a file is not open on the logical unit, although the IOSTAT real-valued function returns an error code.

Parameters

Parameter	Description
logical_unit	Real value, variable, or expression (interpreted as an integer) that identifies the device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 command for a description of logical unit numbers.

Details

After a program has finished accessing a file that has been opened with an FOPEN command, the program must close the file by executing an FCLOSE command. 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. If this operation is not performed, the disk file may not actually contain all of the information written to it.
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 currently open on the specified logical unit.

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-4-59 FOPENR on page 3-354
3-4-56 FOPEN on page 3-348
3-1-58 IOSTAT on page 3-83
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)

3-4-50 FCMND

Generate a device-specific command to the input / output device specified by the logical unit.

Syntax

```
FCMND (logical_unit, command_code) $out_string, $in_string
```

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. Refer to the *3-2-10 DEFUALT* on page 3-179 monitor command for more information.

Parameters

Parameter	Description
logical_unit	Real-valuedexpression that identifies the device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 program command for a description of logical unit numbers.
command_code	Real-valued expression that specifies the command to be executed.
$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

The FCMD program command allows a program to generate device-specific command sequences. For example, this command 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 commands exist for performing these operations.
Any error in the specification of this command such as attempting to access an invalid unit, will cause a program error and will halt program execution. 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.

File Command Codes

A file cannot be open on the logical unit when the FCMND command is executed.

Code	Description
6	Rename a file. The $out_string parameter must contain the old file name, including any required disk unit and directory path specification. The $in_string parameter must contain the new file name.
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 *V+ User's Manual (Cat. No. I671)* for a description of subdirectory specifications. Only the final subdirectory in the specified directory path is created by this operation. All the intermediate subdirectories are not created and they must already exist.
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. Refer to *V+ User's Manual (Cat. No. I671)* for a description of subdirectory specifications. Only the final subdirectory in the specified directory path is created by this operation. All the intermediate subdirectories are not created and they must already exist.
19	Assert the creation date / time for the file currently open on the specified logical unit. This file command can be issued at any time a disk file is opened. Once issued, 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. 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 while observing the following considerations. • 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. This file command code applies only to local disk drives.
20	Return the number of unused and total number of kilobytes on a local disk. The returned string is in the form uuuuu/ttttt where uuuuu is the number of unused kilobytes and ttttt is the total number of kilobytes. A file must be open on the drive with prereads disabled. The open file identifies the disk unit.

Code	Description
21	Read the creation date / time for the file currently open on the specified logical unit. This file 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 statement has been issued to assert file creation date and time, the FCMND 21 statement returns the value set by FCMND19 statement. The string returned by this command contains date and time (use the INTB to extract the values). The following considerations should be made. • 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. This file command code applies only to local disk drives.

TCP Command Codes

Code	Description
600	Initiate a close connection from the TCP server side for the client identified by the handle number "handle" in the statement FCMND (lun, 600) $INTB(handle).*1
601	Initiate a PING monitor command. The resulting IOSTAT function value returned is either 1, indicating the client was found on the network, or -562, indicating a network timeout.

*1. Close-connection requests are more commonly initiated by the client side.

Example

The following example checks to see if a client is on the network.

```
FCMND (lun, 601) "node_address", $str
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-2-24 FDELETE on page 3-199 (monitor command)
3-4-52 FDELETE on page 3-342 (program command)
3-2-25 FDIRECTORY on page 3-200
3-4-56 FOPEN on page 3-348
3-2-28 FRENAME on page 3-205
3-1-58 IOSTAT on page 3-83
3-4-79 MCS on page 3-381

3-4-51 FCOPY

Copy the information in an existing disk file to a new disk file.

Syntax

```
FCOPY err, $new_file = $old_file
```

Parameters

Parameter	Description
err	Optional parameter used to return an error.
$new_file	String constant, variable, or expression that specifies the file for the new disk file to be created. If the period (".") and filename extension are omitted, the default is a blank extension. The current default device, unit, and directory path are considered as appropriate. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information.
$old_file	String constant, variable, or expression that specifies an existing disk file. If the period (".") and filename extension are omitted, the default is a blank extension. The current default device, unit, and directory path are considered as appropriate.

Details

If the new file already exists or the old file does not exist, an error is reported and no copying takes place. You cannot overwrite an existing file. The existing file must first be deleted with an FDELETE command.

If the file to be copied has the read-only attribute, the new file will also have that attribute. Files with the protected attribute cannot be copied. Refer to *3-2-25 FDIRECTORY* on page 3-200 for a description of file protection attributes.

When a file is copied, the file creation date and time are preserved along with the standard file attributes. The only attribute that is affected is the archived bit, which is cleared to indicate that the file is not archived.

In general, a file specification consists of the following six elements.

1. An optional physical device (for example, DISK>)
2. An optional disk unit (for example, D:)
3. An optional directory path (for example, DEMO\)
4. A file name (for example, NEWFILE)
5. A period character (".")
6. A file extension (for example, V2)

Example

The following example creates a file named "newfile.v2" on disk device "D" that is an exact copy of the existing file named "oldfile.v2" on disk device "D".

```
FCOPY "D:\newfile.v2" = "D:\oldfile.v2"
```

Related Keywords

3-2-10 DEFUALT on page 3-179
3-2-23 FCOPY on page 3-197 (monitor command)
3-2-28 FRENAME on page 3-205

3-4-52 FDELETE

Delete the specified disk file.

Syntax

FDELETE (lun) object

Usage Considerations

The logical unit number must be attached, but no file can be currently open on that logical unit.
The FCLOSE keyword must be issued before deleting an open disk file with this keyword.

Parameters

Parameter	Description
lun	Real value, variable, or expression interpreted as an integer that corresponds to a disk. Refer to the *3-4-11 ATTACH* on page 3-279 command for a description of logical unit numbers.
object	String constant, variable, or expression specifying the disk file to delete. The error "Nonexistent file" will be reported with IOSTAT if the specified object does not exist. 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. Refer to the *3-2-10 DEFUALT* on page 3-179 command for more information.

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.

Example

The following example closes and then deletes the disk file defined by the file specification in the string variable $file.

```
FCLOSE (5) $file
FDELETE (5) $file
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-49 FCLOSE on page 3-337
3-4-52 FDELETE on page 3-342 (monitor command)
3-4-56 FOPEN on page 3-348
3-1-58 IOSTAT on page 3-83

3-4-53 FEMPTY

Empty any internal buffers in use for a disk file by writing the buffers to the file if necessary.

Syntax

FEMPTY (lun)

Usage Considerations

When accessing a file, the file must be open for random access on the specified logical unit (refer to the *3-4-56 FOPEN* on page 3-348 command for more information).

Parameters

Parameter	Description
lun	Real value, variable, or expression interpreted as an integer that corresponds to a disk. Refer to the *3-4-11 ATTACH* on page 3-279 command 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 command, 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 command 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.
The IOSTAT real-valued function can be used to determine if any error results from an FEMPTY operation.

Example

The following example empties the internal output buffer for logical unit 5 and write to the disk immediately.

```
FEMPTY (5)
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-56 FOPEN on page 3-348
3-4-57 FOPENA on page 3-349
3-4-58 FOPEND on page 3-352
3-4-59 FOPENR on page 3-354
3-4-60 FOPENW on page 3-356
3-1-58 IOSTAT on page 3-83
3-4-140 WRITE on page 3-466

3-4-54 FINE

Enable a high-precision nulling tolerance for the robot.

Syntax

```
FINE tolerance ALWAYS
```

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 sub- sequent FINE program commands executed during the current execution cycle, regardless of whether ALWAYS is specified.
The statement FINE 100 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins. This is the default state of the V+ system.

The FINE program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the FINE command causes an error.

Parameters

Parameter	Description
tolerance	Optional real value, variable, or expression that specifies the percentage of the standard fine tolerances that are used for each joint of the robot attached by the current execution task.
ALWAYS	Optional qualifier that establishes FINE as the default condition. FINE will remain in effect continuously until disabled by a COARSE program command. If ALWAYS is not specified, the FINE command will apply only to the next robot motion.

Details

The FINE program command enables the high-precision feature in the robot motion servo system so that 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. The value becomes the default for any subsequent FINE commands 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.

Example

The following example enables the high-precision feature for the next motion operation only.

```
FINE
```

The following example enables the high-tolerance feature for the next motion operation with the tolerance settings changed to 50% of the standard tolerance for each joint.

```
FINE 50
```

The following example enables the high-tolerance feature until it is explicitly disabled.

```
FINE ALWAYS
```

Related Keywords

3-4-26 COARSE on page 3-305
3-1-18 CONFIG on page 3-24
3-6-4 DELAY.IN.TOL on page 3-476

3-4-86 NONULL on page 3-393
3-4-88 NULL on page 3-396
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-55 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.

Syntax

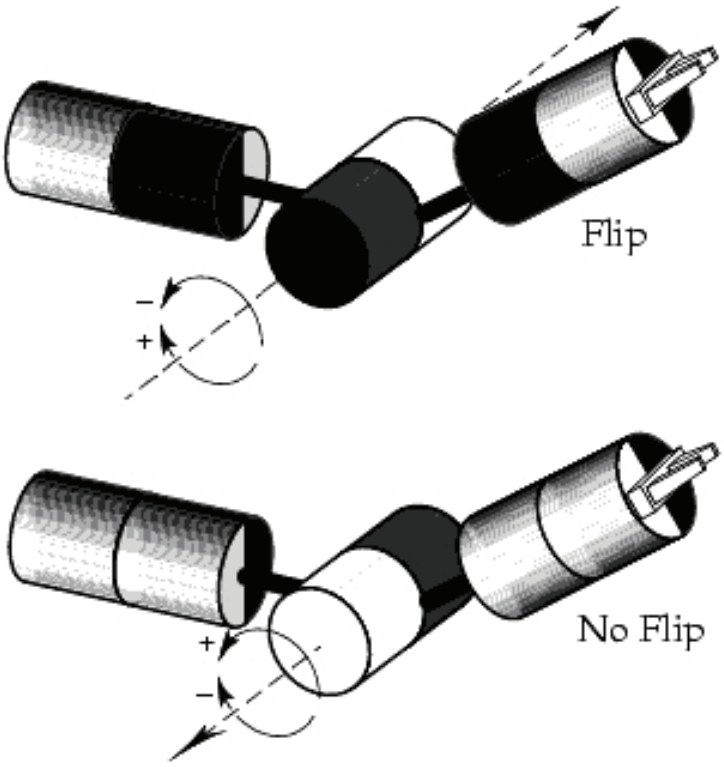
```
FLIP
```

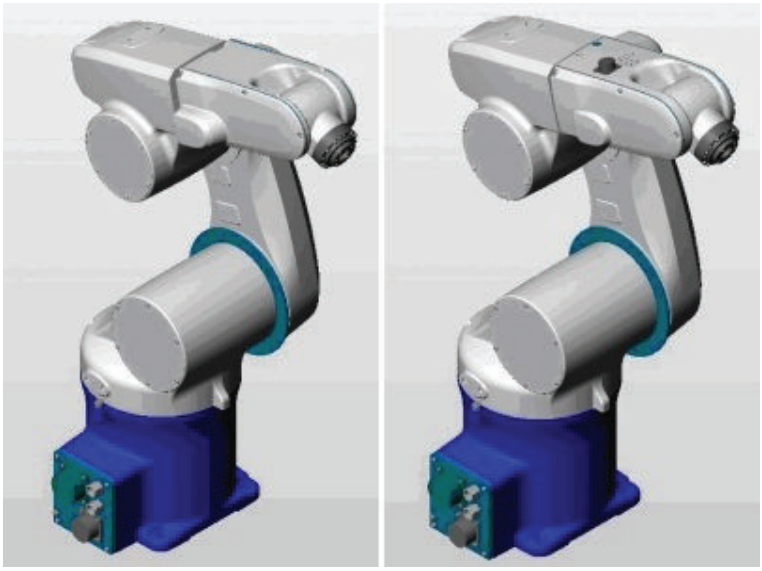
Usage Considerations

Configuration changes cannot be made during straight-line motions.
If the selected robot does not support a flip configuration, this command is ignored by the robot.
The FLIP program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. If the robot is not attached, this command has no effect.
This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the FLIP command causes an error.

Details

Enabling a FLIP configuration forces the wrist joint to have a negative rotation. Enabling a NOFLIP configuration forces a wrist joint to have a positive rotation.
Wrist joint angles are expressed as ±180°.
Robots can change configuration during joint-interpolated moves only.





Example

The following example demonstrates the use of the FLIP and NOFLIP commands.

```
FLIP
MOVE loc_a

NOFLIP
MOVE loc_a
```

Related Keywords

3-1-18 CONFIG on page 3-24
3-4-85 NOFLIP on page 3-392
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real valued function)

3-4-56 FOPEN

Create and open a new TCP connection.

Syntax

```
FOPEN (lun, mode) attribute
```

Usage Considerations

The logical unit must be attached before an open operation will succeed.

Parameters

Parameter	Description
lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number assigned to the TCP device. Refer to the *3-4-11 ATTACH* on page 3-279 command for a description of unit numbers.
mode	Type of TCP connection. 0 = client mode, 16 = server mode.
attribute	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 command is used to compose a single string that is passed to the TCP driver. 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. • A string constant, variable, or expression. • A real-valued constant, variable, or expression that is evaluated to determine a value to be used in the control string. • A format control specifier that determines the format of information in the control string.

*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.

Details

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. 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. If you are writing your own custom protocol, use a port number greater than 255.
The following table shows valid TCP attributes for the FOPEN instruction.

Attribute	Explanation
/CLIENTS	Defines the number of client connections allowable on a server. If omitted, a single client connection is assumed. The maximum number of client connections is 31.
/LOCAL_PORT	Defines the local port number for the connection. If omitted, a local port number is automatically assigned.
/REMOTE_PORT	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.

Example

The following example configures a TCP server with local port 260 to accept 5 client connections.

```
FOPEN (lun, 16) "/LOCAL_PORT 260 /CLIENTS 5"
```

The following example configures a TCP client connection that connects to port number 260 on the server called "server1".

```
FOPEN (lun, 0) "server1 /REMOTE_PORT 260"
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-4-49 FCLOSE on page 3-337
3-4-52 FDELETE on page 3-342
3-4-53 FEMPTY on page 3-343
3-4-63 FSET on page 3-361
3-1-58 IOSTAT on page 3-83

3-4-57 FOPENA

Opens a file for read-write-append access. If the specified file does not already exist, the file is created.

Syntax

FOPENA (lun, record_len, mode**) file_spec**

Usage Considerations

If the file already exists, no error occurs and the file position is set to the end of the file. Write operations then append to the existing file.
A logical unit must be attached before an open operation will succeed.
No more than 60 disk files and 160 network 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).

Parameters

Parameter	Description
lun	Real-valued expression defining the logical unit number of the disk device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 command 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 are processed. In this case, random access of records cannot be done. If the record length is non-zero, it specifies the length (in characters) of fixed-length records to be processed. Random access is allowed with fixed-length records.

Parameter	Description
mode	Optional real-valued expression defining how read access is to be executed. 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 (least significant bit): disable pre-reads (mask value = 1)** If this bit is OFF, V+ will read a record as soon as the file is opened (pre-read) and after each READ command in anticipation of subsequent READ requests. If this bit is ON, no such pre-reads are performed. **Bit 2: enable random access (mask value = 2)** If this bit is OFF, the file will be accessed sequentially. Records are read or written in the order they occur in the file. If this bit is ON, the file is 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 command specifies which record is accessed. **Bit 4: force disk write (mask value = 8)** If ON, the physical disk is written every time a record is written for a disk file being opened for write access. The directory or file allocation information is updated with each write. This mode is equivalent to (and faster than) closing the file after every write. It is significantly 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. The current default disk unit and directory path are considered as appropriate. Refer to *V+ User's Manual (Cat. No. I671)* for more information on disk units and directory paths.

Details

This command opens a disk file so that input / output operations can be performed. When the operations are complete, the file should be closed using an FCLOSE or DETACH command.

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-4-49 FCLOSE on page 3-337

3-4-56 FOPEN on page 3-348
3-4-58 FOPEND on page 3-352
3-4-59 FOPENR on page 3-354
3-4-60 FOPENW on page 3-356
3-4-62 FSEEK on page 3-360
3-1-58 IOSTAT on page 3-83
3-4-106 READ on page 3-421
3-4-140 WRITE on page 3-466

3-4-58 FOPEND

Opens a disk directory for reading.

Syntax

```
FOPEND (lun, record_len, mode) file_spec
```

Usage Considerations

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 return only records from the disk directory file that match the file name template. Any attempt to write to the directory file causes an error. Refer to *V+ User's Manual (Cat. No. I671)* for information about the format of directory records.
The file name and extension can include wild card characters (asterisks "*"). A wild card character within a file name or extension indicates that any character should be accepted in that position. A wild card character at the end of a filename or extension indicates that any trailing characters are acceptable. A wild- card 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 wild card extension.
A logical unit must be attached before an open operation will succeed.
No more than 60 disk files and 160 network 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).

Parameters

Parameter	Description
lun	Real-valued expression defining the logical unit number of the disk device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 command for a description of unit numbers.

Parameter	Description
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 are processed. In this case, random access of records cannot be done. If the record length is non-zero, 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 executed. 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 (least significant bit): disable pre-reads (mask value = 1)** If this bit is OFF, V+ will read a record as soon as the file is opened (pre-read) and after each READ command in anticipation of subsequent READ requests. If this bit is ON, no such pre-reads are performed. **Bit 2: enable random access (mask value = 2)** If this bit is OFF, the file will be accessed sequentially. Records are read or written in the order they occur in the file. If this bit is ON, the file is 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 command specifies which record is accessed. **Bit 4: force disk write (mask value = 8)** If ON, the physical disk is written every time a record is written for a disk file being opened for write access. The directory or file allocation information is updated with each write. This mode is equivalent to (and faster than) closing the file after every write. It is significantly 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. The current default disk unit and directory path are considered as appropriate. Refer to *V+ User's Manual (Cat. No. I671)* for more information on disk units and directory paths.

Details

This command opens a disk file so that input / output operations can be performed.When the operations are complete, the file should be closed using an FCLOSE or DETACH command.

Example

The following example opens the current default directory to find all the files with the extension DAT.

```
FOPEND (5) "*.dat"
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-4-49 FCLOSE on page 3-337
3-4-56 FOPEN on page 3-348
3-4-57 FOPENA on page 3-349
3-4-59 FOPENR on page 3-354
3-4-60 FOPENW on page 3-356
3-4-62 FSEEK on page 3-360
3-1-58 IOSTAT on page 3-83
3-4-106 READ on page 3-421

3-4-59 FOPENR

Opens a file for read-only access.

Syntax

FOPENR (lun, record_len, mode**) file_spec**

Usage Considerations

A logical unit must be attached before an open operation will succeed.
No more than 60 disk files and 160 network 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).

Parameters

Parameter	Description
lun	Real-valued expression defining the logical unit number of the disk device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 command for a description of unit numbers.

Parameter	Description
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 are processed. In this case, random access of records cannot be done. If the record length is non-zero, 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 executed. 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 (least significant bit): disable pre-reads (mask value = 1)** If this bit is OFF, V+ will read a record as soon as the file is opened (pre-read) and after each READ command in anticipation of subsequent READ requests. If this bit is ON, no such pre-reads are performed. **Bit 2: enable random access (mask value = 2)** If this bit is OFF, the file will be accessed sequentially. Records are read or written in the order they occur in the file. If this bit is ON, the file is 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 command specifies which record is accessed. **Bit 4: force disk write (mask value = 8)** If ON, the physical disk is written every time a record is written for a disk file being opened for write access. The directory or file allocation information is updated with each write. This mode is equivalent to (and faster than) closing the file after every write. It is significantly 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. The current default disk unit and directory path are considered as appropriate. Refer to *V+ User's Manual (Cat. No. I671)* for more information on disk units and directory paths.

Details

This command opens a disk file so that input / output operations can be performed. When the operations are complete, the file should be closed using an FCLOSE or DETACH command.

Example

The following example opens 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, pre-reads will occur and the records will be accessed sequentially (this is required for variable-length records).

```
FOPENR (5) "data.dat"
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-4-49 FCLOSE on page 3-337
3-4-56 FOPEN on page 3-348
3-4-57 FOPENA on page 3-349
3-4-58 FOPEND on page 3-352
3-4-60 FOPENW on page 3-356
3-4-62 FSEEK on page 3-360
3-1-58 IOSTAT on page 3-83
3-4-106 READ on page 3-421

3-4-60 FOPENW

Opens a file for read-write access. If the file already exists, an error occurs.

Syntax

FOPENW (lun, record_len, mode**) file_spec**

Usage Considerations

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. 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.
A logical unit must be attached before an open operation will succeed.
No more than 60 disk files and 160 network 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).

Parameters

Parameter	Description
lun	Real-valued expression defining the logical unit number of the disk device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 command 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 are processed. In this case, random access of records cannot be done. If the record length is non-zero, 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 executed. 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 (least significant bit): disable pre-reads (mask value = 1)** If this bit is OFF, V+ will read a record as soon as the file is opened (pre-read) and after each READ command in anticipation of subsequent READ requests. If this bit is ON, no such pre-reads are performed. **Bit 2: enable random access (mask value = 2)** If this bit is OFF, the file will be accessed sequentially. Records are read or written in the order they occur in the file. If this bit is ON, the file is 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 command specifies which record is accessed. **Bit 4: force disk write (mask value = 8)** If ON, the physical disk is written every time a record is written for a disk file being opened for write access. The directory or file allocation information is updated with each write. This mode is equivalent to (and faster than) closing the file after every write. It is significantly 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.

Parameter	Description
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. The current default disk unit and directory path are considered as appropriate. Refer to *V+ User's Manual (Cat. No. I671)* for more information on disk units and directory paths.

Details

This command opens a disk file so that input / output operations can be performed. When the operations are complete, the file should be closed using an FCLOSE or DETACH command.

Example

The following example opens the file named x.d on the device D for read-write access using fixed-length records of 32 characters each. The mode value 3 has both bits 1 and 2 set therefore pre-reads are disabled and the random access method is used.

```
FOPENW (5, 32, 3) "D:x.d"
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-4-49 FCLOSE on page 3-337
3-4-56 FOPEN on page 3-348
3-4-57 FOPENA on page 3-349
3-4-58 FOPEND on page 3-352
3-4-59 FOPENR on page 3-354
3-4-62 FSEEK on page 3-360
3-1-58 IOSTAT on page 3-83
3-4-106 READ on page 3-421
3-4-140 WRITE on page 3-466

3-4-61 FOR

Execute a program loop a specified number of times.

Syntax

```
FOR loop_var = initial TO final STEP increment
```

Usage Considerations

Every FOR command must have an associated END command. An External variable cannot be used for the loop_var parameter.

Parameters

Parameter	Description
loop_var	Real valued variable that is initialized when the FOR command is executed. This is incremented each time the loop is executed. The variable cannot be a specified value or expression. It is typically declared as an AUTO variable. An External variable cannot be used for the loop_var parameter.
initial	Real value that determines the value of the loop_var the first time the loop is executed.
final	Real value that establishes the value to be compared to the loop_var to determine when the loop should be terminated.
increment	Optional real-value that establishes the value to be added to the loop_var every time the loop is executed. If omitted, the increment defaults to 1 and STEP may also be omitted.

Details

The commands between the FOR statement and the matching END statement are executed repeatedly and loop_var is changed each time by the value designated as increment.
The processing of this structure is as is described below.

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 with the following conditions.
   - If increment is positive and loop_var is greater than final, skip to item 7 below.
   - If increment is negative and loop_var is less than final, skip to item 7 below.
4. Execute the keywords following the FOR statement.
5. When the END step is reached, add the value of increment to the loop_ var..
6. Go back to item 3 above.
7. Continue program execution at the first keyword after END. loop_var retains the value it had at the time of the evaluation in item 3 above unless it has been intentionally incremented or decremented within the logic contained in the loop (or elsewhere if not declared as an AUTO variable).

The group of keywords in the FOR structure may not be executed if the evaluation 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 keywords in the FOR loop are executed. Any changes to the values of these parameters within the FOR loop have no effect on the processing of the FOR structure.

Changes to the loop_var within the loop affect the operation of the loop and should normally not occur in a typical program.
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 times 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

3-4-38 DO on page 3-321
3-4-44 END on page 3-328
3-4-47 EXIT on page 3-334
3-4-84 NEXT on page 3-391
3-4-138 WHILE on page 3-463

3-4-62 FSEEK

Position a file open for random access and initiate a read operation on the specified record.

Syntax

FSEEK (logical_unit, record_number)

Usage Considerations

A file must be open for random access on the specified logical unit. Refer to the FOPEN_ commands for more information).
For efficiency in most applications, the file should be opened in no pre-read mode.

Parameters

Parameter	Description
logical_unit	Real-valued expression that identifies the device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 command for a description of unit numbers.

Parameter	Description
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 command, an application program can initiate a disk seek and possible read operation immediately after a READ command is processed but before processing the data.
Any error in the specification of this command (such as referencing an invalid unit)causes a program error and halts program execution. Errors associated with performing the actual seek operation (such as end of file or device not ready) do 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 to always test whether each file operation completed successfully by testing the value from IOSTAT.

Example

The following example will seek record number 130 in the file open on logical unit 5.

```
FSEEK (5, 130)
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-57 FOPENA on page 3-349
3-4-58 FOPEND on page 3-352
3-4-59 FOPENR on page 3-354
3-4-60 FOPENW on page 3-356
3-1-58 IOSTAT on page 3-83
3-4-106 READ on page 3-421

3-4-63 FSET

Set or modify attributes of a network device.

Syntax

```
FSET (logical_unit) attribute_list
```

Usage Considerations

The IOSTAT function should be used after each FSET command to determine the success of the FSET operation.

Parameters

Parameter	Description
logical_unit	Real value, variable, or expression interpreted as an integer that defines the logical unit number of the network device. Refer to the *3-4-11 ATTACH* on page 3-279 program command for a description of unit numbers.
attribute_list	List of string constants, expressions, real values, variables, and format specifiers used to define the characteristics of the network device. Refer to the description of the *3-4-56 FOPEN* on page 3-348 program command for detailed information on this parameter.

Details

The TCP Transmission Control Protocol network device may be referenced with the FSET command. You can use the attributes listed in the following table when accessing these devices with the FSET command.

Attribute	Description
/ADDRESS	IP address (applies only to the TCP device).
/NODE	Node name.

You may define new nodes on the network using the FSET command to access a logical unit that has been attached to the TCP device. The string used with the FSET command has the same format as that used with the NODE statement in the V+ configuration file.

Example

The following example defines a new node called "SERVER2" with the IP address 172.16.200.102.

```
ATTACH (lun, 4) "TCP"
FSET (lun) "/NODE 'SERVER2' /ADDRESS '172.16.200.102'"
```

Related Keywords

3-4-56 FOPEN on page 3-348
3-1-58 IOSTAT on page 3-83

3-4-64 GLOBAL

Declare a variable to be global and specify the type of the variable.

Syntax

```
GLOBAL type variable, ..., variable
```

Usage Considerations

GLOBAL statements must appear before any executable statement in the program.

Parameters

Parameter	Description	
type	If the type parameter is specified, all the variables must match that type (REAL, DOUBLE, OR LOC). Array variables must have their indexes specified explicitly, indicating the highest valid index for the array. 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.	
	LOC	Location variable (transformation or precision point).
	REAL	Single-precision real variable.
	DOUBLE	Double-precision real variable. Refer to the *3-4-64 GLOBAL* on page 3-362GLOBAL command for details on the default type.
variable	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 32.	

Details

Variables that are not declared to be AUTO or LOCAL are GLOBAL by default. Double precision and location global variables do not need to be declared.
Global variables can be accessed by any program that does not declare a LOCAL or AUTO variable of the same name. For example, if program_a declares var1 to be a GLOBAL variable and program_b declares var1 to be AUTO variable, pro- gram_b cannot use or alter GLOBAL var1. A new copy of variable var1 that is specific to program_b is created each time program_b executes.

Example

The following example creates 2 string variables and 1 untyped global variable.

```
GLOBAL $str_1, $str_2, x
```

The following example creates 1 global precision point global variable.

```
GLOBAL LOC #ppoint_1
```

The following example creates 2 double precision real global variables.

```
GLOBAL var_1, var_2
```

Related Keywords

3-4-12 AUTO on page 3-283
3-4-76 LOCAL on page 3-377
3-4-48 EXTERNAL on page 3-335

3-4-65 GOTO

Perform an unconditional branch to the program step identified by the given label.

Syntax

```
GOTO label
```

Parameters

Parameter	Description
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 command causes program execution to jump to the line that contains the specified step label. 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.

Precautions for Correct Use

Modern, structured programming considers GOTO statements to be poor programming practice. It is recommended to use one of the other control structures in place of GOTO statements.

Example

The following example asks you to enter a number from 1 to 100. If the number input is not in that range, the GOTO 10 command causes execution to jump to step label 10.

```
10 PROMPT "Enter a number from 1 to 100: ", number
IF (number < 1) OR (number > 100) THEN
    TYPE /B, /C1, "*Invalid response*", /C1
GOTO 10
END
```

Related Keywords

3-4-38 DO on page 3-321

3-4-47 EXIT on page 3-334
3-4-61 FOR on page 3-358
3-4-69 IF...THEN on page 3-368
3-4-68 IF...GOTO on page 3-367
3-4-84 NEXT on page 3-391
3-4-138 WHILE on page 3-463

3-4-66 HALT

Stop program execution and do not allow the program to be resumed.

Syntax

HALT

Usage Considerations

The PROCEED monitor command cannot be used to resume program execution after a HALT program command has been used to stop a program.
HALT forces an FCLOSE and / or DETACH operation on the disk logical units as required.

Details

The HALT program command causes a break and then terminates execution of the application program regardless of any program loops remaining to be completed. The message (HALTED) is displayed.
After termination by a HALT command, program execution cannot be resumed with a PROCEED or RETRY command.

Example

The following example halts the execution of the program task after the robot reaches the location defined as "safe_loc".

```
MOVE safe_loc
BREAK
HALT
```

Related Keywords

3-4-93 PAUSE on page 3-401
3-4-111 RETURN on page 3-427
3-4-125 STOP on page 3-448

3-4-67 HERE

Set the value of a transformation or precision-point variable equal to the current robot location.

Syntax

```
HERE location_var
```

Usage Considerations

The HERE program command returns information for the robot selected by the task executing the command.
If the V+ system is not configured to control a robot, use of the HERE command will cause an error.

Parameters

Parameter	Description
location_var	Transformation, precision point, or compound transformation that ends with a transformation variable.

Details

The HERE program command sets the value of a transformation or precision- point variable equal to the current robot location.
Normally, the robot location is determined by reading the instantaneous values of the joint encoders. If the robot has either backlash or linearity compensation enabled, the commanded robot location is used.
If the location_var is a compound transformation, only the right-most transformation is defined. Its value is set equal to the current robot location relative to the reference frame determined by the other transformations. An error message results if any of the other transformations are not already defined.

Example

The following example sets the transformation "part" equal to the current robot location.

```
HERE part
```

The following example assigns the current location of the robot to the precision point "#part"

```
HERE #part
```

Related Keywords

3-2-30 HERE on page 3-207 (monitor command)
3-1-48 HERE on page 3-68 (transformation function)
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-117 SET on page 3-433

3-4-68 IF...GOTO

Branch to the specified step label if the value of the logical expression is TRUE (non-zero).

Syntax

```
IF logical_expr GOTO label
```

Usage Considerations

In general, the preferred programming practice is to use the IF ... THEN command rather than this command.

Parameters

Parameter	Description
logical_expr	Real-valued expression whose value is tested for TRUE (nonzero) or FALSE (0).
label	Label of the program step to which execution is to branch.

Details

If the value of the expression is non-zero, program execution branches and begins executing the keyword with a label matching the one specified. If the value of the expression is zero, the next keyword is executed as usual.
If the specified statement label is not defined, the program is not executable. Any attempt to branch to an undefined label is identified when the editor is exited and when the program is loaded into memory from a disk file.
The most common use for IF ... GOTO is as an exit-on-error condition (see example below).

Example

The following example 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)
100 IF IOSTAT(dlun) < 0 THEN
    TYPE $ERROR(IOSTAT(dlun))
END
```

Related Keywords

3-4-65 GOTO on page 3-364
3-4-69 IF...THEN on page 3-368

3-4-69 IF...THEN

Conditionally execute a group of keywords (or one of two groups) depending on the result of a logical expression.

Syntax

```
IF logical_expr THEN
    first keyword area
ELSE
    second keyword area
END
```

Usage Considerations

Every IF ... THEN command must have an associated END command.
The ELSE command may be omitted if there are no items in the second keywords area shown in the syntax section above.

Parameters

Parameter	Description
logical_expr	Real-valued expression whose value is tested for TRUE (non-zero) or FALSE (0).

Details

This control structure provides a means for conditionally executing one of two groups of keywords.
The control structure functions as described below (refer to the *Syntax* on page 3-368 section above).

1. The logical_expr is evaluated. If the result is TRUE, proceed to item 2 below. If the result is FALSE (0), skip to item 4 below.
2. The first keyword area is executed.
3. Skip to item 5 below.
4. If there is an ELSE command, the second keyword area is executed.
5. Program execution continues at the next keyword after the END command.

The ELSE and END steps must be on lines by themselves as shown in the syntax section above.
There are no restrictions on the keywords that can be in either group in the structure and nested IF structures can be used if necessary.

Example

In the following example, if the value of variable "row" is greater than 5, the expression row > 5 will be TRUE (-1.0) and variable "spacing" is assigned a value of 10. If the value of variable "row" is less than 5, the expression row > 5 will be FALSE (0) and variable "spacing" is assigned a value of 20.

```
21    IF row > 5 THEN
22        spacing = 10
23    ELSE
24        spacing = 20
25    END
```

The following program example determines 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. The outer IF does not include an ELSE clause.

```
IF DEFINED(input.signal) THEN
    IF SIG(input.signal) THEN
        TYPE "The input signal is ON"
    ELSE
        TYPE "The input signal is OFF"
    END
END
```

Related Keywords

3-4-23 CASE on page 3-301
3-1-26 DEFINED on page 3-36
3-4-42 ELSE on page 3-326
3-4-68 IF...GOTO on page 3-367

3-4-70 IGNORE

Cancel the effect of a REACT or REACTI program command.

Syntax

IGNORE signal

Usage Considerations

Only digital I/O signals that physically exist in the system as inputs are available for reaction monitoring.
The IGNORE program command must be executed by the same program task that initiated by the REACT or REACTI program commands.

Parameters

Parameter	Description
signal	Digital input signal number (see details below).

Details

The IGNORE program command disables continuous monitoring of the specified signal, canceling the effect of the last REACT or REACTI for this signal.
The following signals can be used with the REACT program command.

Signal Type	Range	Details
Digital Input	1001 to 1999	Signals are only available if they physically exist in the system.
Software	2001 to 2999	All signals are available
Host	4001 to 4999	

Example

The following example stops monitoring of the digital input or soft signal identified by the value of "test".

```
IGNORE test
```

Related Keywords

3-4-77 LOCK on page 3-379
3-4-103 REACT on page 3-415
3-4-105 REACTI on page 3-419

3-4-71 JMOVE

Moves all robot joints to positions described by a list of joint values. The robot performs a coordinated motion in joint-interpolated mode.

Syntax

JMOVE expression1,...,expressionn

Usage Considerations

You must specify at least one expression (joint), in order to move the robot.

Parameters

Parameter	Description
expression1	Optional expression for the joint-1 value. expression
expressionn	Optional expression for the nth joint value.

Details

You can specify a maximum of 12 expressions.
If an expression is omitted, that joint is not moved.
If more expressions are specified than there are joints for a robot, the extra expressions are ignored.

Example

The following example moves joint 1 and joint 3 to the positions represented by variables "j1" and "j3".

```
JMOVE j1,,j3
```

3-4-72 JOG

Jogs the specified joint of the robot or moves the robot tool along the specified cartesian direction.

Syntax

```
JOG (status) robot, mode, axis, speed, location, appro_dist
```

Usage Considerations

Each time the JOG program command is executed, the robot moves for the time specified with the JOG.TIME system parameter.
The specified robot cannot be attached by any other task when using a mode other than COMP. Otherwise, the error message *Robot interlocked* is generated. The robot can be attached by the current program, but it does not need to be attached. If the robot is not attached when the JOG program command is executed, attach the robot after the JOG program command before executing any other motion commands.
After the robot is moved with the JOG command, the system is left in MANUAL mode (i.e., as though a manual mode had been selected on the pendant). The statement JOG mode 5 (or the pendant) can be used to restore COMP mode.
Otherwise, an error *COMP mode disabled* will be returned when a task attempts to attach the robot.
If a joint is out of range, the JOG command can be used to return joint into range.

Parameters

Parameter	Description
status	An optional status variable. Returns 1 for success; otherwise, contains a V+ error code.

Parameter	Description	
robot	Specifies the robot number.	
mode	Specifies the jog mode as described below.	
	-1	Keep-alive mode. Continues the previous operation for the time specified using the JOG.TIME system parameter.
	1	Free joint mode. A positive speed will put the specified joint(s) in Free mode. A negative speed will take the specified joint(s) out of Free mode.
	2	Individual joint control.
	3	World coordinates control.
	4	Tool coordinates control.
	5	Restore COMP mode.
	6	unused.
	7	Jog toward the specified location using the specified speed.
	8	Jog toward alignment of the robot tool-Z axis with the nearest World axis.
	9	Cartesian control relative to a frame defined by the specified location.
axis	Specifies the joint number or Cartesian coordinate (X=1, Y=2, ...), depending on the specified jog mode (see above), for the desired motion. This parameter is ignored for modes 7 and 8, but a value must always be specified.	
speed	Specifies the speed and direction of the motion. This is interpreted as a percentage of the speed in manual mode. Values above 100 are interpreted as 100%, values below - 100 are interpreted as -100%. If Free mode is specified, a positive speed will put the given joint in free mode and a negative speed will put the joint out of free mode.	
location	Optional transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move. This parameter is ignored and can be omitted for all modes except 7 and 9.	
appro_dist	Optional real-valued expression that specifies the distance. The position of the destination location is offset from the given location by the distance given, measured along the Z-axis of the specified location in the negative direction. A positive distance sets the tool back (negative tool-Z) from the specified location. A negative distance offsets the tool forward (positive tool-Z). This parameter is used only for mode 7.	

Details

When the status variable is supplied and there is an error, the JOG command does not cause program execution to stop. The error is returned in the status variable.

A new JOG command can be executed before the previous motion is completed. For extended smooth motion, subsequent JOG commands should be executed within the JOG.TIME value of the

previous JOG command. The keep-alive mode can be used for this purpose. The keep-alive mode will have no effect after the timeout of the JOG.TIME value. It has an effect only before the robot stops.
The following error conditions can be reported when the command is processed.

- Mode 1: The error *Illegal joint number* (-609) is returned if FREE mode is not permitted for the specified joint.
- Mode 2: The error *Joint control of robot not possible* (-938) is returned if the robot does not support joint control.
- Modes 3, 4, 8, 9: The error *Cartesian control of robot not possible* (-635) is returned if the robot does not support Cartesian control.
- Mode 7: If the location cannot be reached, the motion stops at the limit of possible motion and the error *Location out of range* (-610) is returned when the motion stops. If any other motion error occurs during the motion (e.g., an obstacle is encountered), the associated error is reported.
- Modes 7 and 9: The error *Missing argument* (-454) is returned if a location is not specified. For mode 7, a straight-line motion is performed toward the specified location if the location is specified with a trans- formation. A joint-interpolated motion is performed if the location is specified with a precision point.

When a robot joint is out-of-range, it can be driven into range with a method described below.

- Enter MAN mode on the Front Panel and manually control the joint.
- Put the pendant in COMP mode and use the JOG command to move the joint back into range. JOG is allowed only in pendant COMP mode.

Use of COMP mode when a joint is out of range is restricted. All motion commands (except JOG) return a *Position out of range* error in that situation. In addition, JOG can move the joint only in the direction that moves the joint back into range.

Example

The following example jogs joint 3 in a negative direction.

```
JOG 1, 2, 3, -10
```

The following example jogs the X-axis in world mode.

```
JOG 1, 3, 1, 10
```

The following example jogs the Y-axis in tool mode.

```
JOG 1, 4, 2, 10
```

The following example jog the robot tool toward "loc1".

```
JOG 1, 7, 1, 10,loc1
```

The following example jogs the robot tool toward a location 50mm above "loc1".

```
JOG 1, 7, 1, 10,loc1, 50
```

Related Keywords

3-4-72 JOG on page 3-371
3-5-3 JOG.TIME on page 3-470
3-4-81 MOVE on page 3-384

3-4-73 KEYMODE

Set the behavior of a group of keys on the pendant.

Syntax

```
KEYMODE first_key,last_key = mode
```

Usage Considerations

The pendant must be attached before KEYMODE can be processed. Refer to the T20 Pendant User's Manual (Cat. No. I601) for information about the key number assignments.

Parameters

Parameter	Description
first_key	Real-valued expression that defines the first key number in a set of keys to be affected
last_key	Real-valued expression that defines the last key number in a set of keys to be 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

Details

The various key modes are described below. Refer to the description of the *3-4-98 PENDANT* on page 3-406 real-valued function for more information on interaction with the 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 then waits until one of the keys programmed in KEYBOARD MODE is pressed and then number of the key is returned.

Type-ahead is not possible. The program does not detect any keys that are pressed while there is no PENDANT(0) function pending.

● 1 - Toggle Mode

When you press a key that is in this mode, the internal state maintained by V+ is toggled. The current state of the key can be evaluated as necessary.

● 2 - Level Mode

The key's current level is maintained by the pendant and may be requested. The key's state is ON only when it is being pressed. This is useful for cursor control, for example. The value returned is not valid if the pendant is not attached.

● Attach/Detach Requirements

The pendant must be attached (with the ATTACH program command) before the program can read keys using the PENDANT function, set the modes of any of the keys, or send text to the display.

● Defaults

The key modes default to keyboard mode when the pendant is attached.

Example

The following example will set the manual control soft keys to level mode.

```
KEYMODE 1,5 = 2
```

Related Keywords

3-4-11 ATTACH on page 3-279 (program command)
3-4-98 PENDANT on page 3-406 (real-valued function)

3-4-74 KILL

Clear a program execution stack and detach any I/O devices that are attached.

Syntax

KILL task_number

Usage Considerations

The KILL program command cannot be used while the specified program task is executing.
The KILL program command has no effect if the specified task execution stack is empty.

Parameters

Parameter	Description
task_number	Optional real value, variable, or expression interpreted as an integer that specifies which program task is to be cleared.

Details

The KILL program command 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 program command, is terminated by an ABORT keyword, or an error condition occurs while an I/O device is attached or a file is open. If a

limited-access I/O device is left attached, no other program task can use that device until it is detached.
The KILL command always accesses task 0 if the task number is omitted.

Example

The following example will execute the KILL operation on task 1 if signal 2001 is true.

```
IF (SIG(2001)==TRUE )THEN
    KILL(1)
END
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-58 STATUS on page 3-244

3-4-75 LEFTY

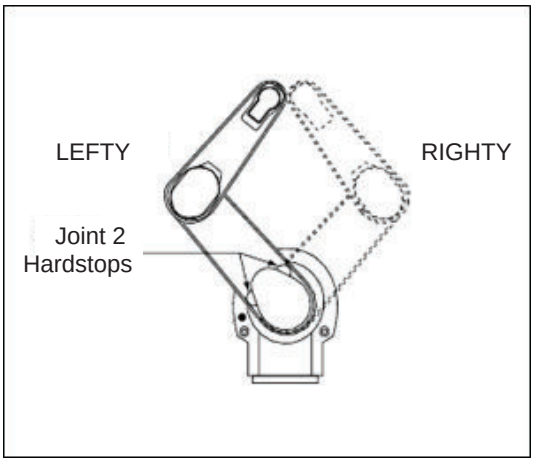
Request a change in the robot configuration during the next motion to make the first two links of a SCARA robot use the left arm orientation.

Syntax

LEFTY

Usage Considerations

Configuration changes cannot be made during straight-line motions.
If the selected robot does not support a left-handed configuration, this command is ignored by the robot.
The LEFTY program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. If the robot is not attached, this command has no effect.
This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the LEFTY command causes an error.
The following figure shows the LEFTY / RIGHTY configurations for the top view of a SCARA robot.



Example

The following example will move the robot to location "point1" in the lefty configuration.

```
LEFTY
MOVE point1
```

Related Keywords

3-1-18 CONFIG on page 3-24
3-4-113 RIGHTY on page 3-428
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)

3-4-76 LOCAL

Declare permanent variables that are defined only within the current program.

Syntax

LOCAL type **variable**, ..., variable

Usage Considerations

Subroutines can be called simultaneously by multiple program tasks and recursively by a single task. Local and global variables can be corrupted if such calls occur inadvertently. The use of automatic variables in place of local variables is recommended for this reason.
LOCAL statements must appear before any executable statement in the program.
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 commands.

Parameters

Parameter	Description	
type	If the type parameter is specified, all the variables must match that type (REAL, DOUBLE, or LOC). Array variables must have their indexes specified explicitly, indicating the highest valid index for the array. 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.	
	LOC	Location variable (transformation or precision point).
	REAL	Single-precision real variable.
	DOUBLE	Double-precision real variable. Refer to the *3-4-64 GLOBAL* on page 3-362 command for details on 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 not have their indexes specified.	

Details

This command is used to declare variables to be defined only within the current program. A local variable can be referenced only within its own program. 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 unpredictable program errors. Therefore, automatic variables should be used for all temporary local variables to minimize the chance of errors. Refer to the *3-4-12 AUTO* on page 3-283 command for more information.
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 res- ult 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 with a different type will result in the error message "Attempt to redefine variable type".
Local variables can be referenced with monitor commands such as DELETE_, DO , HERE , LIST, TOOL by using the optional context specifier @ as shown in the example below.

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

Example

The following example declares the variables "loc.a", "$ans**"**, and "i" to be local to the current program.

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

Related Keywords

3-4-12 AUTO on page 3-283
3-4-76 LOCAL on page 3-377
3-2-57 STACK on page 3-242
3-4-38 DO on page 3-321
3-2-30 HERE on page 3-207 (monitor command)
3-2-67 TOOL on page 3-258

3-4-77 LOCK

Set the program reaction lock-out priority to the value given.

Syntax

```
LOCK priority
```

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 are processed. Before using this command, be sure you know what reactions (and their priorities) are active.

Parameters

Parameter	Description
priority	Real-valued expression with a value from 0 to 127, which becomes 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 command 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 command. For example, if a LOCK command changes the temporary program priority to 20, any REACT or REACTI interrupts with lower priority values are deferred.

If the execution of the REACTI operation is postponed and the LOCK command is used to change the REACTI operation to a higher priority, the system will execute the operation. This differs from the behavior of the normal REACTI operation. After the system executes the operation, execute the REACTI operation.
If the execution of the REACTI operation is postponed and the LOCK command is used to set the priority of the REACTI operation to a higher priority, the current command will be executed without stopping immediately. After the current command is executed, issue the REACTI command. REACTE routines cannot be deferred by priority considerations.
Deferred reactions are not ignored. Every time a new LOCK command 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.
Although a LOCK command can be used to change the program priority within a reaction program, the priority still returns to its pre-reaction value when a RETURN is executed in the program. This occurs only when executing a RETURN from a reaction program.

Example

The following example increases the program priority by 10.

```
LOCK PRIORITY+10
```

Related Keywords

3-1-82 PRIORITY on page 3-105
3-4-103 REACT on page 3-415
3-4-105 REACTI on page 3-419

3-4-78 MC

Introduce a monitor command within a Monitor Command program.

Syntax

```
MC monitor_command
```

Usage Considerations

The MC command can be contained only within a Monitor Command program. Monitor Command programs can contain only MC commands, blank lines, and comment lines.

Additional Information

Refer to *V+ User's Manual (Cat. No. I671)* for more information about Monitor Command programs.

Parameters

Parameter	Description
monitor_command	Any valid V+ monitor command keyword.

Details

Command programs are created using one of the V+ editors. To indicate to the editor that a Monitor Command program is being created, every operation line of a Monitor Command program must begin with the letters MC followed by one or more spaces. As with regular application programs, Monitor Command programs can contain blank lines and comment lines to add clarity.
Every non-blank line of a Monitor Command program must contain a monitor command or a comment. Monitor commands and program commands cannot be mixed. Program commands can be included by using the DO command. You can type a line with the form "MC DO".

Example

The following Monitor Command program example loads disk files, prepares for execution of a program, and begins the execution. A DO command is used to include a MOVE command.

```
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

3-2-7 COMMANDS on page 3-175
3-4-38 DO on page 3-321
3-4-79 MCS on page 3-381

3-4-79 MCS

Invoke a monitor command from an application program.

Syntax

```
MCS string
```

Parameters

Parameter	Description
string	String value, variable, or expression that defines one of the following V+ monitor commands: DELETE , DELETEL , DELETEM , DELETEP, DELETER , DELETES , FCOPY , LOAD , STORE , STOREL , STOREM , STOREP , STORER , STORES

Details

Monitor commands are typically invoked from the Monitor Window or from Monitor Command programs which contain only monitor commands. The MCS command can be used to invoke the following monitor commands from an application program.

- DELETE
- DELETEL
- DELETEM
- DELETEP
- DELETER
- DELETES
- FCOPY
- LOAD
- STORE
- STOREL
- STOREM
- STOREP
- STORER
- STORES

Using these commands, an application program can store, load, and copy pro- grams to and from disk, and also delete programs from memory. Similarly, variables can be deleted from memory when they are no longer needed. Also, vision prototypes can be renamed.

Loading, storing, and deleting programs and global variables is not interlocked for multi-task access in V+. Therefore, if you are incorporating multiple MCS commands in a program, you will need to use TAS interlocks to prevent multiple tasks from issuing the commands. Refer to the *3-1-104 TAS* on page 3-131 command for more information.

If the monitor command specified in the string parameter contains a blank program context (it contains @), any variables listed in the command are treated as though they are referenced within the program containing the MCS command. Refer to *V+ User's Manual (Cat. No. I671)* for more information about program context.

Program execution is not stopped if an error occurs while processing the monitor command. The ERROR real-valued function can be used after the MCS command to check for the occurrence of an error.

If a DELETE_ command is used within a subroutine to delete one of the subroutine parameters (one of the variables in the .PROGRAM statement), the variable is not deleted and no error condition is recorded.

If the FCOPY option is used, logical units 5 (disk 1) and 6 (disk 2) must be available. If LOAD or STORE_ is used, logical unit 5 must be available.

Example

The following example loads a disk file, executes the program in the file, and deletes the program from the system memory. Another program file is then loaded into memory and executed.
Although this simple example can also be implemented with a Monitor Command program, the following demonstrates use of the MCS command 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

3-2-11 DELETE on page 3-182
3-2-12 DELETEL on page 3-183
3-2-13 DELETEM on page 3-184
3-2-14 DELETEP on page 3-185
3-2-15 DELETER on page 3-186
3-2-16 DELETES on page 3-188
3-1-37 ERROR on page 3-48
3-2-23 FCOPY on page 3-197
3-2-40 LOAD on page 3-222
3-4-78 MC on page 3-380
3-2-59 STORE on page 3-247
3-2-60 STOREL on page 3-249
3-2-61 STOREM on page 3-250
3-2-62 STOREP on page 3-252
3-2-63 STORER on page 3-253
3-2-64 STORES on page 3-254
3-1-104 TAS on page 3-131

3-4-80 MULTIPLE

Allow full rotations of the robot wrist joints.

Syntax

```
MULTIPLE ALWAYS
```

Usage Considerations

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

MULTIPLE ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins. This is the default state of the V+ system.
The MULTIPLE program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the MULTIPLE command causes an error.

Parameters

Parameter	Description
ALWAYS	Optional qualifier that establishes MULTIPLE as the default condition. If ALWAYS is specified, MULTIPLE will remain in effect continuously until disabled by a SINGLE program command.If ALWAYS is not specified, the MULTIPLE command applies only to the next robot motion.

Details

While MULTIPLE is active, full rotations of the wrist joints are used as required during motion planning and execution.
The MULTIPLE setting is ignored if NOOVERLAP is active.

Related Keywords

3-4-87 NOOVERLAP on page 3-394
3-4-121 SINGLE on page 3-438

3-4-81 MOVE

Initiate a robot motion to the position and orientation described by the given location with joint-interpolated motion.

Syntax

```
MOVE location
```

Usage Considerations

MOVE causes a joint-interpolated motion.
This command can be executed by any program task as long as the task has attached a robot. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing this command causes an error.

Parameters

Parameter	Description
location	Transformation,precision point, location function, or com- pound transformation that specifies the destination to which the robot is to move.

Details

The MOVE program command causes a joint-interpolated motion. 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 since the last MOVE operation (for example, issuing a LEFTY command) are executed during the motion.

Example

The following example moves to the location described by the precision point "#pick" with joint-interpolated motion.

```
MOVE #pick
```

Related Keywords

3-4-9 APPRO on page 3-277
3-4-10 APPROS on page 3-278
3-4-34 DEPART on page 3-316
3-4-35 DEPARTS on page 3-317
3-4-82 MOVEC on page 3-385
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-82 MOVEC

Initiate a circular / arc-path robot motion using the positions and orientations described by the given locations.

Syntax

```
MOVEC(angle, turn) location1, location2
MOVEC(angle, turn) center
```

Usage Considerations

The MOVEC program command can be executed by any program task as long as the task has attached a robot. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing this command causes an error.

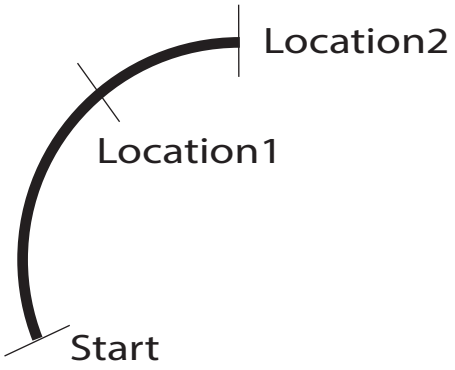
Parameters

Parameter	Description
angle	Real-valued expression that specifies the angle of the arc in degrees. This parameter is optional if location2 is specified. The angle parameter can be a positive or negative number but must be specified within the range of - 360 to +360. A positive value will result in a clockwise tool motion. A negative value will result in a counterclockwise tool motion.
turn	Optional boolean expression that specifies whether the tool should rotate with the arc. If turn is omitted or 0, then the tool orientation will stay constant with a MOVEC (angle) center statement and end at the orientation of location2 for a MOVEC location1, location2 statement . If turn is non-zero, the tool orientation will be rotated by the angle of the arc around the axis of the circle and maintain a constant orientation relative to the trajectory.
center	Transformation,precision point, location function, or compound transformation that specifies the center of the circle.
location1	Transformation, precision point, location function, or compound transformation that specifies an intermediate location on the circle / arc through which the robot is to move.
location2	Optional transformation, precision point, location function,or compound transformation that specifies the end-point of the circle / arc to which the robot is to move. If this parameter is not supplied, then angle must be specified.

Details

● MOVEC(angle,turn) location1, location2

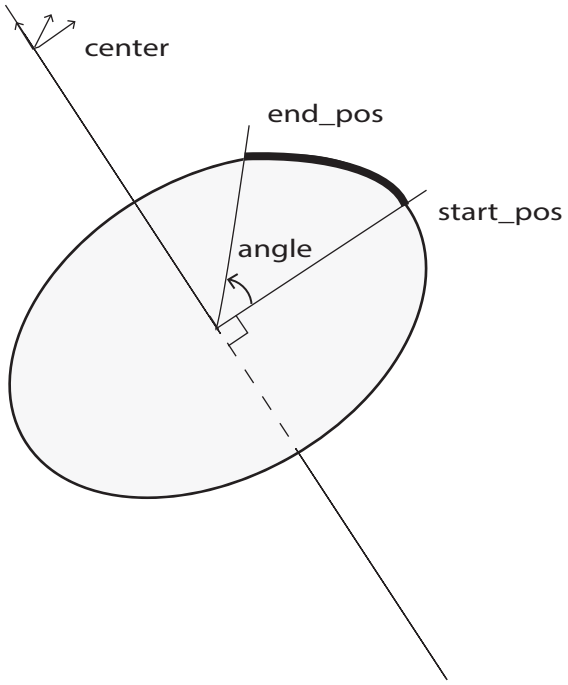
The MOVEC program command syntax is designed to create a circle / arc path ending at a location defined by the location2 parameter. The circle / arc path can begin from the current robot position or the current robot destination when using a continuous path motion. The intermediate location designated with the location1 parameter is used to define the plane of the circle and the angle of the arc as shown in the figure below.



If the three points are aligned or two of them coincide, issuing the MOVEC command will cause a straight-line motion instead of creating a circle or arc.
With this syntax, the orientation of location1 is not used.
If angle is specified, then the robot will move by angle degrees and not necessarily finish at location2. The angle has higher priority than location2 in defining the final position.
When angle is specified, the orientation of location2 is ignored. The final orientation is determined entirely by the turn parameter. If the turn parameter is omitted or 0, then the final orientation will be the orientation of the start position. If the turn parameter is non-zero, then the final orientation is the start position rotated by angle around the axis of the arc.
When turn is non-zero, the MOVEC command will generate an *invalid orientation* error for 4-axis robots when the plane of the circle is not parallel to the X-Y plane of the Tool center point.
As with straight-line motion, circular motion is compatible with multi-turn rotation. If location2 is a precision point, the multi-turn joint can rotate more than360 degrees.

● MOVEC(angle,turn) center

This MOVEC program command syntax is designed to create a circle / arc path that starts from the current robot position or in case of continuous path motion, the current robot destination. The path is centered around the center location. The end location is specified with angle degrees.
The plane of the circle is defined as the plane passing through the start position and parallel to the X-Y plane of the center location. If the Z-orientation of the center location is not perpendicular to the straight line passing through center and the start position, then the center of the circle is not the location center. It is the intersection of the Z-axis of center and a plane that is perpendicular to this axis and passes through the start position.
After the center of the circle is defined, the radius of the circle is the distance from the start position to the center.



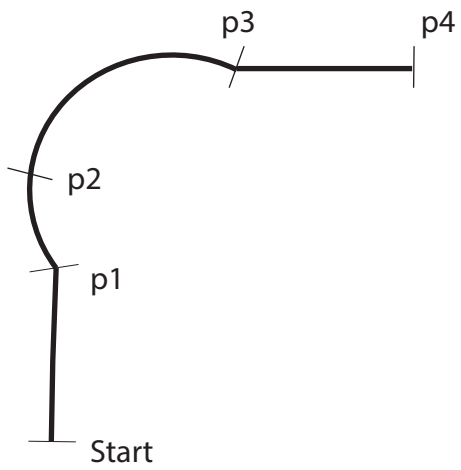
Example

The following example shows MOVEC used with transformations.

```
MOVEC loc1, loc2
```

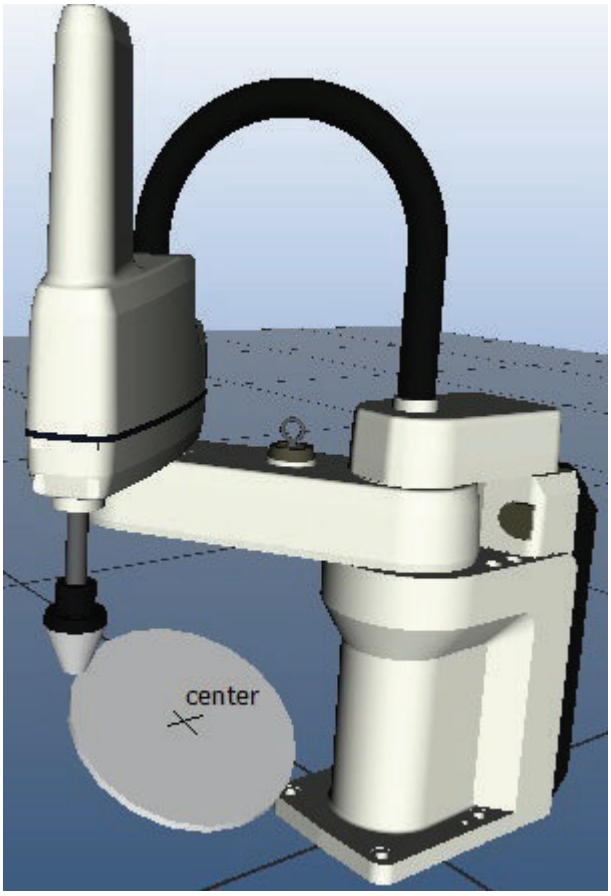
The following example shows MOVEC used with continuous path motion as shown in the figure below.

```
MOVES p1
MOVEC p2, p3
MOVES p4
BREAK
```



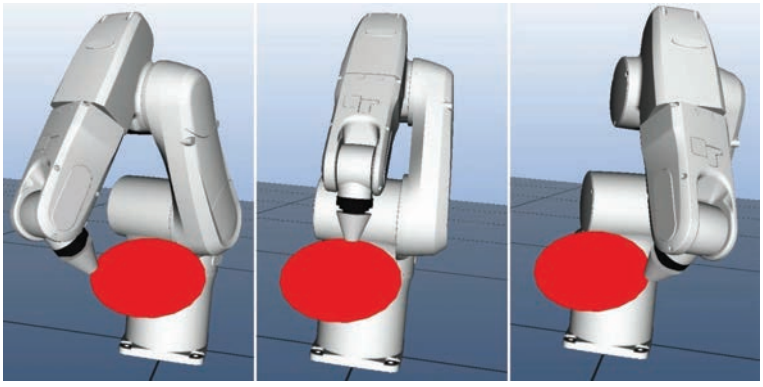
The following example shows MOVEC used to create a full circle with a SCARA robot as shown in the figure below.

```
SET center = TRANS(300,0,210,90,30,0)
SET start_pos = TRANS(420, 0, 210, 0, 180, 0)


MOVES start_pos
BREAK

MOVEC(360) center
BREAK
```



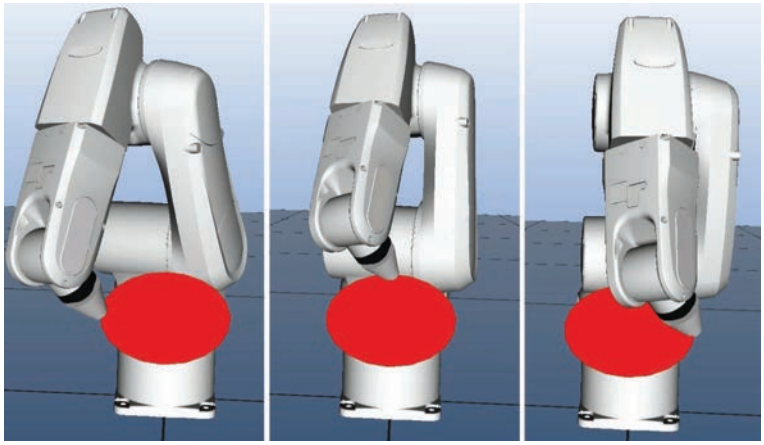
In the previous figure, "MOVEC(360,1) center" will return an *invalid orientation* error because the robot cannot maintain a constant orientation relative to a non-horizontal circle.
The following example shows MOVEC used to create a half circle with rotating orientation for dispensing with a 6-axis robot.

```
SET center = TRANS(300,0,250,0,-150,0)
SET start_pos = center:TRANS(0,0,0,-90):TRANS(100,,,,-30)


MOVES start_pos
BREAK

MOVEC(180,TRUE) center
BREAK
```



Changing the last 3 lines can modify the previous example to create the same motion without rotating the orientation as shown below.

```
MOVEC(180) center
BREAK
```



Related Keywords

3-4-9 APPRO on page 3-277
3-4-34 DEPART on page 3-316
3-4-81 MOVE on page 3-384
3-4-83 MOVES on page 3-390
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-83 MOVES

Initiate a robot motion to the position and orientation described by the given location with straight-line motion.

Syntax

```
MOVES location
```

Usage Considerations

MOVES causes a straight-line motion, during which no changes in configuration are permitted.
This command can be executed by any program task as long as the task has attached a robot. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing this command causes an error.

Parameters

Parameter	Description
location	Transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move.

Details

The MOVES program command 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.

Example

The following example moves along a straight-line path to the location described by the compound transformation "ref:place".

```
MOVES ref:place
```

Related Keywords

3-4-9 APPRO on page 3-277
3-4-10 APPROS on page 3-278
3-4-34 DEPART on page 3-316
3-4-35 DEPARTS on page 3-317
3-4-82 MOVEC on page 3-385
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-84 NEXT

Branch to the END statement of the nth nested loop, perform the loop test, and loop if appropriate.

Syntax

```
NEXT count
```

Usage Considerations

This command can be used with the FOR, WHILE, and DO control structures.

Parameters

Parameter	Description
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 command 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:

```
FOR i = 1 to 20
    FOR j = 1 to 10
        FOR k = 10 to 50
            IF error == 1 THEN
                NEXT
            END
            IF error == 2 THEN
                NEXT 3
            END
        END
    END
END
```

Related Keywords

3-4-38 DO on page 3-321
3-4-44 END on page 3-328
3-4-47 EXIT on page 3-334
3-4-61 FOR on page 3-358
3-4-138 WHILE on page 3-463

3-4-85 NOFLIP

Request a change in the robot configuration during the next motion so the pitch angle of the robot wrist has a positive value.

Syntax

NOFLIP

Usage Considerations

Configuration changes cannot be made during straight-line motions.
If the selected robot does not support a no-flip configuration, this command is ignored by the robot.
The NOFLIP program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. If the robot is not attached, this command has no effect.
This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the NOFLIP command causes an error.
Refer to the *3-4-55 FLIP* on page 3-346 program command for more information.

Example

The following example moves the robot to "point1" with the NOFLIP configuration.

```
NOFLIP
MOVE point1
```

Related Keywords

3-1-18 CONFIG on page 3-24
3-4-55 FLIP on page 3-346
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)

3-4-86 NONULL

Instruct the V+ system to not wait for position errors to be nulled at the end of continuous-path motions.

Syntax

```
NONULL ALWAYS
```

Usage Considerations

Only the next robot motion is affected if the ALWAYS parameter is not specified.
NULL ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins. This is the default state of the V+ system.
The NONULL program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the NONULL command causes an error.
Executing this keyword will have no difference in operation when used in emulation mode. The trajectory of the robot in the emulation mode is considered to be the same as the current target position. Therefore, unlike with an actual robot, no positioning error occurs when the operation is completed.

Parameters

Parameter	Description
ALWAYS	Optional qualifier that establishes NONULL as the default condition. When ALWAYS is included in a NONULL command, NONULL remains in effect continuously until disabled by a NULL program command. If ALWAYS is not specified,the NONULL command applies only to the next robot motion.

Details

When NONULL is active and a break in the robot motion occurs, V+ does not wait for the servos to signal that all moving joints have reached their specified positions before it begins the next motion. At the end of the allotted time, V+ assumes that all joints have 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. Position-error checking is not active while NONULL is active and large position errors can occur.

Example

The following example issues the NONULL operation for a single motion to location "point1".

```
NONULL
MOVE point1
```

The following example issues the NONULL operation for a series of motions to location "point2" and "point3".

```
NONULL ALWAYS
MOVE point2
MOVE point3
```

Related Keywords

3-4-26 COARSE on page 3-305
3-1-18 CONFIG on page 3-24
3-6-4 DELAY.IN.TOL on page 3-476
3-4-54 FINE on page 3-344
3-4-88 NULL on page 3-396
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)

3-4-87 NOOVERLAP

Generate a program error if a subsequent motion is planned that causes a selected multi-turn axis or platform to move more than 180 degrees in any direction to avoid a limit stop.

Syntax

NOOVERLAP ALWAYS

Usage Considerations

NOOVERLAP applies to the operation of the following robots and their respective joints or platform.

- Joints 1, 4, and the final joint for 6-axis robots
- Joint 4 for SCARA robots
- Joint 4 for iX3 robots
- Platforms capable of more than 180 degree rotation for iX4 robots

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

The NOOVERLAP program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.

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

Parameters

Parameter	Description
ALWAYS	Optional qualifier that establishes NOOVERLAP as the default condition. If ALWAYS is specified, NOOVERLAP remains active continuously until disabled by an OVERLAP program command. If ALWAYS is not specified, the NOOVERLAP command applies only to the next robot motion.

Details

When NOOVERLAP is active and the transformation destination of a joint-interpolated or straight-line motion requires that a multiple-turn axis or platform rotates more than 180 degrees in any direction to avoid a limit stop, a program error will occur and the motion will not be performed. If the destination is specified as a precision point, this check is not performed.

Given a transformation destination, a multiple-turn axis normally attempts to move to a new position by moving in the direction that requires less than 180 degrees of motion. The only conditions that force an axis to make a larger change are if SINGLE program command is issued or if a software limit stop would be violated.

When NOOVERLAP is active, the setting of SINGLE mode is ignored.

As with other user program errors, the error condition generated as a result of the NOOVERLAP check can be detected by a standard REACTE subroutine if desired.

Related Keywords

3-4-89 OVERLAP on page 3-397

3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-88 NULL

Instruct the V+ system to wait for position errors to be nulled at the end of continuous path motions.

Syntax

```
NULL ALWAYS
```

Usage Considerations

Only the next robot motion is affected if the ALWAYS parameter is not specified.
NULL ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins. This is the default state of the V+ system.
The NULL program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the NULL command causes an error.
Executing this keyword will have no difference in operation when used in emulation mode. The trajectory of the robot in the emulation mode is considered to be the same as the current target position. Therefore, unlike with an actual robot, no positioning error occurs when the operation is completed.

Parameters

Parameter	Description
ALWAYS	Optional qualifier that establishes NULL as the default condition. If ALWAYS is specified, NULL remains in effect continuously until disabled by a NONULL program command. If ALWAYS is not specified, the NULL command applies only to the next robot motion.

Details

When NULL is active and a break in the robot motion occurs, V+ waits for the servos to signal that all moving joints have reached their specified positions before it begins the next motion. Position accuracy is determined by the COARSE and FINE program commands.

Example

The following example issues the NULL operation for a single motion to location "point1".

```
NULL
MOVE point1
```

Related Keywords

3-4-26 COARSE on page 3-305
3-1-18 CONFIG on page 3-24
3-6-4 DELAY.IN.TOL on page 3-476
3-4-54 FINE on page 3-344
3-4-86 NONULL on page 3-393
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)

3-4-89 OVERLAP

Disable the NOOVERLAP limit-error checking either for the next motion or for all subsequent motions.

Syntax

```
OVERLAP ALWAYS
```

Usage Considerations

OVERLAP applies to the operation of the following robots and their respective joints or platform.

- Joints 1, 4, and the final joint for 6-axis robots
- Joint 4 for SCARA robots
- Joint 4 for iX3 robots
- Platforms capable of more than 180 degree rotation for iX4 robots

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

The OVERLAP program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.

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

Parameters

Parameter	Description
ALWAYS	Optional qualifier that establishes NOOVERLAP as the default condition. If ALWAYS is specified, NOOVERLAP remains active continuously until disabled by an OVERLAP program command. If ALWAYS is not specified, the NOOVERLAP command applies only to the next robot motion.

Details

When OVERLAP is active, the settings of SINGLE program command affects the robot motion if it is applicable to the robot type.
When OVERLAP is active and the transformation destination of a joint-interpolated or straight-line motion requires that a multiple-turn axis or platform rotates more than 180 degrees in any direction, the motion is executed without generating a program error.
OVERLAP disables the limit-error checking of NOOVERLAP. The OVERLAP set- ting is applied whenever program execution is initiated and when a new execution cycle begins.

Related Keywords

3-4-87 NOOVERLAP on page 3-394
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-121 SINGLE on page 3-438

3-4-90 PACK

Replace a substring within an array of (128-character) string variables, or within a (non-array) string variable.

Syntax

```
PACK string_array[index], first_char, num_chars = string
PACK string_var, first_char, num_chars = string
```

Parameters

Parameter	Description
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.

Parameter	Description
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 follows 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 command 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 sub- string whose first character is to replace the 127th character in element [3] supersedes 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 is created and filled with ASCII NUL characters (^H00) up to the specified start of the sub- string. Similarly, if the array element to be modified is too short, the string is padded with ASCII NUL characters to the start of the substring.
In order to efficiently access the string array, this command 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. 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 array element. An error results if the operation causes the string to be longer than 128 characters.

Example

The following example 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 replaces the second through 12th characters of "$list[4]".

```
PACK $list[3], 130, 11 = $string
```

Related Keywords

3-1-66 $MID on page 3-93
3-1-121 $UNPACK on page 3-155

3-4-91 PANIC

Simulate an external E-Stop button press to stop all robots immediately, but do not turn off high power.

Syntax

PANIC robot_num

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Usage Considerations

If no robot is specified with the robot_num parameter, all robots are stopped. This command has no effect on non-robot systems.

Details

This command performs the following actions:

- Immediately stops robot motion.
- Stops execution of the robot control program if the robot is attached and no REACTE has been executed to enable program processing of error.

Unlike pressing the emergency stop button on the pendant, high power is left turned ON after a PANIC operation is processed.

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-4-45 ESTOP on page 3-330 (program command)
3-2-21 ESTOP on page 3-194 (monitor command)
3-2-44 PANIC on page 3-228 (monitor command)

3-4-92 PARAMETER

Set the value of a system parameter.

Syntax

```
PARAMETER parameter_name = value
PARAMETER parameter_name[index] = value
```

Parameters

Parameter	Description
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

The PARAMETER program command 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.

Additional Information

A regular assignment statement cannot be used to set the value of a system parameter.
Refer to *V+ User's Manual (Cat. No. I671)* for more information about system parameters.

Example

The following example will set the BELT.MODE parameter to have bits 1 and 3 turned ON (mask values 1 + 4).

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

Related Keywords

3-5-1 BELT.MODE on page 3-468
3-5-4 NOT.CALIBRATED on page 3-471
3-2-45 PARAMETER on page 3-229 (monitor command)

3-4-93 PAUSE

Stop program execution but allow the program to be resumed.

Syntax

```
PAUSE
```

Usage Considerations

Unlike HALT and STOP, the PAUSE program command does not force FCLOSE or DETACH operations on the disk 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 program command with the appropriate task number to close all files and detach all logical units.

Details

The PAUSE program command causes a break and terminates execution of the application program, displaying the message (PAUSED). Execution can subsequently be continued issuing a PROCEED monitor command with the appropriate task number.
When debugging a program, a PAUSE command can be inserted to stop program execution temporarily while the values of variables are checked.
Any robot motion in progress when a PAUSE command is processed completes normally.

Related Keywords

3-4-66 HALT on page 3-365
3-2-34 KILL on page 3-215 (monitor command)
3-4-74 KILL on page 3-375 (program command)
3-2-48 PROCEED on page 3-233
3-4-125 STOP on page 3-448

3-4-94 PAYLOAD

Adjust the dynamic compensation for the selected robot based on the payload's physical properties.

Syntax

```
PAYLOAD mass, inertia[], center_mass_tool
```

Usage Considerations

This keyword affects the robot that is currently selected with a SELECT program command.
A task does not need to have a robot attached to issue this command. The robot cannot be attached by another task other than the one in which the PAYLOAD command is used.
This command is currently supported by the i4L and i4H series robots. Using this command with other robot types will have no effect.
The PAYLOAD command takes effect immediately and is not synchronized with robot-motion segments.
The value for mass and inertia are internally limited to the maximum payload and inertia for a given robot to protect the mechanism.
If the PAYLOAD command 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 overshoot any time the commanded

acceleration is nonzero. The degree of overshoot depends upon the degree to which the payload is in error, and the size of the acceleration and deceleration during the motion being attempted.
Executing this command with no parameters specified will reset the robot pay- load parameters to the default settings.

Details

Use this keyword to reduce settling time and motion path deviations. This can result in decreased cycle time with increased precision during motion and positioning.

Parameters

Parameter	Description
mass	Optional real-valued expression interpreted as a mass in kg. A negative or omitted value restores the default payload for the robot.
inertia[]	Optional array of real-valued expression where inertia[0] specifies the moment of inertia around the X axis, inertia[1] around Y and inertia[2] around Z in units of kg.m². A negative or omitted value restores the default payload for this robot. center_mass_tool. Optional transformation that specifies the position of the center of mass of the pay- load relative to the robot tool flange. If the parameter is omitted, the offset is reset to NULL.

Example

The following example specifies a 5 kg payload.

```
PAYLOAD 5
```

The following example resets the payload to its default parameters.

```
PAYLOAD
```

3-4-95 PDNT.CLEAR

Creates a pendant notification.

Syntax

PDNT.CLEAR

Usage Considerations

The target pendant will be the one connected to the currently selected robot.
Refer to *V+ User's Manual (Cat. No. I671)* for information about T20 pendant programming.

Example

The following example will clear the screen of the T20 pendant that is connected to the currently selected robot.

```
PDNT.CLEAR
```

Related Keywords

3-4-96 PDNT.NOTIFY on page 3-404
3-4-97 PDNT.WRITE on page 3-405

3-4-96 PDNT.NOTIFY

Creates a pendant notification.

Syntax

```
PDNT.NOTIFY $title, $msg
```

Usage Considerations

The pendant does not need to be attached using an ATTACH command prior to using this operation.
The target pendant will be the one connected to the currently selected robot.
Refer to *V+ User's Manual (Cat. No. I671)* for information about T20 pendant programming.

Parameters

Parameter	Description
$title	Optional string constant, variable, or expression that contains the title of the pendant notification.
$msg	Optional string constant, variable, or expression that contains the message of pendant notification.

Details

PDNT.NOTIFY is used to create a simple notification box on the T20 Pendant screen that can be cleared by pressing the OK or Cancel buttons on the pendant or with a PDNT.CLEAR operation.

Example

The following example will create a notification box with a title and message on the T20 pendant that is connected to the currently selected robot.

```
PDNT.NOTIFY "Manual Mode", "To enable power, press and hold the enable switch."
```

Related Keywords

3-4-95 PDNT.CLEAR on page 3-403
3-4-97 PDNT.WRITE on page 3-405

3-4-97 PDNT.WRITE

Sets the pendant's Custom Message screen.

Syntax

```
PDNT.WRITE (msgsize) $title, $msg, $f1, $f2, $f3, $f4
```

Usage Considerations

The pendant does not need to be attached using an ATTACH command prior to using this operation.
The target pendant will be the one connected to the currently selected robot.
Refer to *V+ User's Manual (Cat. No. I671)* for information about T20 pendant programming.

Parameters

Parameter	Description
msgsize	Optional Real value, variable, or expression whose value represents the array size of $msg.
$title	Optional string constant, variable, or expression that contains the title of the pendant's Custom Message screen.
$msg	Optional string constant, variable, or expression that contains the body of the pendant's Custom Message screen. This can accept html tags to create an html-formatted text box. If $msg is an array and msgsize > 1, it will con- catenate all the elements of the array.
$f1	Optional string constant, variable, or expression that contains the label of the F1 Key of the Custom Mes-sage Screen.
$f2	Optional string constant, variable, or expression that contains the label of the F2 Key of the Custom Mes-sage Screen.
$f3	Optional string constant, variable, or expression that contains the label of the F3 Key of the Custom Mes-sage Screen.
$f4	Optional string constant, variable, or expression that contains the label of the F4 Key of the Custom Mes-sage Screen.

Details

PDNT.WRITE is used to set the screen of the T20 Pendant. This is used to create a user interface to program the pendant through V+. The screen can be either updated with a subsequent PDNT.WRITE or cleared with a PDNT.CLEAR. While the screen is displayed, all green keys, as well as Select Robot, are active, so that the robot can always be jogged. All other keys have no effect.

Example

The following example will create a screen with specified text for the title, message, and function keys for the T20 pendant that is connected to the currently selected robot.

```
$p.title = "Operator Control"
$p.msg[0] = "Select Options from buttons below"
$p.f[1] = "Apps"
$p.f[2] = "Status"
$p.f[3] = ""
$p.f[4] = ""
PDNT.WRITE $p.title, $p.msg[], $p.f[1], $p.f[2], $p.f[3], $p.f[4]
```

Related Keywords

3-4-73 KEYMODE on page 3-373
3-4-98 PENDANT on page 3-406
3-4-95 PDNT.CLEAR on page 3-403
3-4-96 PDNT.NOTIFY on page 3-404

3-4-98 PENDANT

Return input from the manual control pendant.

Syntax

PENDANT (select) key

Usage Considerations

The pendant must be attached using an ATTACH operation prior to using this function.
Refer to *V+ User's Manual (Cat. No. I671)* for information about T20 pendant programming.
This keyword involves communication with ACE. Using this keyword as an argument for another function can cause unnecessary delays or consume unnecessary processing power. Avoid using this keyword more often than necessary.

Parameters

Parameter	Description
select	Real-valued expression whose value selects what type of pendant information is returned.
key	Return value from the pendant depending on select.

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 moment the operation is called. The state of the key depends upon the key mode setting for that key. Refer to the *3-4-73 KEYMODE* on page 3-373 program command for information about setting key modes. 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 and that the key is pressed. The value OFF (0) indicates 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.

● 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.

● select = -2

Returns the current value from the speed label, in the range of 0 to 100 (decimal). When the Pendant Jog mode is COMP, the monitor speed is returned. In other jog modes, the jog speed is returned. The pendant logical unit does not need to be attached for this mode of operation.

● select = -3

Returns the current display screen active on the pendant. 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	Home screen
2	Other screens
3	Error screen
4	USER (custom) screen

- **select = -4**

  Returns the version number of the manual control software. This is the same as the value returned by the real-valued function ID(1,2). The value -1 is returned if the pendant is not connected to the system. The pendant logical unit does not need to be attached for this mode of operation.

Example

The following example sets the manual control soft keys to keyboard mode, and then waits for one of them to be pressed.

```
ATTACH (1)
KEYMODE 1,5 = 0
key = PENDANT(0)
TYPE "Soft key #", key, " pressed"
DETACH (1)
```

The following example sets the DONE key to level mode and loops until the key is pressed.

```
ATTACH (1)
KEYMODE 8 = 2
WAIT PENDANT(8)
DETACH (1)
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-73 KEYMODE on page 3-373

3-4-99 POSE.TO.TRANS

Returns a transformation given an array determined by translation and rotation in extrinsic Euler XYZ.

Syntax

```
POSE.TO.TRANS trans = array_name[index]
```

Usage Considerations

Intrinsic refers to the internal perspective of an object's movement or rotation. When an angle changes, all internal axes of the object rotate simultaneously.
Extrinsic refers to the external viewpoint of an object's movement or rotation. From a global perspective, when an axis rotates, subsequent rotations adhere to the reference convention of the external coordinate system.

Parameters

Parameter	Description
trans	Transformation variable or transformation array element in which the result is stored. If the variable does not exist it will be created.
array_name	Real-valued array that contains the translation and rotation in Euler XYZ for conversion into a transformation. The first specified element of the array must contain the value for x-coordinate. The second element must contain the value for y-coordinate. The third element must contain the value for z-coordinate. The fourth element must contain the value for x-axis rotation, The fifth element must contain the value for y-axis rotation. The sixth element must contain the value for z-axis rotation.
index	Optional integer value that identifies the array element that contains the position for x-coordinate. If no index is specified, element 0 must contain the position for x-coordinate.

Details

This program command assigns a transformation corresponding to the conversion of the specified values to consecutive elements of the named array. The transformation is determined through ZYZ intrinsic Euler axis rotations, with a structured representation consisting of six elements corresponding to X, Y, Z, yaw, pitch, and roll.

The last three elements of the array represent the rotation and they are applied in the following order, as specified by the extrinsic Euler XYZ angles.

- Rotation around the x-axis.
- Rotation around the original y-axis
- Rotation around the original z-axis.

As specified by the intrinsic Euler ZYZ angles, the rotation is applied in the order that follows below.

- Rotation around the z1-axis.
- Rotation around the new y-axis.
- Rotation around the new z2-axis

Example

The following example assigns the conversion obtained from elements 0 to 5 of the array “position” to the transformation “trans”.

```
position[0] = 20
position[1] = 30
```

```
position[2] = 50
position[3] = 5
position[4] = 1
position[5] = 3
POSE.TO.TRANS trans = position[0]
```

Related Keywords

3-1-112 TRANS on page 3-144
3-4-129 TRANS.TO.POSE on page 3-452

3-4-100 PROCEED

Resume execution of an application program.

Syntax

PROCEED task

Usage Considerations

A program cannot resume if it has completed execution normally or has stopped due to a HALT program command.

Parameters

Parameter	Description
task	Real value, variable, or expression interpreted as an integer that specifies which program task is to be executed.If no task number is specified, task number 0 is assumed.

Details

The PROCEED program command resumes execution of the specified program task at the step following the one where execution was halted due to a PAUSE program command, an ABORT program command, a breakpoint, single-step execution, or a runtime error.
In addition to continuing execution of a suspended program, this command can be used to initiate execution of a program that has been prepared for execution with the PRIME monitor command.
If the specified task is executing and the program is at a WAIT or WAIT.EVENT statement (for example, waiting for an external signal condition to be satisfied), issuing a PROCEED command has the effect of skipping the WAIT or WAIT.EVENT statement.
This command has no effect if the specified task is executing and the program is not at a WAIT or WAIT.EVENT statement.

PROCEED differs from RETRY command. If a program statement generated an error, RETRY command attempts to re-execute that statement. The PROCEED command resumes execution at the next program statement. If a robot motion was in progress when the program stopped, RETRY attempts to complete that motion but PROCEED will advance to the next motion.

Example

The following example will resume execution of task 2 if it is stopped due to a PAUSE or breakpoint operation.

```
IF (TASK(1,2)==5)THEN
    PROCEED 2
END
```

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-47 PRIME on page 3-232
3-2-48 PROCEED on page 3-233
3-2-52 RETRY on page 3-236 (monitor command)
3-4-110 RETRY on page 3-426 (program command)
3-2-58 STATUS on page 3-244
3-2-56 SSTEP on page 3-241
3-2-70 XSTEP on page 3-261

3-4-101 .PROGRAM

Define the arguments that are passed to a program when it is invoked.

Syntax

```
.PROGRAM program_name(argument_list) ;comment
```

Usage Considerations

This command is inserted automatically by the V+ editors when a new program is edited.
This special command must be the first line of every program and cannot be omitted from a program.

Additional Information

Refer to *V+ User's Manual (Cat. No. I671)* for more information about passing arguments and variables to and from V+ programs.

Parameters

Parameter	Description
program_name	Name of the program in which this keyword 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 is displayed when the program is loaded from a disk file and when the DIRECTORY monitor command is processed. The semicolon [;] should be omitted if no comment is included.

Details

The V+ editor automatically enters a .PROGRAM line when you edit a new pro- gram. They also prevent you from deleting the line or changing the program name. You can 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 are considered automatic variables for the named program. Refer to the *3-4-12 AUTO* on page 3-283 keyword for more information.

When a program begins execution (for example, with an EXECUTE keyword or a CALL program command), the arguments in the .PROGRAM command are associated with those in the EXECUTE or CALL keywords. This association allows values to be passed between a program and its caller.

Refer to the description of the *3-4-21 CALL* on page 3-297 program command 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 omitted when the program executes.

- Real-valued scalar parameters can be assigned a value within a program if they are omitted.
- Location, string, and belt (scalar or array) parameters, and real-valued array parameters, cannot be assigned a value within a program if they are omitted. AUTO variables can be used to bypass this restriction, as shown in the example below. Undefined parameters can be passed as program arguments and then be assigned a value.

  If a program attempts to assign a value to one of these omitted variables, the error *Undefined value* results. In that case, the error refers to the variable on the left side of the assignment statement.
- If an undefined or omitted parameter is passed to another program through a subsequent CALL program command and the type of the variable is ambiguous (i.e., the type could be real-valued or location), the parameter is assumed to be real-valued.
- Elements of an omitted array parameter cannot be passed by reference in a subsequent CALL program command.

The DEFINED real-valued function can be used within a program to check whether a program parameter is defined (meaning both passed as a argument and as an argument that has been assigned a value previously). The example below shows how a program can be written to accommodate undefined or omitted parameters.

A comment can be included on the .PROGRAM line, which is displayed when the program is loaded from the disk and by the DIRECTORY monitor command.

Example

The following example defines a program that expects no arguments to be passed to it.

```
.PROGRAM get()
```

The following example defines a program that expects a string-valued argument and either a location or real-valued argument. The type of the second argument is determined by its use in the program.

```
.PROGRAM test($n, dx)
```

The following example shows how a program can be written to manage undefined or omitted parameters. The example shows part of the program example, which has a real-valued parameter and a string parameter.

```
.PROGRAM example(real, $string)
    AUTO $internal.var
    IF NOT DEFINED(real) THEN
        real = 1
    END
    IF DEFINED($string) THEN
        $internal.var = $string
    ELSE
        $internal.var = "default"
    END
.END
```

Related Keywords

3-4-21 CALL on page 3-297
3-4-22 CALLS on page 3-300
3-2-22 EXECUTE on page 3-195 (monitor command)
3-4-46 EXECUTE on page 3-331 (program command)
3-2-47 PRIME on page 3-232
3-2-56 SSTEP on page 3-241
3-2-70 XSTEP on page 3-261

3-4-102 PROMPT

Display a string on the Monitor Window and wait for operator input.

Syntax

PROMPT output_string, **variable_list**

Usage Considerations

Do not use the PROMPT keyword with a Monitor Command program or when access to the Monitor Window is not available.
Do not use an External array variable for the variable_list parameter.

Parameters

Parameter	Description
output_string	Optional string expression that is output to the Monitor Window. The cursor is left at the end of the string.
variable_list	A list of real-valued variables or a single string variable that receives the data. External array variables are not supported.

Details

Displays the text of the output string on the Monitor Window and waits for you to type in a line terminated by pressing the return key.

The input line can be processed in either of two ways as described below.

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. Each PROMPT command should request only one value 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 appropriately.

If you press the return key or press CTRL+C, an empty line is read. This results in all the real variables being set to zero or the string variable being assigned an empty string.

If you press CTRL+Z, an end-of-file error condition results. If there is no REACTE command active, program execution is terminated and an error message is displayed. For this reason, CTRL+Z can be a useful way to abort program execution at a PROMPT.

Example

The following example will examine the statement below.

```
PROMPT "Enter the number of parts: ", part.count
```

The following message will be displayed in the Monitor Window.

```
Enter the number of parts:
```

After you type a number and press the return key, the variable "part.count" is set equal to the value typed and program execution resumes.

The following example will examine the statement below.

```
PROMPT "Enter the number of parts: ", $input
```

If you enter characters that are not valid for numeric input, V+ does 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 command as shown below.

```
PROMPT $ENCODE(/B,/C1,/X10)+"Enter the number of parts: ",
$input
```

The statement above will beep the terminal, space down a line, space over ten spaces, output the string, and wait for your input. A + sign must be used between the $ENCODE function and the quoted string because the entire output_string parameter must be a single string expression.
The following statements are equivalent to the previous example.

```
TYPE /B, /C1, /X10, /S
PROMPT "Enter the number of parts: ", $input
TYPE /B, /C1, /X10, "Enter the number of parts: ", /S
PROMPT , $input
```

/S must be included in the TYPE command as shown to have the prompt string output on one line and to have the cursor remain on that line.

Related Keywords

3-1-46 GETC on page 3-65
3-4-106 READ on page 3-421
3-4-130 TYPE on page 3-453

3-4-103 REACT

Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal transitions.

Syntax

```
REACT signal_num, program, priority
```

Usage Considerations

The REACT program command can be executed by any of the program tasks. Each task can have its own independent REACT definition.
The following signals can be used with the REACT program command.

Signal Type	Range	Details
Digital Input	1001 to 1999	Signals are only available if they physically exist in the system.
Software	2001 to 2999	All signals are available
Host	4001 to 4999	

Reactions are triggered by signal transitions and not levels. If a signal is going to be monitored for a transition from OFF to ON and the signal is already ON when a REACT command is executed, the reaction does not occur until the signal goes OFF and then ON again.
A signal must remain in its state for at least 1 cycle plus an additional 2 ms to assure detection of a transition.
If software signals are being used to trigger reactions, the WAIT program command with no argument should be used as required to ensure that the signal state remains constant for the required time period.

The requested signal monitoring is enabled as soon as a REACT command is executed. This can affect the motion initiated by a motion statement preceding the REACT command in the program.

Parameters

Parameter	Description
signal_num	Real-valued expression representing the signal to be monitored.The signal number must be within the ranges specified above. The software signals can be used by one program task to interrupt another task. If signal_num is positive, V+ reacts to the transition from OFF to ON. If signal_num is negative, V+ reacts to a transition from ON to OFF.
program	Real-valued expression representing the signal to be monitored.The signal number must be within the ranges specified above. The software signals can be used by one program task to interrupt another task. If signal_num is positive, V+ reacts to the transition from OFF to ON. If signal_num is negative, V+ reacts to a transition from ON to OFF.
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. Refer to the *3-4-77 LOCK* on page 3-379 program command 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 command against the program priority setting at that time. The program priority is always set to 0 when execution begins and can be changed with the LOCK program command. If the REACT priority is greater than the program priority, the normal program execution sequence is interrupted and the equivalent of a CALL operation is executed.

The program priority is temporarily raised to the REACT priority, locking out any reactions of equal or lower importance. When a RETURN program command 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 does not occur until the program priority is lowered. Depending upon the relative priorities, there can be a considerable delay between the time a signal transition is detected by V+ and the time the reaction program is invoked.

If multiple reactions are pending because of a priority lockout, the reaction with the highest priority is serviced first when the locking priority is lowered. If multiple pending reactions have the same priority, the one associated with the highest signal number is processed first.

The subroutine call to a program is performed such that when a RETURN pro- gram command is encountered, the next statement to be executed is the one that follows the last statement processed before the reaction program was initiated. If there is a sequence of statements that you do not want interrupted by a reaction program, you should use the LOCK program command to raise the program priority during that sequence.
The signal monitoring continues until one of the following occurs.

- An IGNORE program command is executed for the signal.
- A reaction occurs in which case the IGNORE signal_num is automatically performed.
- A REACT program command is executed that refers to the same signal. If the signal specified in a REACT command is already being monitored by a previous REACT or REACTI command, the old command is canceled when the new REACT command is executed.

Example

The following example monitors the digital input signal 1001 for an OFF to ON transition. When this occurs, the program designated as "trap1" will be invoked with a priority level of 10.

```
REACT 1001, trap1, 10
```

Related Keywords

3-4-70 IGNORE on page 3-369
3-4-77 LOCK on page 3-379
3-1-82 PRIORITY on page 3-105
3-4-104 REACTE on page 3-417
3-4-105 REACTI on page 3-419
3-1-91 SIG.INS on page 3-113

3-4-104 REACTE

Initiate the monitoring of system messages that occur during execution of the current program task.

Syntax

```
REACTE program_name
```

Usage Considerations

The REACTE program command can be executed by any of the program tasks.
Each task can have its own independent REACTE statement. A task cannot directly detect system messages caused by another task, but tasks can signal other tasks with global variables or software signals.
Using the REACTE command for purposes other than pre-programmed error routines when an unexpected error occurs requires extreme caution and can cause unpredictable behavior.

Parameters

Parameter	Description
program_name	Optional name of the program that is to be called when a system message occurs. If no program is specified, the previous REACTE operation is canceled and any pending system messages are discarded.

Details

The main purpose for the REACTE program command is to allow for pre-programmed error routines to execute when an unexpected system message occurs. If a robot hardware error occurs, a program can be executed to set external output signal lines to shut down external equipment, for example.
If a system message occurs after a REACTE command has been executed, the specified program is invoked rather than stopping normal program execution. The program is invoked with similar functionality to the CALL program command.
The ERROR function can be used within the error-handling program to determine what system message caused the program to be invoked.
The following considerations should be made when using the REACTE command.

- The program priority is raised to 254 when the error-handling program is invoked, locking out all reaction programs.
- Execution of the program task stops if an error occurs while the system is processing a previous system message.
- The user program stack must have enough available memory for one more subroutine to execute. The error *Too many subroutine calls* cannot be processed. Refer to the *3-2-57 STACK* on page 3-242 monitor command for more information.
- The error-handling program can contain a RETURN statement. When it is executed, the program tries to re-execute the command that caused the system message. This may cause a looping condition if the system message continues to occur.
- Before the error-handling program is entered, a DETACH command for the robot (logical unit number 0) is executed. An ATTACH command must be executed for the robot before program control of the robot can resume.
- If a STOP, HALT, or PAUSE command is executed within the error-hand- ling program, the original system message is output unless the error-hand- ling program contains a REACTE command with no argument.
- Unlike REACT and REACTI commands, execution of the REACTE error- handling program is never deferred because of priority considerations.

Example

The following example initiates monitoring of system messages and the program "error.trap" is executed if any error should occur during execution of the current program task.

```
REACTE error.trap
```

Related Keywords

3-1-37 ERROR on page 3-48
3-4-103 REACT on page 3-415
3-4-105 REACTI on page 3-419
3-4-111 RETURN on page 3-427

3-4-105 REACTI

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.

Syntax

```
REACTI signal_num, program, priority
```

Usage Considerations

For most applications, the REACTI command should be used only in a robot control program.
When a REACTI triggers, the robot that is stopped is the robot selected by the task at the time of the trigger, regardless of which robot was selected at the time the REACTI command was executed.
Refer to the considerations listed for the *3-4-103 REACT* on page 3-415 program command. The following signals can be used with the REACTI program command.

Signal Type	Range	Details
Digital Input	1001 to 1999	Signals are only available if they physically exist in the system.
Software	2001 to 2999	All signals are available.
Host	4001 to 4999	

Parameters

Parameter	Description
signal_num	Real-valued expression representing the signal to be monitored. The signal number must be within the ranges specified above. The software signals can be used by a secondary program to interrupt the robot control program. If signal_num is positive, V+ reacts to a transition from OFF to ON. If signal is negative, V+ reacts to a transition from ON to OFF.
program	Optional name of the subroutine that is called when the signal transitions.

Parameter	Description
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. Refer to the *3-4-77 LOCK* on page 3-379 program command 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+ continues processing the reaction just as it would for a REACT program command. Refer to the description of the *3-4-103 REACT* on page 3-415 program command for a complete explanation of this processing.

When REACTI is used by a program task that is not controlling the robot, care must be exercised to make sure the robot control program does not nullify the intended effect of the reaction subroutine. If your application has one program task monitoring the signal and a different program task controlling the robot, make the following considerations when planning for processing of the reaction.

- The robot motion in process at the time of the reaction is stopped as if a BRAKE program command were executed, but execution of the robot control program is not directly affected.
- If a reaction subroutine is specified, that routine is executed by the task that is monitoring the reaction and not by the task controlling the robot.

The signal monitoring continues until one of the following occurs.

- An IGNORE program command is executed for the signal.
- A reaction occurs in which case IGNORE signal_num is automatically performed.
- A REACTI command is executed that refers to the same signal. If the signal specified in a REACTI command is already being monitored by a previous REACTI or REACT command, the old command is canceled when the new REACTI command is executed.

If you do not want the robot motion to stop until the reaction program is called, use a REACT program command and place a BRAKE statement in the reaction program.

Example

The following example initiates monitoring of external input signal 1001. The robot motion is stopped immediately if the signal ever changes from ON to OFF because the signal is specified as a negative value. A branch to program "alarm" occurs when the program priority falls below 10 if it is not already at or below that level.

```
REACTI -1001, alarm, 10
```

Related Keywords

3-1-37 ERROR on page 3-48
3-4-103 REACT on page 3-415
3-4-104 REACTE on page 3-417
3-4-111 RETURN on page 3-427

3-4-106 READ

Read a record from an open file or from an attached device that is not file oriented. For an network device, read a string from an attached and open TCP connection.

Syntax

```
READ (lun, record_num, mode) var_list
```

Usage Considerations

The logical unit referenced by this command must have been attached previously.
For file-oriented devices, a file must already have been opened with an FOPEN_ keyword.
Do not use an External array variable for the var_list parameter.

Parameters

Parameter	Description
lun	Real-valued expression that identifies the device to be accessed. Refer to the *3-4-11 ATTACH* on page 3-279 program command 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. For nonfile-oriented devices or for sequential access of a file, this parameter should be 0 or omitted. Records are numbered from 1 to 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. The mode is used only for the Monitor Window logical unit. 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, mask value = 1): Wait(0) vs. No- wait (1) If this bit is OFF, program execution is suspended until the read operation is completed. If the bit is ON and the requested data is not available, pro- gram execution continues immediately and the IOSTAT function returns the error code for *No data received* (-526). • Bit 2 (mask value = 2):Echo (0) vs. No-echo (1) If this bit is OFF, input from the terminal is echoed back to the source. If the bit is ON, characters are not echoed back to the source.
var_list	Either a list of real-valued input variables or a list of string variables that receives the data. External array variables are not supported.

Details

The READ program command is a general purpose data input operation 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 command continues to read characters until it has read exactly the number of characters specified during the corresponding FOPEN_ keyword operation.

For variable-length record mode as with most devices, this command reads characters until the first carriage-return (CR) and linefeed (LF) character sequence or Ctrl+Z key input is encountered. For example, if you perform a variable-length record mode read from the disk, you receive all the characters until a CR and LF is 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 Monitor Window and the pendant 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. A READ operation from these devices is not complete until a CR and LF are received as input. For example, if you perform a READ operation from the Monitor Window, you receive all the characters until the RETURN key is pressed.

When a CR is received from the Monitor Window, V+ automatically adds a LF.

The GETC function can be used instead of the READ program command if you want to receive the CR and LF characters at the end of a record.

If bit 1 is ON in the mode parameter value, a read operation that is not complete does not cause the program to wait and returns immediately with the error *No data received* (error code -526). Then, additional READ commands 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 operation is complete.

Once a record has been read, it is processed in one of the following ways.

- 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 0. If data is read that is not a number, an error occurs and program execution stops (or an error reaction occurs).
- 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 are set to the empty string (""). If there is too much data for the number of string variables specified, an error is reported by the IOSTAT function.

When a READ operation is performed in variable-length record mode, the strings contain all the characters up to, but not including, the terminating CR and LF which are discarded.

Any error in the specification of this command such as attempting to read from an invalid unit causes a program error and halts program execution. Errors associated with performing the actual read operation such as end of file or device not ready do 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 to always check whether each read operation completed successfully by testing the value from the IOSTAT function.
When accessing a network device, the record_num parameter allows a server to communicate with multiple clients on a single logical unit. The parameter provides a handle number that you can be used to identify the client from which the READ data was received. Handles are allocated when a client connects to the server and are deallocated when the client disconnects. In order to determine when the client connection or disconnection is done, you must use the IOSTAT function after the READ operation.
The READ command with TCP/IP communication reads data until either the input string is full or the buffer is empty, at which point the command returns. The READ operation with TCP/IP does not allow fixed-length records and does not terminate when encountering a delimiter.

Example

The following example reads a line of text from the disk and stores the record in the string variable "$disk.input".

```
READ (5) $disk.input
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-56 FOPEN on page 3-348
3-4-57 FOPENA on page 3-349
3-4-58 FOPEND on page 3-352
3-4-59 FOPENR on page 3-354
3-4-60 FOPENW on page 3-356
3-4-62 FSEEK on page 3-360
3-1-46 GETC on page 3-65
3-1-58 IOSTAT on page 3-83
3-4-102 PROMPT on page 3-413

3-4-107 READY

Move the robot to the ready location.

Syntax

READY

Usage Considerations

Before executing this command, ensure that the robot will not collide with any objects while moving to the ready location.
Moving the robot to the ready location forces the robot into a standard con- figuration.

The READY program command can be executed by any program task as long as the task has attached a robot. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the READY command causes an error.

Details

The READY program command always succeeds, regardless of where the robot is located at the time the command is executed.
The following table lists the joint positions for the READY locations for various robots.

Additional Information

Refer to the i4L and i4H user manuals for joint position details when using those robot types.

Joint	eCobra 600	eCobra 800	Viper	iX3	iX4
1	-43.5°	-42.9°	0°	0°	0°
2	96.8°	93.4°	-90°	0°	0°
3	10.0mm	10.0mm	180°	0°	0°
4	53.8°	50.5°	0°	0°	0°
5	N/A	N/A	0°	N/A	N/A
6	N/A	N/A	0°	N/A	N/A

Example

The following example selects robot 1, attaches it to the current task, and then moves it to the ready position.

```
SELECT ROBOT=1
ATTACH
READY
```

Related Keywords

3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)

3-4-108 RELEASE

Allow the next available program task to run.

Syntax

RELEASE

Details

This command releases control to another task that is ready to run.
This command can be used in place of the WAIT command with no arguments in cases where other tasks must be given an opportunity to run, but a delay until the next trajectory cycle is not desired.

Related Keywords

3-4-136 WAIT on page 3-460
3-4-137 WAIT.EVENT on page 3-461

3-4-109 RESET

Turn OFF all external output signals.

Syntax

```
RESET
```

Usage Considerations

This program command has no effect on Host I/O (external) signal numbers 4001 to 4999.

⚠ DANGER

Before issuing this command, ensure all devices connected to the output signals can safely be turned OFF.

Details

The RESET program command is useful in the initialization portion of a pro- gram to ensure that all the external output signals are in a known state.

Example

The following example will turn OFF all external output signals if "reset.sig" is true.

```
IF reset.sig==TRUE
    RESET
END
```

Related Keywords

3-2-4 BITS on page 3-171 (monitor command)
3-4-16 BITS on page 3-290 (program command)

3-1-12 BITS on page 3-17 (real-valued function)
3-2-32 IO on page 3-211
3-2-50 RESET on page 3-235 (monitor command)
3-1-93 SIG on page 3-115
3-2-54 SIGNAL on page 3-238 (monitor command)
3-4-120 SIGNAL on page 3-437 (program command)

3-4-110 RETRY

Repeat execution of the last interrupted program command and continue execution of the program.

Syntax

```
RETRY task
```

Usage Considerations

The RETRY program command cannot be processed when the specified task is executing.
A program cannot be resumed if it has completed execution normally or has stopped due to a HALT program command.

Parameters

Parameter	Description
task	Real value, variable, or expression interpreted as an integer that specifies which program task is to be executed.If no task number is specified, task number 0 is assumed.

Details

The RETRY program command restarts execution of the specified task similar to the PROCEED program command. After a RETRY command, the point at which execution resumes depends on the status at the time execution was interrupted. If a program step or robot motion was interrupted before its completion, use of a RETRY command causes the interrupted operation to be completed before execution continues normally. This allows you to retry a step that has been aborted or that caused an error.
If no program step or robot motion was interrupted, the RETRY command has the same effect as the PROCEED program command.
When a RETRY command is used to resume an interrupted motion, all motion parameters are restored to the settings that were active before the motion was interrupted.

Example

The following example will resume from where the task number defined by "task.num" was interrupted (if the task is not executing or has not been stopped from the HALT operation).

```
IF TASK (task.num)<>4 THEN
    RETRY task.num
END
```

Related Keywords

3-4-100 PROCEED on page 3-410 (program command)
3-2-48 PROCEED on page 3-233 (monitor command)
3-2-56 SSTEP on page 3-241
3-2-58 STATUS on page 3-244
3-2-70 XSTEP on page 3-261

3-4-111 RETURN

Terminate execution of the current subroutine and resume execution of the suspended program at its next step.

Syntax

RETURN

Details

A RETURN program command in a main program has the same effect as a STOP program command. A RETURN operation is assumed when program execution reaches the last step of a subroutine. It is poor programming practice to omit the RETURN statement. At RETURN statement should be included as the last line of each subroutine.
In an error reaction subroutine, 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 is executed again. The error may occur again immediately. The RETURNE program command should be used in error reaction subroutines to avoid that situation.
If a RETURN command is used to exit from a reaction routine, the program reaction priority is restored to the value it was before the reaction routine started execution.

Related Keywords

3-4-21 CALL on page 3-297
3-4-22 CALLS on page 3-300
3-4-77 LOCK on page 3-379
3-4-103 REACT on page 3-415
3-4-104 REACTE on page 3-417

3-4-105 REACTI on page 3-419
3-4-112 RETURNE on page 3-428

3-4-112 RETURNE

Terminate execution of an error reaction subroutine and resume execution of the last-suspended program at the step following the statement that caused the subroutine to be invoked.

Syntax

```
RETURNE
```

Details

The RETURNE program command is intended for use in error reaction subroutines invoked by the REACTE keyword.
If a RETURNE command is used to exit from an error reaction routine, the pro- gram reaction priority is restored to the value it was before the error reaction routine started execution.
When a RETURNE command is executed in an error reaction subroutine, execution continues with the statement following the one executing when the error occurred. In this situation, a RETURN command results in the statement that generated the error being executed again, possibly causing a repeat of the error.
Because of the forward processing ability of V+, the statement that is the source of an error may not be the one executing when the error is registered. For example, when a MOVE program command is processed, the robot begins moving but during the motion several additional statements may be processed. If an envelope or similar error occurs after this forward processing, the RETURNE operation is based on the statement processing when the error occurs and not the MOVE program command.
A RETURNE command in a program that is not executed in response to an error has the same effect as a RETURN operation. A RETURNE operation takes slightly longer to execute than a RETURN operation.

Related Keywords

3-4-104 REACTE on page 3-417
3-4-111 RETURN on page 3-427

3-4-113 RIGHTY

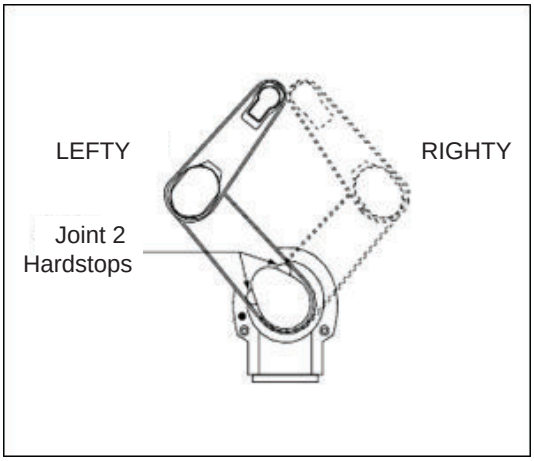
Request a change in the robot configuration during the next motion to make the first two links of a SCARA robot use the right arm orientation.

Syntax

```
RIGHTY
```

Usage Considerations

Configuration changes cannot be made during straight-line motions.
If the selected robot does not support a right-handed configuration, this command is ignored by the robot.
The RIGHTY command can be executed by any program task as long as the robot selected by the task is not attached by any other task. If the robot is not attached, this command has no effect. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the RIGHTY command causes an error.
The following figure shows the LEFTY / RIGHTY configurations for the top view of a SCARA robot.



Example

The following example will move the robot to location "point1" in the righty configuration.

```
RIGHTY
MOVE point1
```

Related Keywords

3-1-18 CONFIG on page 3-24
3-4-75 LEFTY on page 3-376
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-53 SELECT on page 3-237 (monitor command)

3-4-114 RUNSIG

Turn ON or OFF the specified digital signal as long as execution of the invoking program task continues.

Syntax

```
RUNSIG signal_num
```

Usage Considerations

Only one signal can be associated with the RUNSIG program command for each program task.

Parameters

Parameter	Description
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 is set to ON during program execution if the value is positive. A negative value results in the signal being set to OFF during program execution and is turned ON when execution stops. If no signal is specified, any RUNSIG in effect for the task is canceled.

Details

The RUNSIG program command causes the specified digital signal to be turned ON or OFF as soon as the command is executed. The signal state is restored as soon as execution of the invoking program task stops or the STOP program command is executed.
This command is typically used in an application where auxiliary equipment must be stopped when an error occurs during program execution.
Only one signal can be activated by a RUNSIG command at any one time for each program task. An error condition results unless a program cancels the first RUNSIG operation before attempting to initiate a second RUNSIG operation.
If program execution is interrupted after a RUNSIG command has been executed, the specified signal returns to the selected state again if a PROCEED or RETRY command is issued. If an SSTEP or XSTEP monitor command is issued, the signal returns to the specified state during execution of the command that is invoked. Similarly, processing of a DO monitor command temporarily activates the RUNSIG signal for the corresponding program task. The EXECUTE keyword cancels any previous RUNSIG operation for the specified program task.

Example

The following example turns ON the digital signal identified by the value of the variable "run.signal" assuming the value is positive. The signal remains ON throughout execution of the current program. The signal goes OFF when execution ends.

```
RUNSIG run.signal
```

Related Keywords

3-2-32 IO on page 3-211
3-4-109 RESET on page 3-425
3-1-93 SIG on page 3-115

3-1-91 SIG.INS on page 3-113
3-4-120 SIGNAL on page 3-437 (program command)
3-2-54 SIGNAL on page 3-238 (monitor command)

3-4-115 SELECT

Select a unit of the named device for access by the current task.

Syntax

```
SELECT device_type = unit
```

Usage Considerations

The SELECT keyword is only available for use with Robot Integrated Control systems.
The SELECT program command affects only the task in which the command is executed.
The statement SELECT ROBOT can be executed only if there is no robot attached to the current task. If there is uncertainty about whether or not a robot is attached, a program should execute a DETACH program command before executing the SELECT command.
If the EtherCAT connection to the robot is not active and a COMM_STOP error is present, you can still select it. It is recommended to use the SELECT real-valued function to get the state of the robot before issuing the SELECT program command.

Parameters

Parameter	Description
device_type	Specifier that identifies the type of device that is to be selected.The only valid device_type specifier is ROBOT.
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

In a multiple-robot system, the SELECT program command 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 SELECT program command specifies which robot receives motion commands and returns robot- related information (for example, the HERE function).
Each time a program task begins execution, robot 1 is automatically selected. If a robot is selected, information about the robot can be accessed. In order for a program to move a robot, the robot must be selected and attached with the ATTACH program command.
As an example, if robot 2 is selected by a SELECT command, all motion operations executed by the current task are directed to that robot until another SELECT command is issued. All robot-related functions will return information about robot 2.

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 operation waits or generates an error depending on the mode parameter for the ATTACH command.
If the robot number specified for the unit parameter is not valid, a (-407) *Invalid argument* error occurs.
If the robot number specified for the unit parameter is valid, but the robot is not configured in the system, a (-622) *No robot connected to system* error occurs.
If the robot number specified for the unit parameter is valid and the robot is con- figured in the system, a (1) *Success* information message occurs.

Example

The following example selects robot 3 and moves it. This program is normally not executed by task 0, since that task is attached to robot 1 by default.

```
.PROGRAM test()
    SELECT ROBOT = 3
    ATTACH (0,1)
    IF IOSTAT(0) < 0 THEN
        TYPE /B, "Error attaching robot: ", $ERROR(IOSTAT(0))
        PAUSE
    END
    MOVE x
    MOVE y
    DETACH
.END
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-6-7 OBSTACLE on page 3-480
3-1-88 SELECT on page 3-110 (real-value function)

3-4-116 SET.EVENT

Set an event associated with the specified task.

Syntax

```
SET.EVENT task, flag
```

Parameters

Parameter	Description
task	Optional 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. If this parameter is omitted, the number of the current task is used.
flag	Not used, defaults to 1.

Details

The SET.EVENT program command sets the event associated with the specified task. For example, if a task had been suspended by a WAIT.EVENT 1 statement, executing the SET.EVENT command for that task causes it to resume execution during the next available time slice for which it is eligible.

Related Keywords

3-4-24 CLEAR.EVENT on page 3-303
3-1-47 GET.EVENT on page 3-67
3-4-137 WAIT.EVENT on page 3-461

3-4-117 SET

Set the value of the location variable on the left equal to the location value on the right of the equal sign.

Syntax

```
SET location_var = location_value
```

Parameters

Parameter	Description
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, function, compound transformation.

Details

A program error is generated if the right-hand side is not defined or is not the same type of location representation (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.

Example

The following example sets the value of the transformation "pick" equal to the location of "corner" plus the location of shift relative to "corner".

```
SET pick = corner:shift
```

The following example sets the value of the precision point "#place" equal to that of the precision point "#post".

```
SET #place = #post
```

The following example sets the value of the transformation part to the current robot location, relative to the transformation pallet.

```
SET pallet:part = HERE
```

The following example sets 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

3-4-67 HERE on page 3-366 (program command)
3-2-30 HERE on page 3-207 (monitor command)

3-4-118 SETBELT

Set the encoder offset of the specified belt variable equal to the value of the expression.

Syntax

SETBELT %belt_var = expression

Usage Considerations

The BELT system switch must be enabled for this command to be executed.
The SETBELT program command cannot be executed while the robot is moving relative to the specified belt variable.
The belt variable referenced must have been defined previously using a DEFBELT program command.

⚠ WARNING

It is important to execute SETBELT each time the robot needs to track the belt to make sure the difference between the current belt position as returned by the BELT function and the belt position of the specified belt variable does not exceed 8,388,607 (^H7FFFFF) during active belt tracking. Unpredictable robot motion may result if the difference exceeds this value while tracking the belt.

Parameters

Parameter	Description
%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+ sub- tracts 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.
The SETBELT program command is generally used in conjunction with the BELT unction to set the effective belt position to 0. This must be done each time the robot will perform a sequence of motions relative to the belt and must be done before the first motion of such a sequence.
The SETBELT program command can be used to synchronize robot motion with the encoder value latched by an external signal or by a vision system. Refer to the *3-1-61 LATCHED* on page 3-88 function and the *3-1-28 DEVICE* on page 3-39 function for more information.

Example

The following example waits for a digital signal and then sets the belt position to 0. That is done by setting the belt offset equal to the current belt position. Then, the robot is moved onto the belt.

```
WAIT sig(1001)
SETBELT %belt1 = BELT(%belt1)
MOVES %belt1:pickup
```

Related Keywords

3-1-11 BELT on page 3-16
3-4-33 DEFBELT on page 3-314
3-1-61 LATCHED on page 3-88
3-4-139 WINDOW on page 3-464 (program command)
3-1-127 WINDOW on page 3-164 (real-valued function)

3-4-119 SETDEVICE

Initialize a device or set device parameters. The operation performed depends on the device referenced.

Syntax

```
SETDEVICE (type, unit, error, command) p1, p2, ...
```

Usage Considerations

The syntax contains optional parameters that apply only to specific device types and commands.

Parameters

Parameter	Description
type	Real value, variable, or expression interpreted as an integer that indicates the type of device being referenced. The following types are available • 0: Belt encoder • 1: Reserved for future use • 2: Force Processor board • 3: Robot device (Omron Robotics and Safety Technologies, Inc. servo only) • 4: Vision • 5: Reserved for future use
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 command 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 command. 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 command 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 as described below.

Command	Description
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.

Related Keywords

3-1-28 DEVICE on page 3-39

3-4-120 SIGNAL

Turn ON or OFF external digital output signals or internal software signals.

Syntax

```
SIGNAL signal_num, ..., signal_num
```

Parameters

Parameter	Description
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. The SIGNAL command ignores parameters with a 0 value.

Details

The magnitude of a signal_num parameter determines which digital or internal signal is to be considered.

Only digital signals that are installed and configured as outputs can be used. To check your current digital I/O configuration, use the IO monitor command. Signals 3001 and 3002 refer to the robot selected by the current task. Signal 3001 is the state of the hand-close solenoid. Signal 3002 is the state of the hand-open solenoid.

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.

Executing this program command keyword for an IOBlox signal that does not physically exist will cause a -508 *Device not ready* error.

The magnitude of a signal_num parameter determines which digital or internal signal is to be considered.

Only digital signals that are installed and configured as outputs can be used. To check your current digital I/O configuration, use the IO monitor command. Signals 3001 and 3002 refer to the robot selected by the current task. Signal 3001 is the state of the hand-close solenoid. Signal 3002 is the state of the hand-open solenoid.
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.
Executing this program command keyword for an IOBlox signal that does not physically exist will cause a -508 *Device not ready* error.
All V+ digital output keywords do not wait for a complete V+ cycle and they are turned ON immediately. Digital inputs are checked approximately every 4 ms for Standard Control systems and approximately 8 ms or 16 ms for Robot Integrated systems.
This makes it possible to turn ON and OFF a signal before the system can read the output.

Example

The following example turns OFF the external output signal specified by the value of the variable "reset" assuming the value of reset is positive, and turns ON external output signal 4.

```
SIGNAL -reset, 4
```

The following example turns external output signal 1 OFF, external output signal 4 ON, and internal software signal 2010 ON.

```
SIGNAL -1, 4, 2010
```

Related Keywords

3-2-4 BITS on page 3-171 (monitor command)
3-4-16 BITS on page 3-290 (program command)
3-1-12 BITS on page 3-17 (real-valued function)
3-2-32 IO on page 3-211
3-4-87 NOOVERLAP on page 3-394
3-4-89 OVERLAP on page 3-397
3-4-109 RESET on page 3-425
3-4-114 RUNSIG on page 3-429
3-1-93 SIG on page 3-115
3-1-91 SIG.INS on page 3-113
3-2-54 SIGNAL on page 3-238 (monitor command)

3-4-121 SINGLE

Limit rotations of the robot wrist joint to the range -180 degrees to +180 degrees.

Syntax

```
SINGLE ALWAYS
```

Usage Considerations

Only the next robot motion is affected if the ALWAYS parameter is not specified.
The SINGLE program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This program command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the SINGLE command causes an error.

Parameters

Parameter	Description
ALWAYS	Optional qualifier that establishes SINGLE as the default condition. If ALWAYS is specified, SINGLE remains in effect continuously. If ALWAYS is not specified, the SINGLE command applies 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 hits the stop.
Executing the SINGLE program command causes subsequent motion(s) to force the wrist back to near the center of its range, so that straight-line motions will not fail in this way.
The SINGLE command is commonly specified during an APPRO operation 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.
The SINGLE setting is ignored if NOOVERLAP is active.

Related Keywords

3-1-18 CONFIG on page 3-24
3-4-87 NOOVERLAP on page 3-394
3-4-89 OVERLAP on page 3-397
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)

3-4-122 SOLVE.ANGLES

Compute the robot joint positions for the current robot that are equivalent to a specified transformation.

Syntax

```
SOLVE.ANGLES o.jts[o.idx], o.flags, error = trans, i.jts[i.idx],i.flags
```

Usage Considerations

Since the computation performed by this command is a function of the geometry of the robot based on link dimensions, number of axes, tool offsets, and base off- sets, robots with different geometric parameters yields different results. Robots of the same general type may differ slightly in their dimensions and this pro- gram command may return slightly different results when executed on two different robot systems of the same type.
The SOLVE.ANGLES program command returns information for the robot selected by the task executing the command.
If the V+ system is not configured to control a robot, executing this command does not generate an error because of the absence of a robot. However, the information returned may not be meaningful.

Parameters

Parameter	Description	
o.jts	Real-valued array in which the computed joint angles are returned. The first specified element of the array contains the position for joint 1, the second element contains the value for joint 2, etc. For rotating joints, the joint positions are in degrees. For translational joints, the joint positions are in millimeters.	
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 0 contains 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 described below.	
	Bit Flag	Description
	Bit 1 (LSB) RIGHTY (mask value = 1)	If this bit is ON, the position has the robot in a right-armconfiguration. Otherwise, the position has the robot in a left-arm configuration.
	Bit 2 BELOW (mask value = 2)	If this bit is ON, the position has the robot con- figured 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 ON, the position has the robot con- figured 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.

<table>
<tr><th>Parameter</th><th colspan="4">Description</th></tr>
<tr><td>error</td><td colspan="4">Real variable that receives a bit-flag value that indicates whether any joint positions were computed to be outside of their working range, or whether the XYZ position of the destination was outside the working envelope of the robot. The bit flags are interpreted as described below.</td></tr>
<tr><td></td><td>Bit Flag</td><td colspan="3">Description</td></tr>
<tr><td></td><td>Bits 1 - 12 Joint/ Motor out of range</td><td colspan="3">If ON, the computed value for the joint or motor was found to be out- side of its limit stops as shown below.</td></tr>
<tr><td></td><td></td><td>Bit</td><td>Joint/Motor #</td><td>Mask Value</td></tr>
<tr><td></td><td></td><td>1</td><td>1</td><td>^H1</td></tr>
<tr><td></td><td></td><td>2</td><td>2</td><td>^H2</td></tr>
<tr><td></td><td></td><td>3</td><td>3</td><td>^H4</td></tr>
<tr><td></td><td></td><td>4</td><td>4</td><td>^H8</td></tr>
<tr><td></td><td></td><td>5</td><td>5</td><td>^H10</td></tr>
<tr><td></td><td></td><td>6</td><td>6</td><td>^H20</td></tr>
<tr><td></td><td></td><td>7</td><td>7</td><td>^H40</td></tr>
<tr><td></td><td></td><td>8</td><td>8</td><td>^H80</td></tr>
<tr><td></td><td></td><td>9</td><td>9</td><td>^H100</td></tr>
<tr><td></td><td></td><td>10</td><td>10</td><td>^H200</td></tr>
<tr><td></td><td></td><td>11</td><td>11</td><td>^H400</td></tr>
<tr><td></td><td></td><td>12</td><td>12</td><td>^H800</td></tr>
<tr><td></td><td>Bit 13 Collision (mask value = ^H1000)</td><td colspan="3">When this bit is turned ON, a collision has been detected.</td></tr>
<tr><td></td><td>Bit 14 Too close (mask value = ^H2000)</td><td colspan="3">The XYZ position of the destination cannot be reached because it was too close to the column of the robot.</td></tr>
<tr><td></td><td>Bit 15 Too far (mask value = ^H4000)</td><td colspan="3">The XYZ position of the destination cannot be reached because it was too far away from the robot.</td></tr>
<tr><td></td><td>Bit 16 Joint vs. motor (mask value = ^H8000)</td><td colspan="3">If ON, a motor is limiting. Otherwise, a joint is limiting.</td></tr>
<tr><td>trans</td><td colspan="4">Transformation variable, function, or compound transformation that defines the robot location of interest.</td></tr>
<tr><td>i.jts</td><td colspan="4">Transformation variable, function, or compound transformation that defines the robot location of interest.</td></tr>
<tr><td>i.idx</td><td colspan="4">Real array that contains the joint positions representing the starting location for the robot. These values are referenced for multiple-turn joints to minimize joint rotations and 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.</td></tr>
</table>

<table>
<tr><th>Parameter</th><th colspan="2">Description</th></tr>
<tr><td>i.flags</td><td colspan="2">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 0 must contain the position for joint 1.</td></tr>
<tr><td></td><td>Bit Flag</td><td>Description</td></tr>
<tr><td></td><td>Bit 1: (LSB) RIGHTY
(mask value = 1)</td><td>If this bit is ON, the robot is assumed initially to be in a right-arm configuration. Otherwise, the robot is assumed to be in a left-arm configuration.</td></tr>
<tr><td></td><td>Bit 2: BELOW
(mask value = ^H2)</td><td>If this bit is ON, 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 SCARA configurations that do not have an elbow that moves in a vertical plane.</td></tr>
<tr><td></td><td>Bit 3: FLIP
((mask value = ^H4)</td><td>If this bit is ON, 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.</td></tr>
<tr><td></td><td>Bit 9: Change RIGHTY/LEFTY
(mask value = ^H100)</td><td>If this bit is ON, the command attempts to compute a set of joint positions corresponding to the RIGHTY/LEFTY con- figuration specified by bit 10.</td></tr>
<tr><td></td><td>Bit 10: Change to RIGHTY
(mask value = ^H200)</td><td>When bit 9 is ON and this bit is ON, the command attempts to compute joint positions for a right-arm configuration. If bit 9 is ON and this bit is 0, the command attempts to compute a set of joint positions for a left-arm configuration.</td></tr>
</table>

Parameter	Description	
i.flags	Bit Flag	Description
	Bit 11: Change BELOW/ABOVE (mask value = ^H400)	If this bit is ON, the command attempts to compute a set of joint positions corresponding to the BELOW/ABOVE configuration specified by bit 12. This bit is ignored for robots like 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 ON and this bit is ON, the command attempts to compute joint positions for an elbow down configuration. If bit 11 is ON and this bit is 0, the command attempts to compute joint positions for an elbow-up configuration. This bit is ignored for robots like SCARA con- figurations that do not have an elbow that moves in a vertical plane.
	Bit 13: Change FLIP/NOFLIP (mask value = ^H1000)	If this bit is ON, the command attempts to compute a set of joint positions corresponding to the FLIP/NOFLIP configuration specified by bit14. 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 ON and this bit is ON, the command attempts to compute joint positions for a FLIP wrist configuration. If bit 13 is ON and this bit is 0, the command attempts to compute joint positions for a NOFLIP wrist con- figuration. 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 ON, if the computed value of joint #2 is within 10 degrees of having the outer link fully extended (joint 2 between -10 and +10 degrees in value), an out-of-range error for joint 2 is signaled.
	Bit 22: Single-turn joint 4 (mask value = ^H200000)	When this bit is ON, the computed value of joint 4 is restricted to the range of -180 to +180 degrees.
	Bit 23: Straight-line motion (mask value = ^H400000)	When this bit is ON, the joint positions returned must correspond to the same configuration as those initially specified. No change in robot configuration is allowed.

Details

The SOLVE.ANGLES program command 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 examples below do not perform any useful function but are intended to illustrate how the SOLVE.ANGLES command operates. After execution of these commands, both the jts2 and jts arrays contain approximately the same values. Any differences in the values are due to computational rounding 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

3-4-31 DECOMPOSE on page 3-312
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-1-95 SOLVE.FLAGS on page 3-117
3-4-123 SOLVE.TRANS on page 3-444

3-4-123 SOLVE.TRANS

Compute the transformation equivalent to a given set of joint positions for the current robot.

Syntax

SOLVE.TRANS transform, error = joints[index**]**

Usage Considerations

Since the computation performed by this command is a function of the geometry of the robot based on link dimensions, number of axes, tool offsets, and base off- sets, robots with different geometric parameters yield different results. Robots of the same general type may differ slightly in their dimensions and this command may return slightly different results when executed on two different robot systems of the same type.

The SOLVE.TRANS program command refers to the robot selected by the task executing the command.

If the V+ system is not configured to control a robot, executing the SOLVE.TRANS command does not generate an error because of the absence of a robot. However, the information returned may not be meaningful.

Parameters

Parameter	Description
transform	Transformation variable or transformation array element in which the result is stored.

Parameter	Description
error	Real variable that is set to a V+ error code if a computational error occurred during processing of the command. This variable is set to 0 if no error occurs. The only error that is 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 0 must contain the position for joint 1.

Details

The SOLVE.TRANS program command 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 example below computes the position and orientation to which the robot is moved 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

3-4-31 DECOMPOSE on page 3-312
3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-4-122 SOLVE.ANGLES on page 3-439
3-4-123 SOLVE.TRANS on page 3-444

3-4-124 SPEED

Set the nominal speed for subsequent robot motions.

Syntax

```
SPEED speed_factor, r_speed_factor units ALWAYS
```

Usage Considerations

SPEED 100,100 ALWAYS is assumed whenever program execution is started and when a new execution cycle begins. This is the default state of the V+ system.
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 program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the SPEED command causes an error.

Parameters

Parameter	Description
speed_factor	Real value, variable, or expression whose value is used as a new speed factor. The value 100 is considered normal full speed and 50 is one-half of full speed. If IPS or MMPS is specified for units, the value is considered the linear tool tip speed. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information about full speed settings.
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 and 50 is one-half of full speed.
units	Optional keyword that determines how to interpret the speed_factorparameter. The IPS (inches per second) or MMPS (millimeters per second) keywords can be used.
ALWAYS	Optional qualifier that specifies the program speed_factor will be active until the next SPEED command changes pro- gram speed. Otherwise, it is active only for the next motion operation (including APPROS and DEPART).

Details

If the units parameter is omitted, this command 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 performed is determined by

combining the values specified in this command 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 makes 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 are 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.
When the monitor speed is set, its value is limited to a maximum of 100%. No error is reported if a higher speed setting is specified. Speed cannot be less than 0.000001 (1.0E-6).
When the program speed is set, its value is limited to a maximum that depends on the robot being controlled. No error is reported if a higher speed setting is specified. The maximum speed value for the current robot is returned by the real-valued function SPEED(8).
If a tool with a large offset is attached to the robot and straight-line motions are executed, the robot joint and flange speeds can be very large when rotations about the tool tip are made. The r_speed_factor parameter provides 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.

- If the units parameter is omitted, the rotational speed is set to the value of speed_factor.
- 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 function is used to read the program speed, the value returned is a percentage speed factor and not an absolute speed setting.
The final robot speed is a combination of the monitor speed (SPEED monitor command), the program speed (SPEED program command), and the acceleration or deceleration (ACCEL program command).

Example

The following example sets the program speed to 50% for the next motion assuming the monitor speed is 10):

```
SPEED 50
```

The following example sets 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 are limited to 40% of maximum. The settings remains in effect until changed by another SPEED command.

```
SPEED 20, 40 IPS ALWAYS
```

The following sets the monitor speed to 50% of normal.

```
SPEED 50 MONITOR
```

Related Keywords

3-4-3 ACCEL on page 3-270
3-4-41 DURATION on page 3-325
3-3-2 IPS on page 3-265
3-3-3 MMPS on page 3-266
3-6-10 SCALE.ACCEL on page 3-484
3-4-115 SELECT on page 3-431 (program command)
3-2-53 SELECT on page 3-237 (real-valued function)
3-2-55 SPEED on page 3-240 (monitor command)
3-1-96 SPEED on page 3-118 (real-valued function)

3-4-125 STOP

Terminate execution of the current program cycle.

Syntax

```
STOP
```

Usage Considerations

STOP does not halt program execution if there are more program cycles to execute.
The PROCEED program command cannot be used to resume program execution after a STOP command causes the program to halt.
If program execution is halted by a STOP command, FCLOSE and / or DETACH operations are forced on all attached I/O devices.

Details

The STOP program command terminates execution of the current program unless more program loops are to be completed, in which case execution of the program continues at its first step (refer to the EXECUTE keywords). The STOP command is used to mark the end of a program execution pass. The STOP program command counts one more program cycles 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 operation occurred within a subroutine or reaction program.
A RETURN program command in a main program has the same effect as a STOP command. A main program is one that is invoked by an EXECUTE keyword or a PRIME or XSTEP monitor command. A subroutine is a program that is invoked by a CALL or CALLS program command (or a reaction) within another program.
The HALT program command does not have the same effect as a STOP command because it cancels all remaining cycles.

Related Keywords

3-2-1 ABORT on page 3-167 (monitor command)
3-4-1 ABORT on page 3-268 (program command)
3-2-22 EXECUTE on page 3-195
3-4-66 HALT on page 3-365
3-4-93 PAUSE on page 3-401

3-4-126 SWITCH

Enable or disable a system switch based on a value.

Syntax

```
SWITCH switch_name = value
SWITCH switch_name[index] = value
```

Usage Considerations

If the specified switch accepts an index qualifier and the index is 0 or omitted with or without the brackets, all the elements of the switch array are set according to the value given.

Parameters

Parameter	Description
switch_name	Name of the system switch whose setting is to be modified. The switch name can be abbreviated to the minimum length that identifies it uniquely.
index	For system 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.
value	Real value, variable, or expression that determines whether the system switch is to be enabled or disabled. The switch is enabled if the value is TRUE (non-zero).The switch is disabled if the value is FALSE (zero).

Details

The SWITCH program command 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.
Refer to *V+ User's Manual (Cat. No. I671)* for more information about system switches.

Example

The following example shows how the SWITCH function and command can be used to save the setting of a system switch and to restore it.

```
old.upper = SWITCH(UPPER)
.
.
.
SWITCH UPPER = old.upper
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-4-127 TIMER

Set the specified system timer to the given time value.

Syntax

```
TIMER timer_number = time_value
```

Usage Considerations

Times measured by V+ are precise only to within 1 millisecond (0.001 seconds) and shorter times cannot be measured.
Timers with numbers <= 0 are read-only and cannot be set with this command.

Parameters

Parameter	Description
timer_number	Real-valued expression interpreted as the integer number of the timer to be set. The value must range from 1 to 15.
time_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 used as described in the examples below, the timers can be used to measure an interval of 596 hours from when they were set by the TIMER command. Timers have a resolution of one millisecond and a maximum count of > 2.E+009.
The *3-1-109 TIMER* on page 3-141 function keyword can also be used to read the instantaneous value of a system timer.

Example

The following examples provides two ways to wait for a certain amount of time using the TIMER command and function. Each example sets the timer and then waits until the timer value has changed by the delay period.

```
TIMER 1 = 0
WAIT TIMER(1) > delay
TIMER 1 = -delay
WAIT TIMER(1) > 0
```

Related Keywords

3-1-109 TIMER on page 3-141 (real-valued function)

3-4-128 TOOL

Set the internal transformation used to represent the location and orientation of the tool tip relative to the tool mounting flange of the robot.

Syntax

TOOL transformation_value

Usage Considerations

The TOOL program command causes a break in continuous-path motion.
The TOOL program command can be executed by any program task as long as the robot selected by the task is not attached by any other task. This command applies to the robot selected by the task.
If the V+ system is not configured to control a robot, executing the TOOL command causes an error.

Parameters

Parameter	Description
transformation_value	Optional transformation variable or function, or compound transformation expression that is the new tool transformation. If the transformation value is omitted, the tool is set to NULL.

Details

The TOOL program command 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 *3-2-67 TOOL* on page 3-258 monitor command for a complete description of this command's operation. To define a tool transformation, refer to *V+ User's Manual (Cat. No. I671)* for more information.

Related Keywords

3-4-115 SELECT on page 3-431 (program command)
3-1-88 SELECT on page 3-110 (real-valued function)
3-2-67 TOOL on page 3-258 (monitor command)
3-1-110 TOOL on page 3-143 (transformation function)

3-4-129 TRANS.TO.POSE

Returns an array determined by translation and rotation in the extrinsic Euler angles X, Y, and Z that is based on a transformation.

Syntax

```
TRANS.TO.POSE array_name[index] = trans
```

Usage Considerations

Intrinsic refers to the internal perspective of an object's movement or rotation. When an angle changes, all internal axes of the object rotate simultaneously.
Extrinsic refers to the external viewpoint of an object's movement or rotation. From a global perspective, when an axis rotates, subsequent rotations adhere to the reference convention of the external coordinate system.

Parameters

Parameter	Description
array_name	Name of the real-valued array that has its elements defined. If the array is not already defined, it will be created.
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 is incremented as the values are assigned.
trans	Transformation variable, function, or compound transformation that defines the location of interest.

Details

This program command assigns values to consecutive elements of the named array corresponding to the conversion of the specified transformation. The transformation is determined through ZYZ intrinsic Euler axis rotations, with a structured representation consisting of six elements corresponding to X, Y, Z, yaw, pitch, and roll.

The the last three elements of the array represent the rotation of the robot arm. They are applied in the order that follows below.

- Rotation around the x-axis.
- Rotation around the original y-axis.
- Rotation around the original z-axis.

As specified by the intrinsic Euler ZYZ angles, the rotation is applied in the order that follows below.

- Rotation around the z1-axis.
- Rotation around the new y-axis.
- Rotation around the new z2-axis

Example

The following example assigns the components derived from the conversion of the transformation part to elements 0 to 5 of the array named "position."

```
TRANS.TO.POSE position[0] = TRANS(1, 10, 3, 10, 20, 15)
```

Related Keywords

3-4-99 POSE.TO.TRANS on page 3-408
3-1-112 TRANS on page 3-144

3-4-130 TYPE

Display the information described by the output specifications on the Monitor Window.

Syntax

```
TYPE output_specification, ..., output_specification
```

Usage Considerations

No output is generated if the MESSAGES system switch is disabled. A blank line is output if no parameters are provided.

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 command cannot exceed 512 characters. The /S format control specifier described below can be used to output longer messages.

Parameters

An output_specification can consist of any of the following components in any order, separated by commas.

- A string expression.
- A real-valued expression, which is evaluated to determine a value to be displayed.
- Format control specifier, which determines the format of the output message (see details below).

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 command unless another specifier is used to change their effect.

For 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 uses 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.

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.

Specifier	Description
/En.m	Output values in scientific notation in fields n spaces wide, with m digits the fractional parts. If n is not zero, it must be large enough to include space for a minus sign if the dis- played value is negative, one digit to the left of the decimal point, a decimal point if m is not zero, m digits, and four or five characters for the exponent.
/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 /En.m format is used.
/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.

Specifier	Description
/Cn	Output the characters carriage return (CR) and line feed (LF)n times. This will result in n blank lines if the format control specifier is at the beginning or end of an output specification. Otherwise, n-1 blank lines will result.

Specifier	Description
/S	Do not output a carriage return (CR) or line feed (LF) after displaying the current line.
/Un	Move the cursor up n lines.
/Xn	Output n spaces.

The following specifier can be used to perform control functions.

Specifier	Description
/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

The following example will display the message "Point 5 = 12.67". This example assumes that the real variable "i" has the value 5 and that array element "point [5]" has the value 12.666666.

```
TYPE, "Point", i, " = " /F5.2, point[i]
```

If point[5] has the value 1000, the statement above would display "Point 5 = *****" because the value is too large to be displayed in the specified format (/F5.2). The command can display any value for point[5] if the format specification were /F0.5.

Related Keywords

3-1-35 $ENCODE on page 3-45
3-6-6 MESSAGES on page 3-479
3-4-102 PROMPT on page 3-413
3-4-140 WRITE on page 3-466

3-4-131 UNTIL

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.

Syntax

```
UNTIL expression
```

Usage Considerations

UNTIL must be used in conjunction with a DO control structure. Refer to the *3-4-38 DO* on page 3-321 command for more information.

Parameters

Parameter	Description
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 0, program execution continues with the statement following the matching DO command. If the expression is nonzero, program execution continues with the statement following the UNTIL command.

Example

The following example is a loop that continues to prompt you to enter a number until you enter one that is greater than or equal to 0.

```
DO
    PROMPT "Enter a positive number: ", number
UNTIL number >= 0
```

Related Keywords

3-4-38 DO on page 3-321
3-4-47 EXIT on page 3-334
3-4-84 NEXT on page 3-391
3-4-138 WHILE on page 3-463

3-4-132 VALUE

Indicate the values that a CASE statement expression must match in order for the program statements immediately following to be executed.

Syntax

VALUE expression_list:

Usage Considerations

VALUE must be part of a CASE control structure. Refer to the *3-4-23 CASE* on page 3-301 command for more information.

Parameters

Parameter	Description
expression_list	Real value, variable, or expressions, separated by commas, that defines the value to be matched in the CASE structure to determine which statements are executed.

Related Keywords

3-4-8 ANY on page 3-277
3-4-23 CASE on page 3-301

3-4-133 VPARAMETER

Sets the current value of a vision tool parameter.

Syntax

```
VPARAMETER(sequence_id, tool_id, parameter_id, index_id, object_id) $ip = value
```

Usage Considerations

Refer to *Robot Vision Manager User's Manual (Cat. No. I667)* for additional information.

Parameters

Parameter	Description
sequence_id	Optional index of the vision sequence. The first sequence is 1.
tool_id	Optional index of the tool in the vision sequence. The first tool is 1. parameter_id Optional identifier (ID) of the parameter.
parameter_id	Optional identifier (ID) of the parameter.
index_id	Reserved for internal use. Value is always 1.
object_id	Some parameters require an object index to access specific values in an array.
$ip	IP address of the vision server. The vision server is the PC on which the Robot Vision Manager software is running and uses a standard IP address format (192.168.1.120 for example).
value	Real-value expression.

Details

If no value is provided for optional parameters, they default to 1.

Example

The following example will set a Locater to find a maximum of 4 object instances where the Maximum Instance Count is - 519.

```
VPARAMETER(1, 2, 519) $ip = 4
```

Related Keywords

3-1-123 VLOCATION on page 3-157
3-1-124 VPARAMETER on page 3-160 (real-valued function)
3-1-125 VRESULT on page 3-161
3-4-134 VRUN on page 3-458
3-1-126 VSTATE on page 3-163
3-5-5 VTIMEOUT on page 3-473
3-4-135 VWAITI on page 3-459

3-4-134 VRUN

Initiates the execution of a vision sequence.

Syntax

VRUN $ip, sequence_id

Parameters

Parameter	Description
$ip	IP address of the vision server. The vision server is the PC on which the Robot Vision Manager is running and uses a standard IP address format (192.168.1.120 for example).
sequence_id	Index of the vision sequence. The first sequence is 1.

Example

The following example executes the first sequence

```
VRUN $ip, 1
```

Related Keywords

3-1-123 VLOCATION on page 3-157
3-1-124 VPARAMETER on page 3-160 (real-valued function)
3-4-133 VPARAMETER on page 3-457 (program command)
3-1-125 VRESULT on page 3-161
3-1-126 VSTATE on page 3-163

3-5-5 VTIMEOUT on page 3-473
3-4-135 VWAITI on page 3-459

3-4-135 VWAITI

Waits until the specified vision sequence reaches the state specified by the type parameter.

Syntax

```
VWAITI (sequence_id) $ip, type
```

Parameters

Parameter	Description
sequence_id	Optional index of the vision sequence. The first sequence is 1.
$ip	IP address of the vision server. The vision server is the PC on which the Robot Vision Manager is running and uses a standard IP address format (192.168.1.120 for example).
type	Optional vision sequence state to reach before completing as described below. • 0: Wait for full completion (default). • 1: Wait until image acquisition has completed.

Details

Use a VWAITI program command after a VRUN program command. In a conveyor-tracking application, the absence of a specific VWAITI command can interfere with the Virtual Camera tool and the Communication tool to cause a delay in the execution of the application.
Issuing a VWAITI command can block other tasks executing other V+ Robot Vision Manager keywords. Consider using VSTATE if your application has multiple V+ tasks interacting with Robot Vision Manager sequences.

Example

The following example will execute the first sequence and wait for full completion.

```
VRUN $ip, 1
VWAITI (1) $ip, 0
```

Related Keywords

3-1-123 VLOCATION on page 3-157
3-1-124 VPARAMETER on page 3-160 (real-valued function)
3-4-133 VPARAMETER on page 3-457 (program command)
3-1-125 VRESULT on page 3-161

3-4-134 VRUN on page 3-458
3-1-126 VSTATE on page 3-163
3-5-5 VTIMEOUT on page 3-473

3-4-136 WAIT

Put the program into a wait loop for one trajectory cycle. If a condition is specified, wait until the condition is TRUE.

Syntax

```
WAIT condition
```

Usage Considerations

To wait for a specific time period, use the WAIT.EVENT program command.
During execution, use the PROCEED monitor command to cancel a WAIT command in an application program.

Parameters

Parameter	Description
condition	Optional real value, variable, or expression that is tested for a TRUE (nonzero) or FALSE (zero) value.

Details

A WAIT program command with no condition specified is useful in programs that need to perform an operation only once each trajectory cycle.
If no condition is supplied with the WAIT program command, program execution is suspended until the next trajectory cycle. Trajectory cycles occur at 16, 8, and 4 millisecond intervals, depending on the system configuration.
If a trajectory cycle delay is necessary (for example, while manipulating signals monitored by REACT or REACTI program commands), execute two consecutive WAIT commands with no arguments.
If a condition is specified, the WAIT command will suspend program execution until the condition exists. For example, the state of one or more digital signals can be used as the condition for continuation.

Example

The following example will stop program execution while external input signal 1001 is ON and 1003 is OFF and will poll once each V+ trajectory cycle.

```
WHILE SIG(1000, -1003) DO
    WAIT
END
```

Related Keywords

3-4-108 RELEASE on page 3-424
3-4-137 WAIT.EVENT on page 3-461
3-2-68 WAIT.START on page 3-259

3-4-137 WAIT.EVENT

Suspend program execution until a specified event has occurred or until a specified amount of time has elapsed.

Syntax

```
WAIT.EVENT mask, timeout
```

Usage Considerations

If a WAIT.EVENT program command in an application program has execution suspended, the WAIT.EVENT can be canceled with the PROCEED monitor command.

Parameters

Parameter	Description
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, mask value = 1): Wait for I/O If this bit is ON, 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.

Details

The WAIT.EVENT program command is used to suspend program execution until a specified event has occurred or until a specified amount of time has elapsed in the timeout clock.
When the program resumes execution after a WAIT.EVENT command, the GET.EVENT function can be used to verify that 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 0 or is omitted, this command is a very efficient method to suspend program execution for the time period specified by the timeout parameter.
If the timeout parameter is omitted or has a negative or 0 value, this command suspends program execution indefinitely until the specified event occurs.

If both mask and timeout are 0 or omitted, this command has no impact on system behavior.
The WAIT.EVENT 1 statement waits for an event to be signaled for a task. Events are signaled by either a SET.EVENT program command or by a pending no-wait I/O program command when the I/O operation is completed.
There is no way to confirm why the event was set. It may have been set by an I/O operation, a SET.EVENT program command, or some internal system process. For this reason, it is necessary to evaluate for the desired condition after executing the WAIT.EVENT operation. For I/O, repeat the no-wait I/O operation or use the IOSTAT function. For SET.EVENT operation issued by other tasks, define and check a global variable.
To avoid program logic issues where the event is set or cleared between evaluation and waiting, use the following loop in the waiting task.
The statement order is critical in this loop.

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.

Example

The following example suspends program execution for 5.5 seconds.

```
WAIT.EVENT , 5.5
```

The following example suspends program execution until the completion of any system input / output, or until another program task sets events using the SET.EVENT command.

```
WAIT.EVENT 1
```

The following example suspends program execution for five seconds until the completion of any system input/ output or until another program task uses the SET.EVENT command to set events. The current program should use the GET.EVENT function to determine whether an event has occurred or five seconds has elapsed.

```
WAIT.EVENT 1, 5
```

Related Keywords

3-4-24 CLEAR.EVENT on page 3-303
3-1-47 GET.EVENT on page 3-67
3-4-116 SET.EVENT on page 3-432

3-4-138 WHILE

Initiate processing of a WHILE structure if the condition is TRUE or skipping of the WHILE structure if the condition is initially FALSE.

Syntax

```
WHILE condition DO
```

Usage Considerations

Every WHILE statement must be part of a complete WHILE ... DO ... END structure.

Parameters

Parameter	Description
condition	Real-valued expression that is evaluated and tested for a TRUE (nonzero) or FALSE (zero) value.

Details

This structure provides another method for executing a group of statements until a control condition is satisfied (similar to the DO command). The complete syn- tax for the WHILE structure is shown below.

```
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 statement after the END command.

Unlike the DO structure described elsewhere, the group of statements within the WHILE structure may not be executed at all. If the condition has a FALSE value when the WHILE is first executed, then the group of statements are not executed.
When this structure is used, it is assumed that some action occurs within the group of enclosed statements 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 stops and the program continues.

If either feeder is empty when the WHILE structure is first encountered, then execution immediately skips to any statements after END.

```
feeder.1 = 1037
feeder.2 = 1038

WHILE SIG(feeder.1, feeder.2) DO
    CALL move.part.1()
        CALL move.part.2()
END
```

Related Keywords

3-4-38 DO on page 3-321
3-4-47 EXIT on page 3-334
3-4-84 NEXT on page 3-391
3-4-131 UNTIL on page 3-455

3-4-139 WINDOW

Set the boundaries of the operating region of the specified belt variable for conveyor tracking.

Syntax

WINDOW %belt_var = location, location, program, priority

Usage Considerations

The belt variable referenced must have already been defined using a DEFBELT program command.

Parameters

Parameter	Description
%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 command automatically determines which transformation represents the upstream boundary and which is for the downstream boundary.

Parameter	Description
program	Optional program that is called if a window violation occurs while tracking the belt. It is 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 program command 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 system parameter and the nature of the error.
The presumption is made that the specified program directs the robot as required to recover from the window violation.
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 V+ trajectory cycle. 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.
If no program has been specified, the robot is immediately stopped and a window violation program error is signaled. If a REACTE operation has been executed, the REACTE operation is activated. Otherwise, program execution is terminated.

Example

The following example executes the subroutine "belt.error" if a window violation occurs while the robot is tracking the belt. The working window for the belt variable "%belt1" is defined by locations "win1" and "win2".

```
WINDOW %belt1 = win1, win2, belt.error
```

Related Keywords

3-1-11 BELT on page 3-16 (real-valued function)
3-5-1 BELT.MODE on page 3-468
3-1-14 BSTATUS on page 3-20
3-4-33 DEFBELT on page 3-314
3-4-118 SETBELT on page 3-434
3-1-127 WINDOW on page 3-164 (real-valued function)

3-4-140 WRITE

Write a record to an open file or to any I/O device. For network device, write a string to an attached device and open a TCP connection.

Syntax

```
WRITE (lun,record_num) format_list
```

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 program command.
Program execution waits for the write operation to complete unless there is a /N format specifier in the format list.

Parameters

Parameter	Description
lun	Real-valued expression that identifies the device to be accessed.Refer to the *3-4-11 ATTACH* on page 3-279 program command 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 (default) to write the next sequential record. If the value is not 0, 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 1 to 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. Refer to the *3-4-106 READ* on page 3-421 program command for more information.
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 program command. 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 command 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 command appends NULL 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 deallocated when a client dis- connects. During a connection the READ program command that receives data from the TCP logical unit returns the client handle. A WRITE program command can then use the handle value to send data to the corresponding client.

Example

The following example writes a message stored in the variable "$message" to the manual control pendant.

```
ATTACH (1)
WRITE (1) $message IF IOSTAT(0) < 0 THEN

TYPE /B, "Error Writing to Disk: ", $ERROR(IOSTAT(0))
PAUSE
END
DETACH(1)
```

The following example writes a file with variable-length records to the system disk drive.

```
ATTACH (dlun, 4) "DISK"
FOPENW (dlun) "A:testfile.dat"
FOR i = 0 TO LAST($lines[])
    WRITE (dlun) $lines[i]
END
FCLOSE (dlun)
DETACH (dlun)
```

The following writes the string " $str" to the client defined by the handle which must have been defined previously when the a message was received. It does not wait for acknowledgment.

```
WRITE (lun, handle) $str, /N
```

Related Keywords

3-4-11 ATTACH on page 3-279
3-4-36 DETACH on page 3-318
3-4-49 FCLOSE on page 3-337
3-4-53 FEMPTY on page 3-343
3-4-56 FOPEN on page 3-348
3-4-57 FOPENA on page 3-349
3-4-58 FOPEND on page 3-352
3-4-59 FOPENR on page 3-354
3-4-60 FOPENW on page 3-356
3-1-58 IOSTAT on page 3-83
3-4-102 PROMPT on page 3-413
3-4-106 READ on page 3-421
3-4-130 TYPE on page 3-453

3-5 System Parameter Keywords

Use the information in this section to understand system parameter keywords and their use with the V + system.

3-5-1 BELT.MODE

System parameter that sets the characteristics of the conveyor tracking feature of the V+ system.

Syntax

```
...BELT.MODE[robot_num]
```

Usage Considerations

The current value of the BELT.MODE parameter can be determined with the PARAMETER monitor command or function keywords.
The value of the BELT.MODE parameter can be modified only with the PARAMETER monitor command or program command keywords.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

This parameter is interpreted as a bit-flag word. The initial setting of this para- meter is 0 (all the bits are zero). Bits can be set by assigning the value resulting from adding together the desired bit mask values.
The bit flags have the following interpretations:

Bit	Description
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 down- stream 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	Description
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 down-stream 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 viola-tions cause planning to wait until the location comes into the window. Destination locations that are down-stream of the belt window cause window errors.
Bit 4	Effect of window errors (mask value = 8) When this bit is set to one, motion operations that fail during planning due to a window error are ignored (skipped) and program execution continues normally. When this option is selected, each belt-relative motion operation 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 operation to be executed. If no such program has been speci-fied, program execution is halted.

Example

The following example sets the parameter to have bits 1 and 3 set to one (mask values 1 + 4):

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

Related Keywords

3-1-11 BELT on page 3-16 (real-valued function)
3-1-14 BSTATUS on page 3-20 (real-valued function)
3-2-45 PARAMETER on page 3-229 (monitor command)
3-4-92 PARAMETER on page 3-400 (program command)
3-1-75 PARAMETER on page 3-100 (real-valued function)
3-4-139 WINDOW on page 3-464 (program command)
3-1-127 WINDOW on page 3-164 (real-valued Function)

3-5-2 DEVIATION

Adds a path deviation from 1 to 100% to the motion in the singularity region when a robot is in singularity.

Syntax

```
DEVIATION[robot_num] = value
```

Usage Considerations

When the value is set to 0, singularity handling is disabled. When it set to a value between 1 to 100, it deviates from the path.
SCARA robots are not affected by singularity regions.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

The parameter manages problems related to singularities. It detects when the robot is in singularity using and then adds a small deviation to the motion in the singularity region. Once the robot is out of singularity, it does not go back to the original path.

Example

The following example will set deviation to 2% for robot 1.

```
PARAMETER DEVIATION[1] = 2
```

Related Keywords

3-1-75 PARAMETER on page 3-100 (real-valued function)
3-2-45 PARAMETER on page 3-229 (monitor command)
3-4-92 PARAMETER on page 3-400 (program command)

3-5-3 JOG.TIME

System parameter that sets the keep-alive time of a jog operation.

Syntax

```
...JOG.TIME
```

Usage Considerations

The value of the JOG.TIME parameter can be modified only with the PARAMETER monitor command or program command keywords.
If the V+ system is controlling more than one robot, the JOG.TIME parameter controls the keep-alive time of jog operation for all the robots.
The current value of the JOG.TIME parameter can be determined with the PARAMETER monitor command or function keywords.

Details

The Jog program command and monitor command keywords are used to execute a jog move.
The JOG.TIME parameter setting determines the keep-alive time of jog operation. Each time a jog operation is executed, this parameter setting specifies the time the axis or joint moves.
The value for this parameter can range from 0.1 to 5.0 seconds. This parameter is set to 0.3 seconds when the V+ system is initialized.

Example

The following example sets the keep-alive time to 0.5 seconds.

```
PARAMETER JOG.TIME = 0.5
```

Related Keywords

3-4-72 JOG on page 3-371 (program command)
3-2-33 JOG on page 3-212 (monitor command)
3-4-92 PARAMETER on page 3-400 (program command)
3-1-75 PARAMETER on page 3-100 (real-valued function)
3-2-45 PARAMETER on page 3-229 (monitor command)

3-5-4 NOT.CALIBRATED

System parameter that indicates or asserts the calibration status of the robots connected to the system.

Syntax

```
... NOT.CALIBRATED
```

Usage Considerations

The current value of the NOT.CALIBRATED system parameter can be determined with the PARAMETER monitor command or function.
The results indicate that a runtime calibration has been performed with the calibration operation. It does not indicate if calibration offsets have been stored previously.
The parameter name can be abbreviated.

Details

The value of this parameter can range from 0 to 255 and should be interpreted as a bit mask. Bits 1 through 8 correspond to robots 1 through 8. Values have the following interpretations, for example.

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 monitor command and program command attempt to calibrate any enabled robot that has its NOT.CALIBRATED bit ON. When the calibration operation completes, the NOT.CALIBRATED bits are updated.
Consider a system that has only one robot installed. If the CALIBRATE monitor command is issued and it succeeds, the NOT.CALIBRATED is set to 0. If three robots are connected and the CALIBRATE monitor command succeeds in calibrating robots 1 and 2, but not robot 3, 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 ON to force the corresponding robot to be calibrated the next time a CALIBRATE monitor command or program command 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 affected.

- If the new value asserts that a robot is not calibrated, the V+ system behaves as if the robot is not calibrated whether or not the servo software acknowledges that the robot is not calibrated.
- If the new value asserts that a robot is calibrated, V+ tracks the calibrated/ not calibrated state for that robot.

It is normally not meaningful to use a PARAMETER NOT.CALIBRATED statement to turn OFF a bit.

Example

The following example will perform the runtime calibration for all robots if any robot is not calibrated.

```
IF PARAMETER(NOT.CALIBRATED) <> 0 THEN
    CALIBRATE
END
```

Related Keywords

3-2-5 CALIBRATE on page 3-172 (monitor command)
3-4-20 CALIBRATE on page 3-294 (program command)
3-2-45 PARAMETER on page 3-229 (monitor command)
3-4-92 PARAMETER on page 3-400 (program command)
3-1-75 PARAMETER on page 3-100 (real-valued function)
3-6-9 ROBOT on page 3-483

3-5-5 VTIMEOUT

System parameter that sets a timeout value so that an error message is returned if no response is received following a vision command.

Syntax

```
PARAMETER VTIMEOUT = value
```

Details

The statement VTIMEOUT = 0 sets the timeout value to 16 ms. This is the minimum timeout that will be used.
The timeout value is expressed in seconds (0.15 represents 150 ms).

Example

The following example will set a timeout value so an error message will occur if there is no response after 200 ms.

```
PARAMETER VTIMEOUT = 0.20
```

Related Keywords

3-4-92 PARAMETER on page 3-400 (program command)
3-1-75 PARAMETER on page 3-100 (real-valued function)
3-1-123 VLOCATION on page 3-157
3-1-124 VPARAMETER on page 3-160 (real-valued function)
3-4-133 VPARAMETER on page 3-457 (program command)
3-1-125 VRESULT on page 3-161
3-4-134 VRUN on page 3-458
3-1-126 VSTATE on page 3-163
3-4-135 VWAITI on page 3-459

3-6 System Switch Keywords

Use the information in this section to understand system switch keywords and their use with the V+ system.

3-6-1 AUTO.POWER.OFF

This system switch disables high power when certain motion errors occur.

Syntax

```
...AUTO.POWER.OFF[robot_num]
```

Usage Considerations

This system switch functions during automatic mode but not during manual mode. It is especially useful in reducing operator intervention during common nulling-timeout and envelope errors.
If a parameter is not specified while using this switch with a monitor command, all robots in the system will be affected.
This system switch is disabled by default.

Parameters

Parameter	Description
robot_num	Required real value, variable, or expression interpreted as an integer that indicates the number of the robot affected.

Details

When this switch is enabled, V+ disables high power for all associated motion errors that occur on the specified robot_num. The associated motion errors are listed below.

- Soft Envelope Error
- Soft Overspeed Error
- Force Protect Limit Exceeded

Example

The following example instructs V+ to disable high power to robot 1 when associated motion errors occur.

```
ENABLE AUTO.POWER.OFF[1]
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-6-2 CP

System switch that controls the continuous-path function of a robot.

Syntax

```
... CP[robot_num]
```

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

The CP switch can be used to turn OFF continuous-path motion processing. This switch is ON when the V+ system is initialized.
Refer to *V+ User's Manual (Cat. No. I671)* for more information about continuous-path trajectories.

Example

The following example turns OFF continuous-path motion processing for all robots in the system.

```
DISABLE CP
```

Related Keywords

3-4-28 CPOFF on page 3-308 (program command)
3-4-29 CPON on page 3-309 (program command)
3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)

3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-6-3 DECEL.100

System switch that enables or disables the maximum deceleration of 100% for the ACCEL program command keyword.

Syntax

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

Usage Considerations

When DECEL.100 is enabled for a selected robot, the maximum deceleration percentage is used to limit the value specified by the ACCEL program command keyword.
DECEL.100is disabled for all robots by default.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Example

The following example will cause the ACCEL program command keyword to use 100% for the maximum deceleration on robot 2.

```
DECEL.100[2]
```

Related Keywords

3-4-3 ACCEL on page 3-270 (program command)
3-1-2 ACCEL on page 3-8 (real-valued function)
3-2-55 SPEED on page 3-240 (monitor command)
3-4-124 SPEED on page 3-445 (program command)

3-6-4 DELAY.IN.TOL

System switch that controls the timing of coarse or fine nulling after the V+ system completes a motion segment.

Syntax

```
... DELAY.IN.TOL [robot_num]
```

Usage Considerations

For many applications, enabling this switch produces the best nulling behavior. However, the switch should be disabled for backward compatibility with previous V+ systems.
Enabling this switch will have no difference in operation when used in emulation mode. The trajectory of the robot in the emulation mode is considered to be the same as the current target position and no positioning error occurs when the operation is completed.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected. If the index is omitted or zero in an ENABLE or DISABLE keyword, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

The DELAY.IN.TOL system switch is disabled by default for all SCARA robots.
If the switch is enabled, the final commanded location reached by the robot is more precise,but the overall time to complete the motion is increased. When the robot has reached the end- point of the motion segment within the specified FINE / COARSE tolerance specified, the system will add extra delay (typically 2-3 milliseconds). During this time, the system is waiting for the commanded velocity to reach 0 while the robot remains within the specified tolerance before the next motion can begin. This function ensures that the robot has reached the desired location within the specified FINE / COARSE tolerance and has come to a complete stop before proceeding to the next motion segment.
If the switch is disabled and the robot has reached the final commanded location within the specified FINE / COARSE tolerance, the next motion is immediately commanded without delay.
Changing the state of this switch during a move will not affect the current robot motion.

Example

The following example will enable the DELAY.IN.TOL system switch for robot 2.

```
ENABLE DELAY.IN.TOL[2]
```

Related Keywords

3-4-26 COARSE on page 3-305
3-4-54 FINE on page 3-344

3-4-88 NULL on page 3-396
3-4-86 NONULL on page 3-393

3-6-5 DRY.RUN

System switch that controls whether or not V+ communicates with the robot.

Syntax

```
... DRY.RUN[robot_num]
```

Usage Considerations

The DRY.RUN System Switch can be enabled or disabled by an application program, but the new setting of the switch does not take effect until the next time any of the following events occur:

- An EXECUTE keyword is processed for task 0
- The robot is attached with an ATTACH keyword
- A CALIBRATE keyword is executed

Before an application program changes the setting of the DRY.RUN switch, the program must have the robot detached. Otherwise, a -602 *Robot already attached to program* error results when the attempt is made to change the switch setting.

⚠ WARNING

Digital and analog I/O are not affected by DRY.RUN, so external devices driven by analog or digital output commands still operate.

Precautions for Correct Use

Do not allow multiple tasks to change DRY.RUN simultaneously, since the DRY.RUN state can then be different from that expected. Your programs should use a software interlock in this case.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

The DRY.RUN switch is initially disabled. This system switch can be used to stop V+ from sending motion commands to the robot and expecting position information back from the robot. 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 concerns about the robot collisions.
The pendant can still be used to control the robot while the system is in DRY.RUN mode.
The DRY.RUN switch is examined whenever a robot is attached. 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.

Example

The following example enables the DRY.RUN system switch for robot 2.

```
ENABLE DRY.RUN[2]
```

Related Keywords

3-2-18 DISABLE on page 3-190
3-2-20 ENABLE on page 3-193
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-6-6 MESSAGES

System switch to enable or disable output to the Monitor Window from TYPE program commands.

Syntax

```
... MESSAGES
```

Details

If this system switch is enabled, output from TYPE program commands are displayed on the Monitor Window. Otherwise, output is suppressed.
This switch is enabled by default.

Example

The following example will enable the MESSAGES system switch if it is disabled and then display "Messages are now enabled" in the Monitor Window.

```
IF (SWITCH(MESSAGES)==FALSE)THEN
    ENABLE MESSAGES
    TYPE "Messages now enabled"
END
```

Related Keywords

3-4-37 DISABLE on page 3-320 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-4-43 ENABLE on page 3-327 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-4-126 SWITCH on page 3-449 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)
3-4-130 TYPE on page 3-453

3-6-7 OBSTACLE

System switch that enables or disables up to four obstacles for a selected robot.

Syntax

```
...OBSTACLE[switch_ID]
```

Usage Considerations

You must specify a robot with the SELECT program command keyword before manipulating the OBSTACLE system switch.
If the switch_ID parameter value is not included in the OBSTACLE statement or the value specified is not a defined obstacle, an *Invalid argument* (-407) error will occur.
Obstacles must be configured using the Sysmac Studio or ACE software. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information.

Parameters

Parameter	Description
switch_ID	Required real value, variable, or expression interpreted as an integer that indicates the obstacle to enable or disable. Only values of 1 to 4 are permitted.

Details

Up to four obstacles are available for each robot configured in the system. Each obstacle can be enabled or disabled with the OBSTACLE system switch parameter switch_ID. Use the examples below to understand how to enable and disable obstacles associated with each robot in the system.
The SWITCH keyword can be used to read the state of obstacles for a selected robot. Use the examples below to understand how to read the state of obstacles associated with each robot in the system.

Additional Information

Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for information about configuring obstacles.

Example

The following program command example will enable obstacle 1 for robot 2.

```
SELECT ROBOT = 2
SWITCH OBSTACLE [1] = TRUE
```

The following program command example will enable obstacle 1 and obstacle 2 for robot 2.

```
SELECT ROBOT = 2
ENABLE OBSTACLE [1], OBSTACLE [2]
```

The following program command example will enable obstacle 1 for robot 1 and disable obstacle 1 for robot 2.

```
SELECT ROBOT = 1
ENABLE OBSTACLE [1]
SELECT ROBOT = 2
DISABLE OBSTACLE [1]
```

The following real-valued function example will return the state to variable "VAR" of obstacle2 for robot 2.

```
SELECT ROBOT = 2
VAR = SWITCH(OBSTACLE [2])
```

The following monitor command example will list all obstacle states for all robots in the Monitor Window.

```
SWITCH OBSTACLE
```

The following monitor command example will list the state of obstacle 1 for robot #2 in the Monitor Window.

```
SELECT ROBOT = 2
SWITCH OBSTACLE [1]
```

Related Keywords

3-4-43 ENABLE on page 3-327 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-18 DISABLE on page 3-190 (monitor command)
3-4-115 SELECT on page 3-431 (program command)
3-2-53 SELECT on page 3-237 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)

3-6-8 POWER

System Switch that controls or monitors the status of the robot high power.

Syntax

```
... POWER[robot_num]
```

Usage Considerations

Before an application program changes the setting of the POWER switch, the pro- gram must have the robot detached. Otherwise, a -314 error *Switch can't be enabled* error results when the attempt is made to change the switch setting.

⚠ DANGER

Do not use the POWER switch to enable power from within a program unless your system is subject to European certification. With European certification, special safety features are built-in to the system to prevent the robot from being activated without warning.

⚠ DANGER

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 you do not have a clear view of the robot workspace or do not have immediate access to an emergency stop button.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

Enabling this switch is equivalent to pushing the COMP/PWR button on the pendant to turn ON high power. If there is no error condition that prevents power from turning ON, the enabling process proceeds to the second step below, in which you must press the HIGH POWER button on the Front Panel. Systems not subject to European certification 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 the following steps.

1. Decelerating all robots according to the user-specified parameters. See the following Note.
2. Turning ON the brakes.
3. Waiting for the user-specified brake-delay interval. See the following Note.
4. Turning OFF the amplifiers and power.
5. Asserting the backplane Emergency Stop signal and deasserting the High Power Enable (HPE) signal.

DISABLE POWER may take an arbitrarily long time due to long deceleration times and long brake turn-on delays. Use the ESTOP keyword when an immediate shutdown is necessary.. 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 must either be in DRY.RUN mode or detach the robot from pro- gram control. See the DRY.RUN System Switch or DETACH program command keyword for details.

Example

The following program example detaches the robot and turns high power OFF.

```
DETAC
DISABLE POWER[2]
```

Related Keywords

3-4-36 DETACH on page 3-318
3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-6-5 DRY.RUN on page 3-478
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-4-45 ESTOP on page 3-330 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-value function)

3-6-9 ROBOT

System switch that enables or disables one robot or all robots.

Syntax

```
... ROBOT [robot_num]
```

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.

The ROBOT switches may be modified only for robots that are present and that started up without a fatal error.

The settings of these switches can be checked at any time with the SWITCH keyword monitor command or real-valued function to determine which robots are enabled.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

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 System Switches are enabled. After start up, the ROBOT switches can be used to selectively disable robots. This can aid in troubleshooting of individual robots.

Motion operations should not be executed for a robot that has been disabled.

When a robot is disabled by use of the ROBOT switch, that robot is bypassed when:

- Power is enabled for all robots with the COMP/PWR button on the pendant or with the POWER System Switch.
- All the robots are calibrated with the CALIBRATE keyword.

Example

The following example disables robot 2

```
DISABLE ROBOT[2]
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)

3-6-10 SCALE.ACCEL

System switch that enables or disables the scaling of acceleration and deceleration as a function of program speed.

Syntax

```
... SCALE.ACCEL [robot_num]
```

Usage Considerations

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.
This system switch will affect the scaling of acceleration and deceleration as long as the program speed is below the SCALE.ACCEL Upper Limit. Refer to *Sysmac Studio for Robot Integrated System Building Function with Robot Integrated CPU Unit Operation Manual (Cat. No. W595)* / *Automation Control Environment (ACE) Version 4 User's Manual (Cat. No. I633)* for more information about this setting.

⚠ 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.

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the parameter is omitted or 0, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

This switch is enabled when the V+ system is initialized.
When this switch is enabled and the program speed is below the preset threshold value, the effective acceleration and deceleration for that robot are calculated as follows where acceleration_setting and deceleration_setting are values set by the ACCEL keyword.

```
effective acceleration = program_speed * acceleration_setting
effective deceleration = program_speed * deceleration_setting
```

For example, if program speed 50% is specified and the threshold value is 150, the effective acceleration and deceleration are 50% of the current settings. If the program speed is higher than 150% with the threshold set to 150, the current acceleration and deceleration are used without modification.
All robots have the SCALE.ACCEL speed threshold set by default to a very large value (1e16), effectively forcing the scaling of accelerations and deceleration for all speeds when this switch is enabled.
If the SCALE.ACCEL switch is disabled for a robot, accelerations and decelerations are not scaled based on the program speed. In this case, accelerations and decelerations are higher than normal at reduced speeds. This is particularly noticeable at very slow speeds and as a result, robot motions may appear to be more rough or jerky.

Example

Turn OFF acceleration scaling for robot 2:

```
DISABLE SCALE.ACCEL[2]
```

Related Keywords

3-4-3 ACCEL on page 3-270 (program command)
3-1-2 ACCEL on page 3-8 (real-valued function)
3-2-55 SPEED on page 3-240 (monitor command)
3-1-96 SPEED on page 3-118 (real-valued function)
3-6-11 SCALE.ACCEL.ROT on page 3-486

3-6-11 SCALE.ACCEL.ROT

System switch that specifies whether or not the SCALE.ACCEL switch takes into account the Cartesian rotational speed during straight-line motions.

Syntax

```
... SCALE.ACCEL.ROT [robot_num]
```

Parameters

Parameter	Description
robot_num	Optional real value, variable, or expression interpreted as an integer that indicates the number of the robot affected. If the index is omitted or zero in an ENABLE or DISABLE keyword, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected.

Details

If SCALE.ACCEL.ROT is enabled for a selected robot, the lesser of the Cartesian linear and rotational speeds is used to scale acceleration and deceleration during straight-line motions. 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

The following example will cause SCALE.ACCEL not to use Cartesian rotational speed for robot 2:

```
DISABLE SCALE.ACCEL.ROT[2]
```

Related Keywords

3-4-3 ACCEL on page 3-270 (program command)
3-1-2 ACCEL on page 3-8 (real-valued function)
3-2-55 SPEED on page 3-240 (monitor command)
3-1-96 SPEED on page 3-118 (real-valued function)

3-6-12 UPPER

System switch that controls whether or not the case of each character is ignored when string comparisons are performed.

Syntax

```
... UPPER
```

Usage Considerations

This system switch value is shared globally by all program tasks. If you change the value in one task, it affects comparisons in all other tasks. Do not change this switch during normal program execution.

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. When UPPER is enabled, both of the following comparisons return a TRUE value.

```
"a" == "A"    and    "A" == "A"
```

When UPPER is disabled, the case of characters is considered during string comparisons. For example, the comparison on the left in the statements above returns a FALSE value while the comparison on the right returns a TRUE value.

UPPER is enabled by default so that string comparisons are performed without considering the case of the characters.

The STRDIF function always compares strings considering their case. You can leave UPPER enabled always and then use STRDIF in situations where case is important.

Example

The following example will display a "0" (true) in the Monitor Window.

```
$s1="Case"
$s2="case"

DISABLE UPPER
a= $s1==$s2
TYPE a
```

The following example will display a "-1" (false) in the Monitor Window.

```
$s1="Case"
$s2="case"

ENABLE UPPER
a= $s1==$s2
TYPE a
```

Related Keywords

3-2-18 DISABLE on page 3-190 (monitor command)
3-4-37 DISABLE on page 3-320 (program command)
3-2-20 ENABLE on page 3-193 (monitor command)
3-4-43 ENABLE on page 3-327 (program command)
3-2-65 SWITCH on page 3-256 (monitor command)
3-4-126 SWITCH on page 3-449 (program command)
3-1-102 SWITCH on page 3-129 (real-valued function)
3-1-101 STRDIF on page 3-128

THIS DOCUMENT CONTAINS IMPORTANT SAFETY INSTRUCTIONS. SAVE THIS DOCUMENT.

OMRON Corporation Industrial Automation Company

Kyoto, JAPAN **Contact : www.ia.omron.com**

Regional Headquarters

OMRON EUROPE B.V.
Wegalaan 67-69, 2132 JD Hoofddorp
The Netherlands
Tel: (31) 2356-81-300 Fax: (31) 2356-81-388

OMRON ASIA PACIFIC PTE. LTD.
438B Alexandra Road, #08-01/02 Alexandra
Technopark, Singapore 119968
Tel: (65) 6835-3011 Fax: (65) 6835-3011

OMRON ELECTRONICS LLC
2895 Greenspoint Parkway, Suite 200
Hoffman Estates, IL 60169 U.S.A.
Tel: (1) 847-843-7900 Fax: (1) 847-843-7787

OMRON ROBOTICS AND SAFETY TECHNOLOGIES, INC.
4225 Hacienda Drive, Pleasanton, CA 94588 U.S.A.
Tel: (1) 925-245-3400 Fax: (1) 925-960-0590

OMRON (CHINA) CO., LTD.
Room 2211, Bank of China Tower, 200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-6023-0333 Fax: (86) 21-5037-2388

Authorized Distributor:

Cat. No. I672-E-04 0825 (0122)

28319-000 D