EPSON

EPSON RC+ Ver.4.2

SPEL+ Language Reference

Rev.1 EM073S1477F

EPSON RC+ Ver.4.2 SPEL^{+} Language Reference Rev.1

EPSON RC+ Ver.4.2

SPEL+ Language Reference

Rev.1

FOREWORD

Thank you for purchasing our robot products. This manual contains the information necessary for the correct use of the EPSON RC+ software.

Please carefully read this manual and other related manuals when using this software.

Keep this manual in a handy location for easy access at all times.

WARRANTY

The robot and its optional parts are shipped to our customers only after being subjected to the strictest quality controls, tests and inspections to certify its compliance with our high performance standards.

Product malfunctions resulting from normal handling or operation will be repaired free of charge during the normal warranty period. (Please ask your Regional Sales Office for warranty period information.)

However, customers will be charged for repairs in the following cases (even if they occur during the warranty period):

1. Damage or malfunction caused by improper use which is not described in the manual, or careless use.

2. Malfunctions caused by customers' unauthorized disassembly.

3. Damage due to improper adjustments or unauthorized repair attempts.

4. Damage caused by natural disasters such as earthquake, flood, etc.

Warnings, Cautions, Usage:

1. If the robot or associated equipment is used outside of the usage conditions and product specifications described in the manuals, this warranty is void.

2. If you do not follow the WARNINGS and CAUTIONS in this manual, we cannot be responsible for any malfunction or accident, even if the result is injury or death.

3. We cannot foresee all possible dangers and consequences. Therefore, this manual cannot warn the user of all possible hazards.

TRADEMARKS

Microsoft, Windows, Windows logo, Visual Basic, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Other brand and product names are trademarks or registered trademarks of the respective holders.

TRADEMARK NOTATION IN THIS MANUAL

Microsoft® Windows® 2000 Operating system
Microsoft® Windows® XPOperating system
Throughout this manual, Windows 2000, and Windows XP refer to above respective operating systems. In some cases, Windows refers generically to Windows 2000, and Windows XP.

NOTICE

The content of this manual is subject to change without notice.

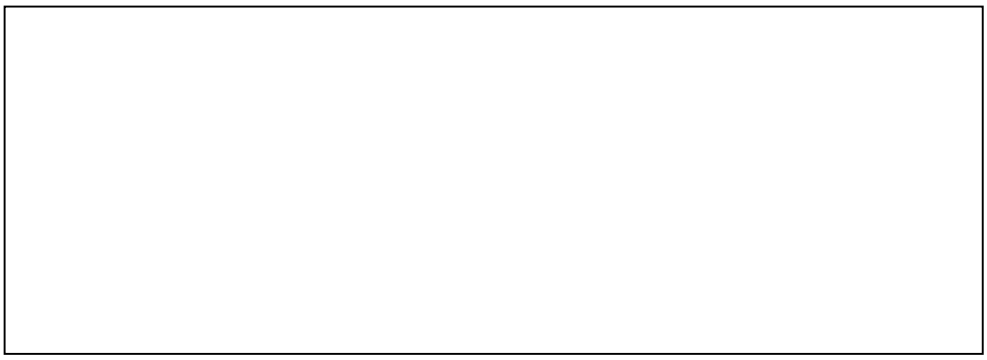
We ask that you please notify us if you should find any errors in this manual or if you have any comments regarding its content.

INQUIRIES

Contact the following service center for robot repairs, inspections or adjustments.
If service center information is not indicated below, please contact the supplier office for your region.
Please prepare the following items before you contact us.

- Your controller model and its serial number
- Your manipulator model and its serial number
- Software and its version in your robot system
- A description of the problem

SERVICE CENTER



MANUFACTURER & SUPPLIER

Japan & Others

SEIKO EPSON CORPORATION
Suwa Minami Plant
Factory Automation Systems Dept.
1010 Fujimi, Fujimi-machi,
Suwa-gun, Nagano, 399-0295
JAPAN
TEL : +81-(0)266-61-1802
FAX : +81-(0)266-61-1846

SUPPLIERS

North & South America

EPSON AMERICA, INC.
Factory Automation/Robotics
18300 Central Avenue
Carson, CA 90746
USA
TEL : +1-562-290-5900
FAX : +1-562-290-5999
E-MAIL : info@robots.epson.com

Europe

EPSON DEUTSCHLAND GmbH
Factory Automation Division
Otto-Hahn-Str.4
D-40670 Meerbusch
Germany
TEL : +49-(0)-2159-538-1391
FAX : +49-(0)-2159-538-3170
E-MAIL : robot.infos@epson.de

SAFETY PRECAUTIONS

Installation of robots and robotic equipment should only be performed by qualified personnel in accordance with national and local codes. Please carefully read this manual and other related manuals when using this software.
Keep this manual in a handy location for easy access at all times.

⚠ WARNING	■ This symbol indicates that a danger of possible serious injury or death exists if the associated instructions are not followed properly.
⚠ CAUTION	■ This symbol indicates that a danger of possible harm to people or physical damage to equipment and facilities exists if the associated instructions are not followed properly.

Table of Contents

Summary of SPEL+ Commands

The following is a summary of SPEL+ commands.

System Management Commands

Reset	Reset the controller.
Stat	Returns controller status bits.
Ver	Displays controller setup.
Date	Sets the system date.
Time	Sets system time.
Date$	Returns the system date.
Time$	Returns system time as string.
Hour	Displays / returns controller operation time.
Time	Sets system time.
Errhist	Displays error history.
ShutDown	Shuts down RC+ and Windows.

Robot Control Commands

Calib	Calibrates motors.
CalPls	Sets / returns calibration values.
Hofs	Sets / returns offset pulses between encoder origin and home sensor.
Mcal	Calibrates incremental encoder robots.
MCordr	Sets / returns joint order of Mcal.
Mcorg	Automatically calculates calibration parameters for Mcal.
MCofs	Sets / returns MCal parameters.
HTest	Displays / returns pulse count from home sensor to encoder Z phase.

Power	Sets / returns servo power mode.
Motor	Sets / returns motor status.
SFree	Removes servo power from the specified servo axis.
SLock	Restores servo power to the specified servo axis.
Jump	Jump to a point using point to point motion.
Jump3	Jump to a point using 3D gate motion.
Jump3Cp	Jump to a point using 3D motion in continuous path.
Arch	Sets / returns arch parameters for Jump motion.
LimZ	Sets the upper Z limit for the Jump command.
Sense	
JS	Returns status of Sense operation.
JT	Returns the status of the most recent Jump command for the current robot.
Go	Move the robot to a point using point to point motion.
Pass	Executes simultaneous four joint Point to Point motion, passing near but not through the specified points.
Pulse	Move the robot to a position defined in pulses.
BGo	Executes Point to Point relative motion, in the selected local coordinate system.
BMove	Executes linear interpolation relative motion, in the selected local coordinate system.
TGo	Executes Point to Point relative motion, in the current tool coordinate system.
TMove	Executes linear interpolation relative motion, in the selected tool coordinate system.
Till	Specifies motion stop when input occurs.
!…!	Process statements during motion.
Speed	Sets / returns speed for point to point motion commands.
Accel	Sets / returns acceleration and deceleration for point to point motion.
Weight	Specifies or displays the inertia of the robot arm.
Arc	Moves the arm using circular interpolation.
Arc3	Moves the arm in 3D using circular interpolation.
Move	Move the robot using linear interpolation.
CV Move	Performs the continuous spline path motion defined by the Curve instruction.
SpeedS	Sets / returns speed for linear motion commands.
AccelS	Sets / returns acceleration and deceleration for linear motion.
SpeedR	Sets / returns speed for tool rotation.
AccelR	Sets / returns acceleration and deceleration for tool rotation.
Home	Move robot to user defined home position.
HomeSet	Sets user defined home position.
Hordr	Sets motion order for Home command.

InPos	Checks if robot is in position (not moving).
CurPos	Returns current position while moving.
TCPSpeed	Returns calculated current tool center point velocity.
Pallet	Defines a pallet or returns a pallet point.
Fine	Sets positioning error limits.
QP	Sets / returns Quick Pause status.
JRange	Sets / returns joint limits for one joint.
Range	Sets limits for all joints.
XYLim	Sets / returns Cartesian limits of robot envelope.
PTPBoost	Sets / returns boost values for small distance PTP motion.
CX	Sets / returns the X axis coordinate of a point.
CY	Sets / returns the Y axis coordinate of a point.
CZ	Sets / returns the Z axis coordinate of a point.
CU	Sets / returns the U axis coordinate of a point.
CV	Sets / returns the V axis coordinate of a point.
CW	Sets / returns the W axis coordinate of a point.
Pls	Returns the pulse value of one joint.
Agl	Returns joint angle at current position.
PAgl	Return a joint value from a specified point.
PPls	Return the pulse position of a specified joint value from a specified point.
Robot	Sets / returns the current robot.

Input / Output Commands

On	Turns an output on.
Off	Turns an output off.
Oport	Reads status of one output bit.
Sw	Returns status of input.
In	Reads 8 bits of inputs.
InW	Returns the status of the specified input word port.
InBCD	Reads 8 bits of inputs in BCD format.
Out	Sets / returns 8 bits of outputs.
OutW	Simultaneously sets 16 output bits.
OpBCD	Simultaneously sets 8 output bits using BCD format.
MemOn	Turns a memory bit on.
MemOff	Turns a memory bit off.
MemSw( )	Returns status of memory bit.
MemIn( )	Reads 8 bits of memory I/O.
MemOut	Sets / returns 8 memory bits.
ZeroFlg	Returns value of memory I/O previous to it last being switched on or off.
Wait	Wait for condition or time.

TMOut	Sets default time out for Wait statement.
Tw	Returns the status of the Wait condition and Wait timer interval.
Input	Input one or more variables from current display window.
Print	Display characters on current display window.
Line Input	Input a string from the current display window.
Input #	Input one or more variables from file or port.
Print #	Output characters to a file or port.
Line Input #	Input a string from a file or port.
Lof	Returns the number of lines in a communications buffer.
ClrScr	Clears the *EPSON RC+* Run or Operator window text area.
EPrint	Ends a print sequence of LPrint statements and sends to printer.
InputBox	Display input box and return user reply.
LPrint	Sends a line to the default printer.
MsgBox	Display message to user.

Coordinate Change Commands

Arm	Sets / returns current arm.
ArmSet	Defines an arm.
ArmClr	Clears an arm definition.
Tool	Sets / returns the current tool number.
TLSet	Defines or displays a tool coordinate system.
TLClr	Clears a tool definition.
ECP	Sets / returns the current ECP number
ECPSet	Defines or displays an external control point.
ECPClr	Clears an ECP definition
Local	Define a local coordinate system.
LocalClr	Clears (undefines) a local coordinate system.
Base	Defines and displays the base coordinate system.
Elbow	Sets / returns elbow orientation of a point.
Hand	Sets / returns hand orientation of a point.
Wrist	Sets / returns wrist orientation of a point.
J4Flag	Sets / returns the J4Flag setting of a point.
J6Flag	Sets / returns the J6Flag orientation of a point.

Program Control Commands

Function	Declare a function.
For...Next	Executes one or more statements for a specific count.
GoSub	Execute a subroutine.
Return	Returns from a subroutine.

GoTo	Branch unconditionally to a line number or label.
Call	Call a user function.
If..Then..Else..EndIf	Conditional statement execution
Else	Used with the If instruction to allow statements to be executed when the condition used with the If instruction is False. Else is an option for the If/Then instruction.
Select ... Send	Executes one of several groups of statements, depending on the value of an expression.
While...Wend	Executes specified statements while specified condition is satisfied
Do...Loop	Do...Loop construct.
Trap	Specify a trap handler.
OnErr	Defines an error handler.
Era	Returns robot joint number for last error.
Erf$	Returns the function name for last error.
Erl	Returns line number of error.
EClr	Clears the current error.
Err	Returns error number.
Ert	Returns task number of error.
ErrMsg$	Returns error message.
Signal	Sends a signal to tasks executing WaitSig.
SyncLock	Synchronizes tasks using a mutual exclusion lock.
SynUnlock	Unlocks a sync ID that was previously locked with SyncLock.
WaitSig	Waits for a signal from another task.

Program Execution Commands

Xqt	Execute a task.
Pause	Pause all tasks that have pause enabled.
Cont	Continue after pause.
Halt	Suspend a task.
Quit	Quits a task.
Resume	Resume a task in the halt state.
MyTask	Returns current task.
Chain	Stops all tasks and starts a program group.
Restart	Stops all tasks and restarts the current program group.

Pseudo Statements

#define	Defines a macro.
#ifdef ... #endif	Conditional compile.
#ifndef ... #endif	Conditional compile.
#include	Include a file.

File Management Commands

Dir	Displays a directory.
ChDir	Change the current directory.
MkDir	Make directory.
Rmdir	Removes a directory.
Rendir	Rename a directory.
FileDateTime$	Returns the date and time of a file.
FileExists	Checks if a file exists.
FileLen	Returns the length of a file.
FolderExists	Checks if a folder exists.
LoadPoints	Load points for the current robot.
SavePoints	
Type	Displays the contents of a text file.
Kill	Deletes a file.
Del	Deletes a file.
Copy	Copy a file.
Rename	Rename a file.
AOpen	Opens a file for append.
BOpen	Opens a file for binary access.
ROpen	Opens a file for read only access.
UOpen	Opens a file for read / write access.
WOpen	Opens a file for write access.
Read	Reads one or more characters from file or port.
Seek	Move a file pointer for random access.
Write	Writes one or more characters to a file or port.
Close	Close a file.

Numeric Value Commands

Oport	Reads status of one output bit.
Sw	Returns status of input.
In	Reads 8 bits of inputs.
InBCD	Reads 8 bits of inputs in BCD format.
Sw($)	Returns status of memory bit.
In($)	Reads 8 bits of memory I/O.
ZeroFlg	Returns value of memory I/O previous to it last being switched on or off.
Ctr	Return the value of a counter.
CTReset	Resets a counter.
Tmr	Returns the value of a timer.
TmReset	Resets a timer to 0.
Time	Sets system time.

Js	Returns status of Sense operation.
CX	Sets / returns the X axis coordinate of a point.
CY	Sets / returns the Y axis coordinate of a point.
CZ	Sets / returns the Z axis coordinate of a point.
CU	Sets / returns the U axis coordinate of a point.
CV	Sets / returns the V axis coordinate of a point.
CW	Sets / returns the W axis coordinate of a point.
Pls	Returns the pulse value of one joint.
Agl	Returns joint angle at current position.
Era	Returns robot joint number for last error.
Erf$	Returns function name for last error.
Erl	Returns line number for last error.
Err	Returns error number for last error.
Ert	Returns task number for last error.
TW	Returns the status of the Wait condition and Wait timer interval.
MyTask	Returns current task.
Lof	Returns the number of lines in a communications buffer.
Stat	Returns controller status bits.
Sin	Returns the sine of an angle.
Cos	Returns cosine of an angle.
Tan	Returns the tangent of an angle.
Acos	Returns arccosine.
Asin	Returns arcsine.
Atan	Returns arctangent.
Atan2	Returns arctangent based on X, Y position.
Sqr	Returns the square root of a number.
Abs	Returns the absolute value of a number.
Sgn	Returns the sign of a number.
Int	Converts a real number to an integer.
Not	Not operator.
LShift	Shifts bits to the left.
RShift	Shifts bits to the right.

String Commands

String	Declares string variables.
Asc	Returns the ASCII value of a character.
Chr$	Returns the character of a numeric ASCII value.
Left$	Returns a substring from the left side of a string.
Mid$	Returns a substring.
Right$	Returns a substring from the right side of a string.

Len	Returns the length of a string.
LSet$	Returns a string padded with trailing spaces.
RSet$	Returns a string padded with leading spaces.
Space$	Returns a string containing space characters.
Str$	Converts a number to a string.
Val	Converts a numeric string to a number.
ErrMsg$	Returns error message.
LCase$	Converts a string to lower case.
UCase$	Converts a string to upper case.
LTrim$	Removes spaces from beginning of string.
RTrim$	Removes spaces from end of string.
Trim$	Removes spaces from beginning and end of string.
ParseStr	Parse a string and return array of tokens.
FmtStr	Format a number or string.

Logical operators

And	Performs logical and bitwise AND operation.
Or	Or operator.
LShift	Shifts bits to the left.
Mod	Modulus operator.
Not	Not operator.
RShift	Shifts bits to the right.
Xor	Exclusive Or operator.

Variable commands

Boolean	Declares Boolean variables.
Byte	Declares byte variables.
Double	Declares double variables.
Global	Declares global variables.
Globals	Displays Global variables in memory
Integer	Declares integer variables.
Long	Declares long integer variables.
Real	Declares real variables.
String	Declares string variables.

Commands used with VB Guide

SPELCom_Event	Fire an event in ActiveX client.
SPELCom_Return	Set function return value for ActiveX client.

Ethernet I/O commands

ENetIO_AnaGetConfig	Returns analog input or output module configuration values.
ENetIO_AnaIn	Reads one analog input or output channel.
ENetIO_AnaOut	Sets the value for one analog output.
ENetIO_AnaSetConfig	Configures analog input or output module.
ENetIO_ClearLatches	Clears both the on and off digital latches for one input point.
ENetIO_DigGetConfig	Returns digital input configuration.
ENetIO_DigSetConfig	Sets digital input configuration.
ENetIO_In	Reads one digital input or output module (4 points).
ENetIO_IONumber	Returns the bit number of the specified Ethernet I/O label.
ENetIO_Off	Turns off one digital output.
ENetIO_On	Turns on one digital output.
ENetIO_Oport	Returns the status of one digital output.
ENetIO_Out	Sets one digital output module (4 points).
ENetIO_ReadCount	Reads digital input counter.
ENetIO_Sw	Returns the status of one digital input.
ENetIO_SwLatch	Returns the status of one digital input on or off latch.

Fieldbus I/O commands

FbusIO_GetBusStatus	Returns the status of specified fieldbus.
FbusIO_GetDeviceStatus	Returns the status of specified fieldbus device.
FbusIO_In	Returns the status of an 8 bit input port.
FbusIO_InW	Returns the status of a 16 bit input port.
FBusIO_IONumber	Returns the bit number of the specified Fieldbus I/O label.
FbusIO_Off	Turns off one output.
FbusIO_On	Turns on one output.
FbusIO_Oport	Returns the status of one output.
FbusIO_Out	Sets one output byte.
FbusIO_OutW	Sets one output word.
FbusIO_Sw	Returns the status of one input.
FbusIO_SendMsg	Sends an explicit message to a device and returns the reply.

Force sensing commands

Force_Calibrate	Zeros out all axes for the current sensor.
Force_ClearTrigger	Clears all trigger conditions for the current sensor.
Force_GetForce	Returns the current value for one axis for the current sensor.
Force_GetForces	Returns the current values for all axes for the current sensor in an array.
Force_Sensor	Sets / returns the current sensor for the current task.
Force_SetTrigger	Sets the force limit trigger for motion commands for the current sensor.
Force_TC	Sets / returns the current torque control mode for the current robot.
Force_TCLim	Sets / returns the current torque limits for the current robot.
Force_TCSpeed	Sets / returns the current speed after torque limit is reached for the current robot.

Security commands

GetCurrentUser$	Returns the current EPSON RC+ user.
Login	Log into EPSON RC+ as another user.

Conveyor Tracking Commands

Cnv_AbortTrack	Aborts a motion command to a conveyor queue point.
Cnv_ Downstream	Returns the downstream limit of a conveyor.
Cnv_Fine	Sets / returns the value of Cnv_Fine for one conveyor.
Cnv_Name$	Returns the name of the specified conveyor.
Cnv_Number	Returns the number of a conveyor specified by name.
Cnv_Point	Returns a robot point in the specified conveyor's coordinate system derived from sensor coordinates.
Cnv_PosErr	Returns deviation in current tracking position compared to tracking target.
Cnv_Pulse	Returns the current position of a conveyor in pulses.
Cnv_QueAdd	Adds a robot point to a conveyor queue.
Cnv_QueGet	Returns a point from the specified conveyor's queue.

Cnv_QueLen	Returns the number of items in the specified conveyor's queue.
Cnv_QueList	Displays a list of items in the specified conveyor's queue.
Cnv_QueMove	Moves data from upstream conveyor queue to downstream conveyor queue.
Cnv_QueReject	Sets / returns and displays the queue reject distance for a conveyor.
Cnv_QueRemove	Removes items from a conveyor queue.
Cnv_QueUserData	Sets / returns and displays user data associated with a queue entry.
Cnv_RobotConveyor	Returns the conveyor being tracked by a robot.
Cnv_Speed	Returns the current speed of a conveyor.
Cnv_Trigger	Latches current conveyor position for the next Cnv_QueAdd statement.
Cnv_ Upstream	Returns the Upstream limit of a conveyor

Operator Pendant Commands

OP_Cls	Clears the OP (Operator Pendant) user page.
OP_Inkey	Waits for OP button to be pressed and returns button number.
OP_Input	Gets input from operator using OP keypad.
OP_Key	Returns status of the OP user buttons and touch screen switches.
OP_Page	Sets the current OP page.
OP_Print	Prints text on the OP Run and User pages.

SPEL+ Language Reference

This section describes each SPEL+ command as follows:

Syntax	Syntax describes the format used for each command. For some commands, there is more than one syntax shown, along with a number that is referenced in the command description. Parameters are shown in italics.
Parameters	Describes each of the parameters for this command.
Return Values	Describes any values that the command returns.
Description	Gives details about how the command works.
Notes	Gives additional information that may be important about this command.
See Also	Shows other commands that are related to this command. Refer to the Table of Contents for the page number of the related commands.
Example	Gives one or more examples of using this command.

SYMBOLS

This manual uses the following symbols to show what context the command can be used in:

>	May be used as a monitor mode command.
S	May be used as a statement in a SPEL$^+$ program.
F	May be used as a Function in a SPEL$^+$ program.
INC	Dedicated for INC robot equipment with incremental encoder. When executing for Abs robot equipped with absolute encoder, Error 123 will occur.

!...! Parallel Processing

> S

Processes input/output statements in parallel with motion.

Syntax

motion cmd **!***statements* **!**

Parameters

motion cmd	Any valid motion command included in the following list: Arc, Arc3, Go, Jump, Jump3, Jump3CP, Move, Pulse.
statements	Any valid parallel processing I/O statement(s) which can be executed during motion. (See table below)

Description

Parallel processing commands are attached to motion commands to allow I/O statements to execute simultaneously with the beginning of motion travel. This means that I/O can execute while the arm is moving rather than always waiting for arm travel to stop and then executing I/O. There is even a facility to define when within the motion that the I/O should begin execution. (See the Dn parameter described in the table below.)

The table below shows all valid parallel processing statements. Each of these statements may be used as single statements or grouped together to allow multiple I/O statements to execute during one motion statement.

D*n*	Used to specify %travel before the next parallel statement is executed. *n* is a percentage between 0 and 100 which represents the position within the motion where the parallel processing statements should begin. Statements which follow the Dn parameter will begin execution after n% of the motion travel has been completed. When used with the Jump, Jump3, and Jump3CP commands, %travel does not include the depart and approach motion. To execute statements after the depart motion has completed, include D0 (zero) at the beginning of the statement. Dn may appear a maximum of 16 times in a parallel processing statement.
On / Off *n*	Turn Output bit number *n* on or off.
MemOn / MemOff *n*	Turns memory I/O bit number *n* on or off.
Out *p,d*	Outputs data *d* to output port *p*.
MemOut *p, d*	Outputs data *d* to memory I/O port *p*
Signal *s*	Generates synchronizing signal.
Wait *t*	Delays for *t* seconds prior to execution of the next parallel processing statement.
WaitSig *s*	Waits for signal *s* before processing next statement.
Wait Sw(*n*) = *j*	Delays execution of next parallel processing statement until the input bit n is equal to the condition defined by j. (On or Off)

Wait MemSw(*n*) = *j*	Delays execution of the next parallel processing statement until the memory I/O bit n is equal to the condition defined by j. (On or Off)
Print / Input	Prints data to and inputs data from the controlling device.
Print # / Input #	Prints data to and inputs data from the specified communications port.
EnetIO_AnaOut EnetIO_Off EnetIO_On EnetIO_Out	Ethernet I/O outputs.
FbusIO_Off FbusIO_On FbusIO_Out FbusIO_OutW	Fieldbus I/O outputs.

Notes

When Motion is Completed before All I/O Commands are Complete

If, after completing the motion for a specific motion command, all parallel processing statement execution has not been completed, subsequent program execution is delayed until all parallel processing statements execution has been completed. This situation is most likely to occur with short moves with many I/O commands to execute in parallel.

What happens to Parallel I/O Execution when the Till statement is used to stop the arm prior to completing the intended motion

If Till is used to stop the arm at an intermediate travel position, the next statement after the motion statement's execution is delayed until all the execution of all parallel processing statements has been completed.

Specifying n near 100% can cause path motion to decelerate

If a large value of n is used during path motion, the robot may decelerate to finish the current motion. This is because the position specified would normally be during deceleration if CP was not being used. To avoid deceleration, consider placing the processing statement after the motion command. For example, in the example below, the On 1 statement is moved from parallel processing during the jump to P1 to after the jump.

```
CP On
Jump P1 !D96; On 1!
Go P2

CP On
Jump P1
On 1
Go P2
```

The Jump statement and Parallel Processing

It should be noted that execution of parallel processing statements which are used with the Jump statement begins after the rising motion has completed and ends at the start of falling motion.

See Also

Arc, Arc3, Go, Jump, Jump3, Jump3CP, Move, Pulse

!...! Parallel Processing Example

The following examples show various ways to use the parallel processing feature with Motion Commands:.

Parallel processing with the Jump command causes output bit 1 to turn on at the end of the Z joint rising travel and when the 1st, 2nd, and 4th axes begin to move. Then output bit 1 is turned off again after 50% of the Jump motion travel has completed.

```
Function test
   Jump P1 !D0; On 1; D50; Off 1!
Fend
```

Parallel processing with the Move command causes output bit 5 to turn on when the joints have completed 10% of their move to the point P1. Then 0.5 seconds later turn output bit 5 off.

```
Function test2
   Move P1 !D10; On 5; Wait 0.5; Off 5!
Fend
```

#define

S

Defines identifier to be replaced by specified replacement string.

Syntax

#define *identifier* [(*parameter*, [*parameter*])] *string*

Parameters

identifier Keyword defined by user which is an abbreviation for the *string* parameter. Rules for identifiers are as follows:

- The first character must be alphabetic while the characters which follow may be alphanumeric or an underscore (_).
- Spaces or tab characters are not allowed as part of the *identifier* .

parameter Normally used to specify a variable (or multiple variables) which may be used by the replacement string. This provides for a dynamic define mechanism which can be used like a macro. A maximum of up to 8 parameters may be used with the #define command. However, each parameter must be separated by a comma and the parameter list must be enclosed within parenthesis.

string This is the replacement string which replaces the identifier when the program is compiled. Rules regarding replacement strings are as follows:

- Spaces or tabs are allowed in replacement strings.
- Identifiers used with other #define statements cannot be used as replacement strings.
- If the comment symbol (') is included, the characters following the comment symbol will be treated as a comment and will not be included in the replacement string.
- The replacement string may be omitted. In this case the specified identifier is replaced by "nothing" or the null string. This actually deletes the identifier from the program

Description

The #define instruction causes a replacement to occur within a program for the specified identifier. Each time the specified identifier is found the identifier is replaced with the replacement string prior to compilation. However, the source code will remain with the identifier rather than the replacement string. This allows code to become easier to read in many cases by using meaningful identifier names rather than long difficult to read strings of code.

The defined identifier can be used for conditional compiling by combining with the #ifdef or #ifndef commands.

If a parameter is specified, the new identifier can be used like a macro.

Notes

Using #define for variable declaration or label substitutions will cause an error:

It should be noted that usage of the #define instruction for variable declaration will cause an error.

See Also

#ifdef
#ifndef

#define Example

```
' Uncomment next line for Debug mode.
' #define DEBUG

Input #1, A$
#ifdef DEBUG
    Print "A$ = ", A$
#endif
Print "The End"

#define SHOWVAL(x) Print "var = ", x

Integer a

a = 25

SHOWVAL(a)
```

#ifdef...#else...#endif

S

Provides conditional compiling capabilities.

Syntax

#ifdef *identifier*
..put selected source code for conditional compile here.
[**#else**
...put selected source code for false condition here.]
#endif

Parameters

identifier Keyword defined by the user which when defined allows the source code defined between #ifdef and #else or #endif to be compiled. Thus the identifier acts as the condition for the conditional compile.

Description

#ifdef...#else...#endif allows for the conditional compiling of selected source code. The condition as to whether or not the compile will occur is determined based on the *identifier*. #ifdef first checks if the specified identifier is currently defined by #define. The #else statement is optional.

If defined, and the #else statement is not used, the statements between #ifdef and #endif are compiled. Otherwise, if #else is used, then the statements between #ifdef and #else are compiled.

If not defined, and the #else statement is not used, the statements between #ifdef and #endif are skipped without being compiled. Otherwise, if #else is used, then the statements between #else and #endif are compiled.

See Also

#define, #ifndef

#ifdef Example

A section of code from a sample program using #ifdef is shown below. In the example below, the printing of the value of the variable A$ will be executed depending on the presence or absence of the definition of the #define DEBUG pseudo instruction. If the #define DEBUG pseudo instruction was used earlier in this source, the Print A$ line will be compiled and later executed when the program is run. However, the printing of the string "The End" will occur regardless of the #define DEBUG pseudo instruction.

```
' Uncomment next line for Debug mode.
' #define DEBUG

Input #1, A$
#ifdef DEBUG
    Print "A$ = ", A$
#endif
Print "The End"
```

#ifndef...#endif

S

Provides conditional compiling capabilities.

Syntax

#ifndef *identifier*
..Put selected source code for conditional compile here.
[**#else**
...put selected source code for true condition here.]
#endif

Parameters

identifier Keyword defined by the user which when **Not** defined allows the source code defined between #ifndef and #else or #endif to be compiled. Thus the identifier acts as the condition for the conditional compile.

Description

This instruction is called the "if not defined" instruction. #ifndef...#else...#endif allow for the conditional compiling of selected source code. The #else statement is optional.

If defined, and the #else statement is not used, the statements between #ifndef and #endif are not compiled. Otherwise, if #else is used, then the statements between #else and #endif are compiled.

If not defined, and the #else statement is not used, the statements between #ifndef and #endif are compiled. Otherwise, if #else is used, then the statements between #else and #endif are not compiled.

Notes

Difference between #ifdef and #ifndef

The fundamental difference between #ifdef and #ifndef is that the #ifdef instruction compiles the specified source code if the identifier is defined. The #ifndef instruction compiles the specified source code if the identifier is not defined.

See Also

#define, #ifdef

#ifndef Example

A section of code from a sample program using #ifndef is shown below. In the example below, the printing of the value of the variable A$ will be executed depending on the presence or absence of the definition of the #define NODELAY pseudo instruction. If the #define NODELAY pseudo instruction was used earlier in this source, the Wait 1 line **will Not be compiled** along with the rest of the source for this program when it is compiled. (i.e. submitted for running.) If the #define NODELAY pseudo instruction was not used (i.e. NODELAY is not defined) earlier in this source, the Wait 1 line **will be compiled** and later executed when the program is run. The printing of the string "The End" will occur regardless of the #define NODELAY pseudo instruction.

```
' Comment out next line to force delays.
#define NODELAY 1

Input #1, A$
#ifndef NODELAY
    Wait 1
#endif
Print "The End"
```

#include

S

Includes the specified file into the file where the #include statement is used.

Syntax

#include "*fileName*.**INC"**

Parameters

fileName fileName must be the name of an include file in the current project. All include files have the **INC** extension. The filename specifies the file which will be included in the current file.

Description

#include inserts the contents of the specified include file with the current file where the #include statement is used.

Include files are used to contain #define statements.

The #include statement must be used outside of any function definitions.

If the pathname is omitted, #include searches for the include file from the current directory.

An include file may contain a secondary include file. For example, FILE2 may be included within FILE1, and FILE3 may be included within FILE2. This is called nesting.

Line numbers are not allowed in include files.

See Also

#define, #ifdef, #ifndef

#include Example

Include File (Defs.inc)

```
#define DEBUG 1
#define MAX_PART_COUNT 20
```

Program File (main.prg)

```
#include "defs.inc"

Function main
    Integer i

    Integer Parts(MAX_PART_COUNT)

Fend
```

Abs Function

F

Returns the absolute value of a number.

Syntax

Abs(*number*)

Parameters

number Any valid numeric expression.

Return Values

The absolute value of a number.

Description

The absolute value of a number is its unsigned magnitude. For example, **Abs**(-1) and **Abs**(1) both return 1.

See Also

Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Abs Function Example

The following examples are done from the Monitor window using the Print instruction.

```
> print abs(1)
1
> print abs(-1)
1
> print abs(-3.54)
3.54
>
```

Accel Statement

> S

Sets (or displays) the acceleration and deceleration rates for the point to point motion instructions Go, Jump and Pulse.

Syntax

(1) Accel *accel*, *decel* [, *departAccel, departDecel, approAccel, approDecel*]
(2) Accel

Parameters

accel	Integer expression between 1-100 representing a percentage of maximum acceleration rate.
decel	Integer expression between 1-100 representing a percentage of the maximum deceleration rate.
departAccel	Optional. Depart acceleration for Jump. Valid Entries are 1-100
departDecel	Optional. Depart deceleration for Jump. Valid Entries are 1-100
approAccel	Optional. Approach acceleration for Jump. Valid Entries are 1-100
approDecel	Optional. Approach deceleration for Jump. Valid Entries are 1-100

Return Values

When parameters are omitted, the current Accel parameters are displayed.

Description

Accel specifies the acceleration and deceleration for all Point to Point type motions. This includes motion caused by the Go, Jump and Pulse robot motion instructions.

Each acceleration and deceleration parameter defined by the **Accel** instruction may be an integer value from 1-100. This number represents a percentage of the maximum acceleration (or deceleration) allowed.

The Accel instruction can be used to set new acceleration and deceleration values or simply to print the current values. When the Accel instruction is used to set new accel and decel values, the first 2 parameters (*accel* and *decel*) in the **Accel** instruction are required.

The optional *departAccel, departDecel, approAccel,* and *approDecel* parameters are effective for the Jump instruction only and specify acceleration and deceleration values for the depart motion at the beginning of Jump and the approach motion at the end of Jump.

The **Accel** value initializes to the default values (low acceleration) when any one of the following conditions occurs:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Abort All button or Ctrl + C Key

Notes

Executing the Accel command in Low Power Mode (Power Low)

If **Accel** is executed when the robot is in low power mode (Power Low), the new values are stored, but the current values are limited to low values.

The current acceleration values are in effect when Power is set to High, and Teach mode is OFF.

Accel vs. AccelS

It is important to note that the **Accel** instruction does not set the acceleration and deceleration rates for straight line and arc motion. The AccelS instruction is used to set the acceleration and deceleration rates for the straight line and arc type moves.

See Also

AccelR, AccelS, Go, Jump, Jump3, Power, Pulse, Speed, TGo

Accel Statement Example

The following example shows a simple motion program where the acceleration (**Accel**) and speed (Speed) is set using predefined variables.

```
Function acctest
   Integer slow, accslow, decslow, fast, accfast, decfast

   slow = 20     'set slow speed variable
   fast = 100    'set high speed variable
   accslow = 20  'set slow acceleration variable
   decslow = 20  'set slow deceleration variable
   accfast = 100 'set fast acceleration variable
   decfast = 100 'set fast deceleration variable

   Accel accslow, decslow
   Speed slow
   Jump pick
   On gripper
   Accel accfast, decfast
   Speed fast
   Jump place
   .
   .
   .
Fend
```

<Example 2>

Assume the robot is currently in Low Power Mode (Power Low) and from the monitor window the user tries to set the Accel value to 100. Because the robot is in Low Power Mode, a maximum acceleration value of 10 will be set automatically. (The system will not allow an acceleration larger than 10 when in Low Power Mode.)

```
>Accel 100,100
>
>Accel
Low Power Mode
    100     100
    100     100
    100     100
```

<Example 3>

Set the Z joint downward deceleration to be slow to allow a gentle placement of the part when using the Jump instruction. This means we must set the *Zdnd* parameter low when setting the **Accel** values.

```
>Accel 100,100,100,100,100,35

>Accel
    100     100
    100     100
    100     35
>
```

Accel Function

> F

Returns specified acceleration value.

Syntax

Accel(*paramNumber*)

Parameters

paramNumber Integer expression which can have the following values:

1: acceleration specification value

2: deceleration specification value

3: depart acceleration specification value for Jump

4: depart deceleration specification value for Jump

5: approach acceleration specification value for Jump

6: approach deceleration specification value for Jump

Return Values

Integer from 1-100%

See Also

Accel Statement

Accel Function Example

This example uses the **Accel** function in a program:

```
Integer currAccel, currDecel

' Get current accel and decel
currAccel = Accel(1)
currDecel = Accel(2)
Accel 50, 50
Jump pick
' Restore previous settings
Accel currAccel, currDecel
```

AccelR Statement

> S

Sets or displays the acceleration and deceleration values for tool rotation control of CP motion.

Syntax

(1) AccelR *accel*, [*decel*]
(2) AccelR

Parameters

accel Real expression in degrees / second^2.

decel Real expression in degrees / second^2.

Return Values

When parameters are omitted, the current AccelR settings are displayed.

Description

AccelR is effective when the ROT modifier is used in the Move, Arc, Arc3, BMove, TMove, and Jump3CP motion commands.

The **AccelR** value initializes to the default values (low acceleration) when any one of the following conditions occurs:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Abort All button or Ctrl + C Key

See Also

Arc, Arc3, BMove, Jump3CP, Power, SpeedR, TMove

AccelR Statement Example

```
AccelR 360, 200
```

AccelR Function

> F

Returns specified tool rotation acceleration value.

Syntax

AccelR(*paramNumber*)

Parameters

paramNumber Integer expression which can have the following values:

1: acceleration specification value

2: deceleration specification value

Return Values

Real value in degrees / second^2

See Also

AccelR Statement

AccelR Function Example

```
Real currAccelR, currDecelR

' Get current accel and decel
currAccelR = AccelR(1)
currDecelR = AccelR(2)
```

AccelS Statement

> S

Sets the acceleration and deceleration rates for the Straight Line and Continuous Path robot motion instructions such as Move, Arc, Jump3, etc.

Syntax

(1) AccelS *accel*, [*decel*], [*departAccel*], [*departDecel*], [*approAccel*], [*approDecel*]
(2) AccelS

Parameters

accel	Real expression (6-axis robots: 1-25000, other robots: 1-5000) represented in mm/sec^2 units to define acceleration and deceleration values for straight line and continuous path motion. If *decel* is omitted, then *accel* is used to specify both the acceleration and deceleration rates.
decel	Optional. Real expression (6-axis robots: 1-25000, other robots: 1-5000) represented in mm/sec^2 units to define the deceleration value.
departAccel	Optional. Real expression for depart acceleration value for Jump3, Jump3CP.
departDecel	Optional. Real expression for depart deceleration value for Jump3, Jump3CP.
approAccel	Optional. Real expression for approach acceleration value for Jump3, Jump3CP.
approDecel	Optional. Real expression for approach deceleration value for Jump3, Jump3CP.

Return Values

Displays Accel and Decel values when used without parameters

Description

AccelS specifies the acceleration and deceleration for all interpolated type motions including linear and curved interpolations. This includes motion caused by the Move and Arc motion instructions.

The **AccelS** value initializes to the default values (low acceleration) when any one of the following conditions occurs:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Abort All button or Ctrl + C Key

Notes

Executing the AccelS command in Low Power Mode (Power Low):

If **AccelS** is executed when the robot is in low power mode (Power Low), the new values are stored, but the current values are limited to low values.

The current acceleration values are in effect when Power is set to High, and Teach mode is OFF.

Accel vs. AccelS:

It is important to note that the **AccelS** instruction does not set the acceleration and deceleration rates for point to point type motion. (i.e. motions initiated by the Go, Jump, and Pulse instructions.) The Accel instruction is used to set the acceleration and deceleration rates for Point to Point type motion.

See Also

Accel, Arc, Arc3, Jump3, Jump3CP, Power, Move, TMove, SpeedS

AccelS Example

The following example shows a simple motion program where the straight line/continuous path acceleration (AccelS) and straight line/continuous path speed (SpeedS) are set using predefined variables.

```
Function acctest
   Integer slow, accslow, fast, accfast

   slow = 20         'set slow speed variable
   fast = 100        'set high speed variable
   accslow = 200     'set slow acceleration variable
   accfast = 5000    'set fast acceleration variable
   AccelS accslow
   SpeedS slow
   Move P1
   On 1
   AccelS accfast
   SpeedS fast
   Jump P2
   .
   .
   .
Fend
```

<Example 2>

Assume the robot is currently in Low Power Mode (Power Low) and from the monitor window the user tries to set the AccelS value to 1000. Because the robot is in Low Power Mode, a maximum acceleration value of 200 will be set automatically. (The system will not allow an acceleration larger than 200 when in Low Power Mode.)

```
>AccelS 1000

>AccelS
Low Power Mode
     1000.000    1000.000
>
```

AccelS Function

> F

Returns acceleration or deceleration for CP motion commands.

Syntax

AccelS(*paramNumber*)

Parameters

paramNumber Integer expression which can have the following values:

1: acceleration value

2: deceleration value

3: depart acceleration value for Jump3, Jump3CP

4: depart deceleration value for Jump3, Jump3CP

5: approach acceleration value for Jump3, Jump3CP

6: approach deceleration value for Jump3, Jump3CP

Return Values

Real value from 0 - 5000 mm/sec/sec

See Also

AccelS Statement, Arc3, SpeedS, Jump3, Jump3CP

AccelS Function Example

```
Real savAccelS

savAccelS = AccelS(1)
```

Acos Function

F

Returns the arccosine of a numeric expression.

Syntax

Acos(*number***)**

Parameters

number Numeric expression representing the cosine of an angle.

Return Values

Real value, in radians, representing the arccosine of the parameter *number*.

Description

Acos returns the arccosine of the numeric expression. *number* may be any numeric value. The value returned by **Acos** will range from 0 to PI radians. If *number* is < -1 or > 1, the value returned is undefined.

To convert from radians to degrees, use the RadToDeg function.

See Also

Abs, Asin, Atan, Atan2, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Acos Function Example

```
Function acostest
   Double x

   x = Cos(DegToRad(30))
   Print "Acos of ", x, " is ", Acos(x)
Fend
```

Agl Function

F

Returns the joint angle for the selected rotational joint, or position for the selected linear joint.

Syntax

Agl(*jointNumber*)

Parameters

jointNumber	Integer expression representing the joint number. Values are from 1 to the number of joints on the robot.

Return Values

The joint angle for selected rotational joint or position for selected linear joints.

Description

The **Agl** function is used to get the joint angle for the selected rotational joint or position for the selected linear joint.

If the selected joint is rotational, **Agl** returns the current angle, as measured from the selected joint's 0 position, in degrees. The returned value is a real number.

If the selected joint is a linear joint, **Agl** returns the current position, as measured from the selected joint's 0 position, in mm. The returned value is a real number.

If an auxiliary arm is selected with the Arm statement, **Agl** returns the angle (or position) from the standard arm's 0 pulse position to the selected arm.

See Also

PAgl, Pls, PPls

Agl Function Example

The following examples are done from the monitor window using the Print instruction.

```
> print agl(1), agl(2)
 17.234   85.355
```

AglToPls Function

F

Converts robot angles to pulses.

Syntax

AglToPls(*j1*, *j2*, *j3*, *j4*, [*j5*], [*j6*]**)**

Parameters

j1 - j6 Real expressions representing joint angles.

Return Values

A robot point whose location is determined by joint angles converted to pulses.

Description

Use AglToPls to create a point from joint angles.

Note

Assignment to point can cause part of the joint position to be lost.

In certain cases, when the result of **AglToPls** is assigned to a point data variable, the arm moves to a joint position that is different from the joint position specified by **AglToPls**.

For example:

```
P1 = AglToPls(0, 0, 0, 90, 0, 0)
Go P1 ' moves to AglToPls(0, 0, 0, 0, 0, 90) joint position
```

Similarly, when the AglToPls function is used as a parameter in a CP motion command, the arm may move to a different joint position from the joint position specified by **AglToPls**.

```
Move AglToPls(0, 0, 0, 90, 0, 0) ' moves to AglToPls(0, 0, 0, 0, 0, 90)
joint position
```

When using the **AglToPls** function as a parameter in a PTP motion command, this problem does not occur.

See Also

Agl, JA, Pls

AglToPls Function Example

```
Go AglToPls(0, 0, 0, 90, 0, 0)
```

And Operator

Operator used to perform a logical or bitwise And of 2 expressions.

Syntax

result = *expr1* **And** *expr2*

Parameters

expr1, expr2	For logical And, any valid expression which returns a Boolean result. For bitwise And, an integer expression.
result	For logical And, result is a Boolean value. For bitwise And, result is an integer.

Description

A logical **And** is used to combine the results of 2 or more expressions into 1 single Boolean result. The following table indicates the possible combinations.

expr1	*expr2*	*result*
True	True	True
True	False	False
False	True	False
False	False	False

A bitwise **And** performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in expr1 is	And bit in expr2 is	The result is
0	0	0
0	1	0
1	0	0
1	1	1

See Also

LShift, Mask, Not, Or, RShift, Xor

And Operator Example

```
Function LogicalAnd(x As Integer, y As Integer)

  If x = 1 And y = 2 Then
    Print "The values are correct"
  EndIf
Fend

Function BitWiseAnd()

  If (Stat(0) And &H800000) = &H800000 Then
    Print "The enable switch is open"
  EndIf
Fend

>print 15 and 7
7
>
```

AOpen Statement

S

Opens file for appending data.

Syntax

AOpen *fileName* **As** #*fileNumber*
.
.
Close #*fileNumber*

Parameters

fileName	String expression that specifies valid path and file name.
fileNumber	Integer expression representing values from 30 - 63.

Description

Opens the specified file and identifies it by the specified file number. This statement is used for appending data to the specified file. **Close** closes the file and releases the file number.

The specified file must exist on disk or an error will occur. *fileNumber* identifies the file while it is open and cannot be used to refer to a different file until the current file is closed. *fileNumber* is used by other file operations such as Print#, Read, Write, Seek, and Close.

It is recommended that you use the FreeFile function to obtain the file number so that more than one task are not using the same number.

See Also

Close, Print #, BOpen, ROpen, WOpen

AOpen Statement Example

```
Real data(200)
Integer fileNum

fileNum = FreeFile
WOpen "TEST.TXT" As #fileNum
For i = 0 To 100
   Print #fileNum, data(i)
Next I
Close #fileNum
....
....
....
fileNum = FreeFile
AOpen "TEST.TXT" As #fileNum
For i = 101 to 200
   Print #fileNum, data(i)
Next i
Close #fileNum
```

Arc, Arc3 Statements

> S

Arc moves the arm to the specified point using circular interpolation in the XY plane.
Arc3 moves the arm to the specified point using circular interpolation in 3 dimensions.

Syntax

(1) Arc *midPoint, endPoint* [**ROT**] [**CP**] [*searchExpr*] [*!...!*]
(2) Arc3 *midPoint, endPoint* [**ROT**] [**ECP**] [**CP**] [*searchExpr*] [*!...!*]

Parameters

midPoint	Point expression. The middle point (taught previously by the user) which the arm travels through on its way from the current point to *endPoint*.
endPoint	Point expression. The end point (taught previously by the user) which the arm travels to during the arc type motion. This is the final position at the end of the circular move.
ROT	Optional. :Decides the speed/acceleration/deceleration in favor of tool rotation.
ECP	Optional. External control point motion. This parameter is valid when the ECP option is enabled.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Parallel processing statements may be used with the Arc statement. These are optional. (Please see the Parallel Processing description for more information.)

Description

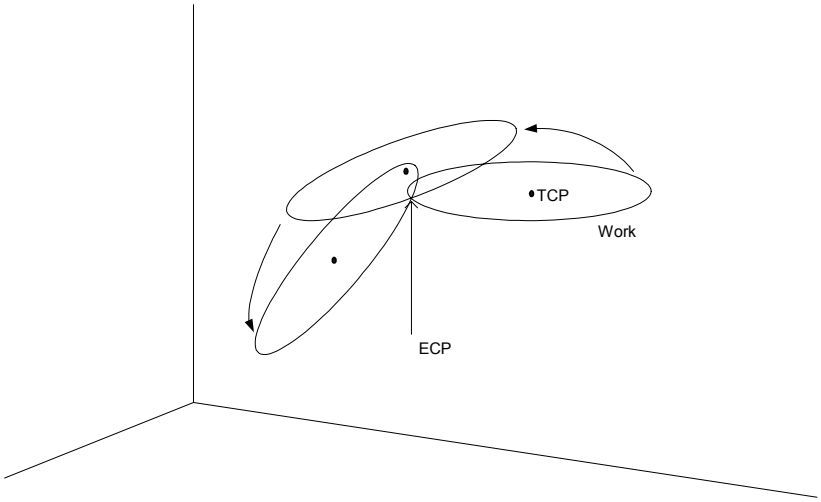
Arc and **Arc3** are used to move the arm in a circular type motion from the current position to *endPoint* by way of *midPoint*. The system automatically calculates a curve based on the 3 points (current position, *endPoint*, and *midPoint*) and then moves along that curve until the point defined by *endPoint* is reached. The coordinates of *midPoint* and *endPoint* must be taught previously before executing the instruction. The coordinates cannot be specified in the statement itself.

Arc and **Arc3** use the SpeedS speed value and AccelS acceleration and deceleration values. Refer to *Using Arc3 with CP* below on the relation between the speed/acceleration and the acceleration/deceleration. If, however, the ROT modifier parameter is used, **Arc** and **Arc3** use the SpeedR speed value and AccelR acceleration and deceleration values. In this case SpeedS speed value and AccelS acceleration and deceleration value have no effect.

Usually, when the move distance is 0 and only the tool orientation is changed, an error will occur. However, by using the ROT parameter and giving priority to the acceleration and the deceleration of the tool rotation, it is possible to move without an error. When there is not an orientational change with the ROT modifier parameter and movement distance is not 0, an error will occur.

Also, when the tool rotation is large as compared to move distance, and when the rotation speed exceeds the specified speed of the manipulator, an error will occur. In this case, please reduce the speed or append the ROT modifier parameter to give priority to the rotational speed/acceleration/deceleration.

When ECP is used (Arc3 only), the trajectory of the external control point coresponding to the ECP number specified by ECP instruction moves circular with respect to the tool coordinate system. In this case, the trajectory of tool center point does not follow a circular line.
.



Setting Speed and Acceleration for Arc Motion

SpeedS and AccelS are used to set speed and acceleration for the **Arc** and **Arc3** instructions. SpeedS and AccelS allow the user to specify a velocity in mm/sec and acceleration in mm/sec^2.

Notes

Arc3 not supported for PG robots

Arc3 cannot be used for robots that use the PG Motion System.

Arc Instruction works in Horizontal Plane Only

The **Arc** path is a true arc in the Horizontal plane. The path is interpolated using the values for *endPoint* as its basis for Z and U. Use **Arc3** for 3 dimensional arcs.

Range Verification for Arc Instruction

The **Arc** and **Arc3** statements cannot compute a range verification of the trajectory prior to the arc motion. Therefore, even for target positions that are within an allowable range, en route the robot may attempt to traverse a path which has an invalid range, stopping with a severe shock which may damage the arm. To prevent this from occurring, be sure to perform range verifications by running the program at low speeds prior to running at faster speeds.

Suggested Motion to Setup for the Arc Move

Because the arc motion begins from the current position, it may be necessary to use the Go, Jump or other related motion command to bring the robot to the desired position prior to executing **Arc** or **Arc3**.

Using Arc, Arc3 with CP

The CP parameter causes the arm to move to the end point without decelerating or stopping at the point defined by *endPoint*. This is done to allow the user to string a series of motion instructions together to cause the arm to move along a continuous path while maintaining a specified speed throughout all the motion. The **Arc** and **Arc3** instructions without CP always cause the arm to decelerate to a stop prior to reaching the end point.

Potential Errors

Changing Hand Attributes

Pay close attention to the HAND attributes of the points used with the **Arc** instruction. If the hand orientation changes (from Right Handed to Left Handed or vice-versa) during the circular interpolation move, an error will occur. This means the arm attribute (/L Lefty, or /R Righty) values must be the same for the current position, *midPoint* and *endPoint* points.

Attempt to Move Arm Outside Work Envelope

If the specified circular motion attempts to move the arm outside the work envelope of the arm, an error will occur.

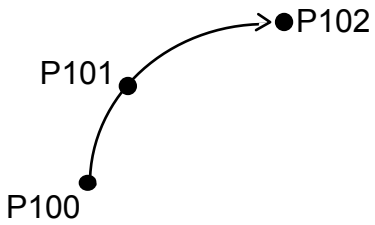
See Also

!Parallel Processing!, AccelS, Move, SpeedS

Arc Example

The diagram below shows arc motion which originated at the point P100 and then moves through P101 and ends up at P102. The following function would generate such an arc:

```
Function ArcTest
   Go P100
   Arc P101, P102
Fend
```



Tip

When first trying to use the **Arc** instruction, it is suggested to try a simple arc with points directly in front of the robot in about the middle of the work envelope. Try to visualize the arc that would be generated and make sure that you are not teaching points in such a way that the robot arm would try to move outside the normal work envelope.

Arch Statement

> S

Defines or displays the **Arch** parameters for use with the Jump, Jump3, Jump3CP instructions.

Syntax

(1) Arch *archNumber, departDist, approDist*
(2) Arch

Parameters

archNumber Integer expression representing the **Arch** number to define. Valid **Arch** numbers are (0-6) making a total of 7 entries into the **Arch** table. (see default **Arch** Table below)

departDist The vertical distance moved (Z) at the beginning of the Jump move before beginning horizontal motion. (specified in millimeters)

approDist The vertical distance required (as measured from the Z position of the point the arm is moving to) to move in a completely vertical fashion with all horizontal movement complete. (specified in millimeters)

Return Values

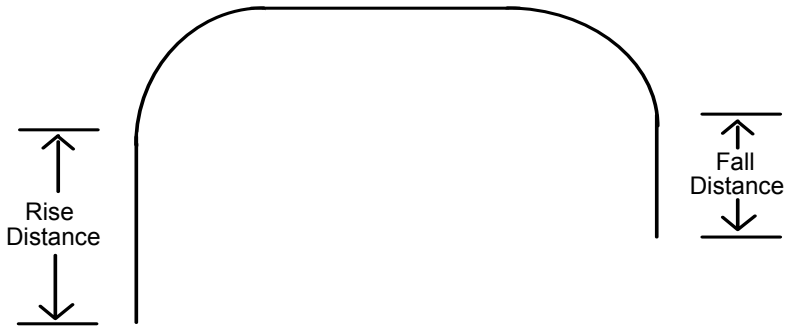
Displays Arch Table when used without parameters.

The entire contents of the Arch table will be displayed when the Arch instruction is issued from the monitor window without any parameters.

Description

The primary purpose of the **Arch** instruction is to define values in the **Arch** Table which is required for use with the Jump motion instruction. The **Arch** motion is carried out per the parameters corresponding to the arch number selected in the Jump C modifier. (To completely understand the **Arch** instruction, the user must first understand the Jump instruction.)

The **Arch** definitions allow the user to "round corners" in the Z direction when using the Jump C instruction. While the Jump instruction specifies the point to move to (including the final Z joint position), the **Arch** table entries specify how much distance to move up before beginning horizontal motion (*riseDist*) and how much distance up from the final Z joint position to complete all horizontal motion (*fallDist*). (See diagram below)



There are a total of 8 entries in the **Arch** Definition Table with 7 of them (0-6) being user definable. The 8th entry (**Arch** 7)is the default Arch which actually specifies no arch at all which is referred to as Gate Motion. (See Gate Motion diagram below) The Jump instruction used with the default **Arch** entry (Entry 8) causes the arm to do the following:

1) Begin the move with only Z-joint motion until it reaches the Z-Coordinate value specified by the LimZ command. (The upper Z value)
2) Next move horizontally to the target point position until the final X, Y and U positions are reached.

3) The Jump instruction is then completed by moving the arm down with only Z-joint motion until the target Z-joint position is reached.

Gate Motion
(JUMP with ARCH 7)

Arch Table Default Values:

Arch Number	Depart Distance	Approach Distance
0	30	30
1	40	40
2	50	50
3	60	60
4	70	70
5	80	80
6	90	90

Notes

Jump Motion trajectory changes depending on motion and speed

Jump motion trajectory is comprised of vertical motion and horizontal motion. It is not a continuous path trajectory. The actual Jump trajectory of arch motion is not determined by **Arch** parameters alone. It also depends on motion and speed.

Always use care when optimizing Jump trajectory in your applications. Execute Jump with the desired motion and speed to verify the actual trajectory.

When speed is lower, the trajectory will be lower. If Jump is executed with high speed to verify an arch motion trajectory, the end effector may crash into an obstacle with lower speed.

In a Jump trajectory, the depart distance increases and the approach distance decreases when the motion speed is set high. When the fall distance of the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the fall distance to be larger.

Even if Jump commands with the same distance and speed are executed, the trajectory is affected by motion of the robot arms. As a general example, for a SCARA robot the vertical upward distance increases and the vertical downward distance decreases when the movement of the first arm is large. When the vertical fall distance decreases and the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the fall distance to be larger.

Another Cause of Gate Motion

When the specified value of the Rising Distance or Falling Distance is larger than the actual Z-joint distance which the robot must move to reach the target position, Gate Motion will occur. (i.e. no type **Arch** motion will occur.)

Setting the Arch Table back to Default Value (Using the Verinit instruction)

Any time the Verinit instruction is issued the **Arch** Table values are set back to their defaults.

Arch values are Maintained

The **Arch** Table values are permanently saved and are not changed until either the user changes them or a Verinit instruction is issued.

See Also

Jump, Verinit

Arch Example

The following are examples of Arch settings done from the monitor window.

```
> arch 0, 15, 15
> arch 1, 25, 50
> jump p1 c1
> arch
  arch0 =    15.000        15.000
  arch1 =    25.000        50.000
  arch2 =    50.000        50.000
  arch3 =    60.000        60.000
  arch4 =    70.000        70.000
  arch5 =    80.000        80.000
  arch6 =    90.000        90.000
>
```

Arch Function

> F

Returns arch settings.

Syntax

Arch(*archNumber, paramNumber*)

Parameters

archNumber	Integer expression representing arch setting to retrieve parameter from (0 to 6).
paramNumber	1: depart distance 2: approach distance

Return Value

Real number containing distance.

See Also

Arch statement

Arch Function Example

```
Real archValues(6, 1)
Integer i

' Save current arch values
For i = 0 to 6
    archValues(i, 0) = Arch(i, 1)
    archValues(i, 1) = Arch(i, 2)
Next i
```

Arm Statement

> S

Selects or displays the arm number to use.

Syntax

(1) Arm *armNumber*
(2) Arm

Parameters

armNumber	Optional integer expression. Valid range is from 0 - 3. The user may select up to 4 different arms. Arm 0 is the standard (default) robot arm. Arm 1 - 3 are auxiliary arms defined by using the ArmSet instruction. When omitted, the current arm number is displayed.

Return Values

When the **Arm** instruction is executed without parameters, the system displays the current arm number.

Description

Allows the user to specify which arm to use for robot instructions. **Arm** allows each auxiliary arm to use common position data. If no auxiliary arms are installed, the standard arm (arm number 0) operates. Since at time of delivery the arm number is specified as 0, it is not necessary to use the **Arm** instruction to select an arm. However, if auxiliary arms are used they must first defined with the ArmSet instruction.

The auxiliary arm configuration capability is provided to allow users to configure the proper robot parameters for their robots when the actual robot configuration is a little different than the standard robot. For example, if the user mounted a 2nd orientation joint to the 2nd robot link, the user will probably want to define the proper robot linkages for the new auxiliary arm which is formed. This will allow the auxiliary arm to function properly under the following conditions:

- Specifying that a single data point be moved through by 2 or more arms.
- Using Pallet
- Using Continuous Path motion
- Using relative position specifications
- Using Local coordinates

For SCARA robots with rotating joints or cylindrical coordinate robots used with a Cartesian coordinate system, joint angle calculations are based on the parameters defined by the ArmSet parameters. Therefore, this command is critical if any auxiliary arm or hand definition is required.

Notes

Arm 0

Arm 0 cannot be defined or changed by the user through the ArmSet instruction. It is reserved since it is used to define the standard robot configuration. When the user sets Arm to 0 this means to use the standard robot arm parameters.

Arm Number Not Defined

Selecting auxiliary arm numbers that have not been defined by the ArmSet command will result in an error.

See Also

ArmClr, ArmSet, ECPSet, TLSet

Arm Statement Example

The following examples are potential auxiliary arm definitions using the ArmSet and Arm instructions. ArmSet defines the auxiliary arm and Arm defines which Arm to use as the current arm. (Arm 0 is the default robot arm and cannot be adjusted by the user.)

From the Monitor window:

```
> ArmSet 1, 300, -12, -30, 300, 0
> ArmSet
  arm0 250  0  0  300  0
  arm1 300  -12  -30  300  0

> Arm 0
> Jump P1     'Jump to P1 using the Standard Arm Config
> Arm 1
> Jump P1     'Jump to P1 using auxiliary arm 1
```

Arm Function

> F

Returns the current arm number for the current robot.

Syntax

Arm

Return Values

Integer containing the current arm number.

See Also

Arm Statement

Arm Function Example

```
Print "The current arm number is: ", Arm
```

ArmClr Statement

> S

Clears (undefines) an arm definition.

Syntax

ArmClr *armNumber*

Parameters

armNumber	Integer expression representing which of 3 arms to clear (undefine). (Arm 0 is the default arm and cannot be cleared.)

See Also

Arm, ArmSet, ECPSet, Local, LocalClr, Tool, TLSet, Verinit

ArmClr Example

```
ArmClr 1
```

ArmSet Statement

> S

Specifies and displays auxiliary arms.

Syntax

(1) ArmSet *armNumber* , *link2Dist*, *joint2Offset, zOffset,* [*link1Dist*], [*orientAngOffset*]
(2) ArmSet

Parameters

armNumber	Integer expression: Valid range from 1-3. The user may define up to 3 different auxiliary arms.
link2Dist	**(For SCARA Robots)** The horizontal distance from the center line of the elbow joint to the center line of the new orientation joint. (i.e. the position where the new auxiliary arm's orientation axis center line is located.)
	(For Cartesian Robots) X axis direction position offset from the original X position specified in mm.
joint2Offset	**(For SCARA Robots)** The offset (in degrees) between the line formed between the normal Elbow center line and the normal orientation Axis center line and the line formed between the new auxiliary arm elbow center line and the new orientation axis center line. (These 2 lines should intersect at the elbow center line and the angle formed is the *joint2Offset*.)
	(For Cartesian Robots) Y axis direction position offset from the original Y position specified in mm.
zOffset	**(For SCARA & Cartesian Robots)** The Z height offset difference between the new orientation axis center and the old orientation axis center. (This is a distance.)
link1Dist	**(For SCARA Robots)** The distance from the shoulder center line to the elbow center line of the elbow orientation of the new auxiliary axis.
	(For Cartesian Robots) This is a dummy parameter (Specify 0)
orientAngOffset	**(For SCARA & Cartesian Robots)** The angular offset (in degrees) for the new orientation axis vs. the old orientation axis.

Return Values

When the **ArmSet** instruction is initiated without parameters, the system displays the current auxiliary arm definition parameters. However, **ArmSet** does not return anything.

Description

Allows the user to specify auxiliary arm parameters to be used in addition to the standard arm configuration. This is most useful when an auxiliary arm or hand is installed to the robot. When using an auxiliary arm, the arm is selected by the Arm instruction.

The *link1Dist* and *orientAngOffset* parameters are optional. If they are omitted, the default values are the standard arm values.

The auxiliary arm configuration capability is provided to allow users to configure the proper robot parameters for their robots when the actual robot configuration is a little different than the standard robot. For example, if the user mounted a 2nd orientation joint to the 2nd robot link, the user will probably want to define the proper robot linkages for the new auxiliary arm which is formed. This will allow the auxiliary arm to function properly under the following conditions:

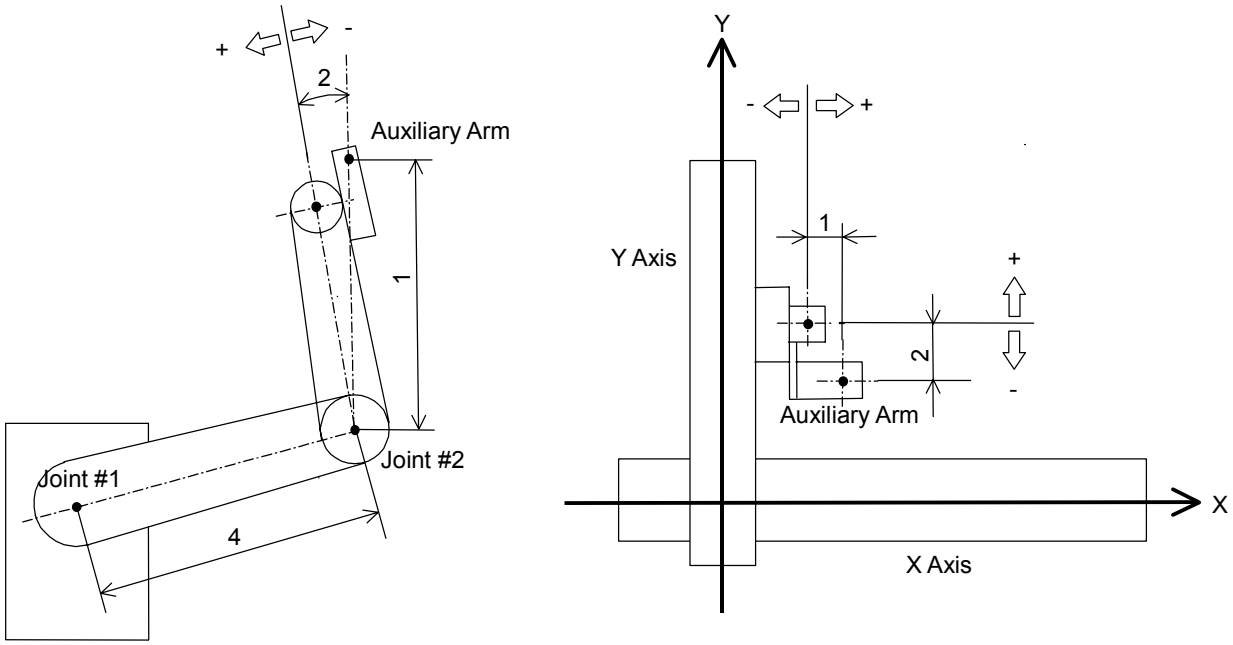
- Specifying that a single data point be moved through by 2 or more arms.
- Using Pallet
- Using Continuous Path motion
- Using relative position specifications
- Using Local coordinates

For SCARA robots with rotating joints or cylindrical coordinate robots used with a Cartesian coordinate system, joint angle calculations are based on the parameters defined by the **ArmSet** parameters. Therefore, this command is critical if any auxiliary arm or hand definition is required.

Notes

Arm 0

Arm 0 cannot be defined or changed by the user. It is reserved since it is used to define the standard robot configuration. When the user sets Arm to 0 this means to use the standard robot arm parameters.



See Also

Arm, ArmClr

ArmSet Statement Example

The following examples are potential auxiliary arm definitions using the **ArmSet** and Arm instructions. **ArmSet** defines the auxiliary arm and Arm defines which Arm to use as the current arm. (Arm 0 is the default robot arm and cannot be adjusted by the user.)

From the Monitor window:

```
> ArmSet 1, 300, -12, -30, 300, 0
> ArmSet
  arm0 250  0  0  300  0
  arm1 300  -12  -30  300  0

> Arm 0
> Jump P1     'Jump to P1 using the Standard Arm Config
> Arm 1
> Jump P1     'Jump to P1 using auxiliary arm 1
```

ArmSet Function

> F

Returns one ArmSet parameter.

Syntax

ArmSet(*armNumber, paramNumber*)

Parameters

armNumber Integer expression representing the arm number to retrieve values for.

paramNumber Integer expression representing the parameter to retrieve (0 to 5), as described below.

SCARA Robots

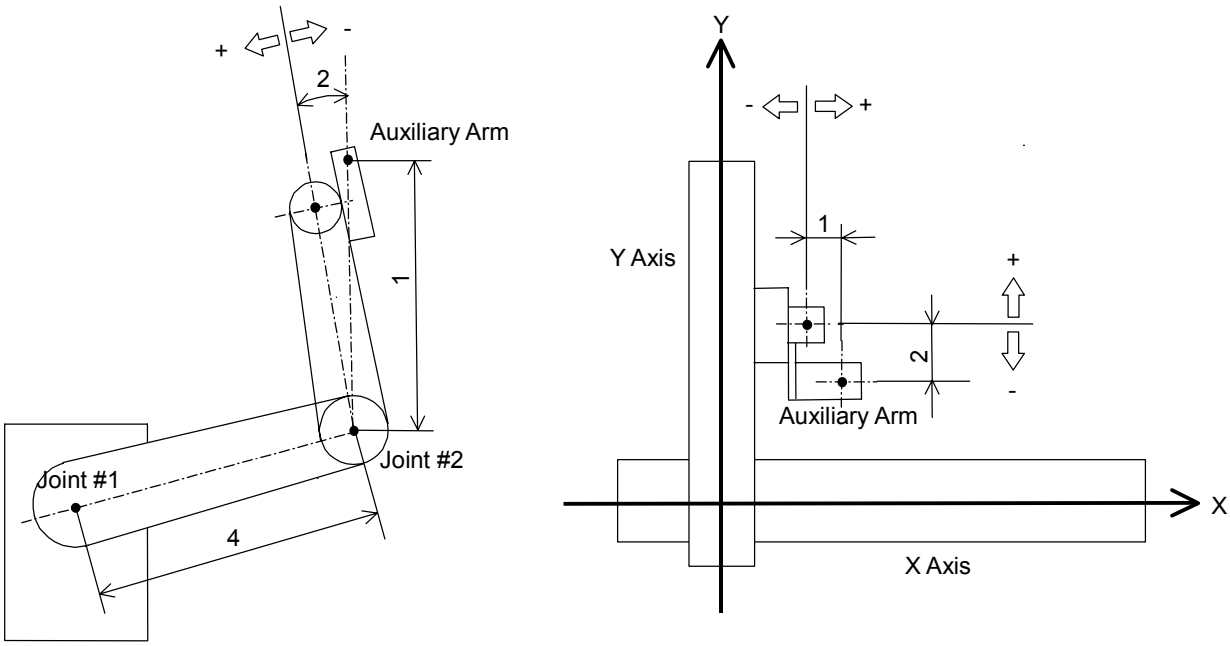
paramNumber	**Value Returned**
1	Horizontal distance from joint #2 to orientation center (mm)
2	Joint #2 angle offset (degree)
3	Height offset (mm)
4	Horizontal distance from joint #1 to joint #2 (mm)
5	Orientation joint angle offset in degrees.

CARTESIAN Robots

paramNumber	**Value Returned**
1	X coordinate axis direction position offset (mm)
2	Y coordinate axis direction position offset (mm)
3	Height offset (mm)
4	Reserved. Must be 0.
5	Orientation joint angle offset in degrees.

Return Values

Real number containing the value of the specified parameter, as described above.



See Also

ArmClr, ArmSet Statement

ArmSet Function Example

```
Real x

x = ArmSet(1, 1)
```

Asc Function

F

Returns the ASCII value of the first character in a character string.

Syntax

Asc(*string*)

Parameters

string Any valid string expression of at least 1 character in length.

Return Values

Returns an integer representing the ASCII value of the 1st character in the string sent to the **ASC** function.

Description

The **Asc** function is used to convert a character to its ASCII numeric representation. The character string send to the **ASC** function may be a constant or a variable.

Notes

Only the First Character ASCII Value is Returned

Although the **Asc** instruction allows character strings larger than 1 character in length, only the 1st character is actually used by the **Asc** instruction. **Asc** returns the ASCII value of the 1st character only.

Potential Errors

Using NULL String:

If the **Asc** function is initiated with an empty or null character string as the parameter value, an error will be issued.

See Also

Chr$, InStr, Left$, Len, Mid$, Right$, Space$, Str$, Val

Asc Function Example

This example uses the **Asc** instruction in a program and from the monitor window as follows:

```
Function asctest
    Integer a, b, c
    a = Asc("a")
    b = Asc("b")
    c = Asc("c")
    Print "The ASCII value of a is ", a
    Print "The ASCII value of b is ", b
    Print "The ASCII value of c is ", c
Fend
```

From the Monitor window:

```
>print asc("a")
97
>print asc("b")
98
>
```

Asin Function

F

Returns the arcsine of a numeric expression.

Syntax

Asin(*number***)**

Parameters

number Numeric expression representing the sine of an angle.

Return Values

Real value, in radians, representing the arc sine of the parameter *number*.

Description

Asin returns the arcsine of the numeric expression. *number* may be any numeric value. The value returned by **Asin** will range from -PI / 2 to PI / 2 radians. If *number* is < -1 or > 1, the value returned is undefined.

To convert from radians to degrees, use the RadToDeg function.

See Also

Abs, Acos, Atan, Atan2, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Asin Function Example

```
Function asintest
   Double x

   x = Sin(DegToRad(45))
   Print "Asin of ", x, " is ", Asin(x)
Fend
```

Atan Function

F

Returns the arctangent of a numeric expression.

Syntax

Atan(*number***)**

Parameters

number Numeric expression representing the tangent of an angular value.

Return Values

Real value, in radians, representing the arctangent of the parameter *number*.

Description

Atan returns the arctangent of the numeric expression. The numeric expression (*number)* may be any numeric value. The value returned by **Atan** will range from -PI to PI radans.

To convert from radians to degrees, use the RadToDeg function.

See Also

Abs, Acos, Asin, Atan2, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Atan Function Example

```
Function atantest
   Real x, y
   x = 0
   y = 1
   Print "Atan of ", x, " is ", Atan(x)
   Print "Atan of ", y, " is ", Atan(y)
Fend
```

Atan2 Function

F

Returns the angle of the imaginary line connecting points (0,0) and (X, Y) in radians.

Syntax

Atan2*(X, Y)*

Parameters

X	Numeric expression representing the X coordinate.
Y	Numeric expression representing the Y coordinate.

Return Values

Numeric value in radians (-PI to +PI).

Description

Atan2(*X*, *Y*) returns the angle of the line which connects points (0, 0) and (*X*, *Y*). This trigonometric function returns an arctangent angle in all four quadrants.

See Also

Abs, Acos, Asin, Atan, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Atan2 Function Example

```
Function at2test
   Real x, y
   Print "Please enter a number for the X Coordinate:"
   Input x
   Print "Please enter a number for the Y Coordinate:"
   Input y
   Print "Atan2 of ", x, ", ", y, " is ", Atan2(x, y)
Fend
```

ATCLR Statement

> S

Clears and intializes the average torque for one or more joints.

Syntax

ATCLR [*j1*], [*j2*], [*j3*], [*j4*], [*j5*], [*j6*]

Parameters

j1 - j6	Optional. Integer expression representing the joint number. If no parameters are supplied, then the average torque values are cleared for all joints.

Description

ATCLR clears the average torque values for the specified joints.

You must execute ATCLR before executing ATRQ.

See Also

ATRQ, PTRQ

ATCLR Statement Example

```
> atclr
> go p1
> atrq 1
      0.028
> atrq
      0.028    0.008
      0.029    0.009
      0.000    0.000
>
```

ATRQ Statement

> S

Displays the average torque for the specified joint.

Syntax

ATRQ [*jointNumber*]

Parameters

jointNumber Optional. Integer expression representing the joint number.

Return Values

Displays current average torque values for all joints.

Description

ATRQ displays the average RMS (root-mean-square) torque of the specified joint. The loading state of the motor can be obtained by this instruction. The result is a real value from 0 to 1 with 1 being maximum average torque.

You must execute ATCLR before this command is executed.

This instruction is time restricted. You must execute ATRQ within 60 seconds after ATCLR is executed. When this time is exceeded, error 4030 "Torque RMS calculation value overflowed." occurs.

See Also

ATCLR, ATRQ Function, PTRQ

ATRQ Statement Example

```
> atclr
> go p1
> atrq 1
      0.028
> atrq
      0.028    0.008
      0.029    0.009
      0.000    0.000
>
```

ATRQ Function

F

Returns the average torque for the specified joint.

Syntax

ATRQ(*jointNumber*)

Parameters

jointNumber Integer expression representing the joint number.

Return Values

Real value from 0 to 1.

Description

The **ATRQ** function returns the average RMS (root-mean-square) torque of the specified joint. The loading state of the motor can be obtained by this instruction. The result is a real value from 0 to 1 with 1 being maximum average torque.

You must execute ATCLR before this function is executed.

This instruction is time restricted. You must execute ATRQ within 60 seconds after ATCLR is executed. When this time is exceeded, error 4030 "Torque RMS calculation value overflowed." occurs.

See Also

ATRQ Statement, PTCLR, PTRQ Statement

ATRQ Function Example

This example uses the **ATRQ** function in a program:

```
Function CheckAvgTorque
  Integer i

  Go P1
  ATCLR
  Go P2
  Print "Average torques:"
  For i = 1 To 4
    Print "Joint ", i, " = ", ATRQ(i)
  Next i
Fend
```

Base Statement

> S

Defines and displays the base coordinate system.

Syntax

(1) Base (*pBase1* **:** *pRobot1* **),** *(pBase2* **:** *pRobot2)* [, { **L** | **R** }]
(2) Base *pOrigin*
(3) Base *pOrigin, pXaxis, pYaxis*, [{ **X** | **Y** }]

Parameters

pBase1, pBase2	Point variables with point data in the base coordinate system.
pRobot1, pRobot2	Point variables with point data in the robot absolute coordinate system.
L | **R**	Optional. Align base origin to left (first) or right (second) robot absolute points.
pOrigin	Origin point of base coordinate system.
pXaxis	A point on the X axis of coordinate system if X alignment is specified.
pYaxis	A point along the Y axis of coordinate system if Y alignment is specified.
X | **Y**	Optional. If X alignment is specified, then *pXaxis* is on the X axis of the new coordinate system and only the Z coordinate of *pYaxis* is used. If Y alignment is specified, then *pYaxis* is on the Y axis of the new coordinate system and only the Z coordinate of *pXaxis* is used. If ommitted, X alignment is assumed.

Description

Defines the robot base coordinate system by specifying base coordinate system origin and rotation angle in relation to the robot absolute coordinate system.

To reset the Base coordinate system to default, execute the following statement. This will make the base coordinate system the same as the robot absolute coordinate system.

```
Base XY(0, 0, 0, 0)
```

Notes

Changing the base coordinate system affects all local definitions

When base coordinates are changed, all local coordinate systems must be re-defined.

See Also

Local

Base Statement Example

Define base coordinate system origin at 100 mm on X axis and 100 mm on Y axis

```
> Base XY(100, 100, 0, 0)
```

BClr Function

F

Clear one bit in a number and return the new value

Syntax

BClr (*number, bitNum*)

Parameters

number Specifies the numeric value to clear the bit by an expression or numeric value.

bitNum Specifies the bit (positive integer) to be cleared by an expression or numeric value.

Return Values

Returns the new value of the specified numeric value (integer).

See Also

BSet, BTst

BClr Example

```
flags = BClr(flags, 1)
```

Beep Statement

S

Sounds a tone through the computer's speaker.

Syntax

Beep

See Also

MsgBox, Print

Beep Statement Example

```
Beep
Print "Initializing robot"
```

BGo Statement

> S

Executes Point to Point relative motion, in the selected local coordinate system.

Syntax

BGo *destination* [**CP**] [*searchExpr*] [*!...!*]

Parameters

destination	The target destination of the motion using a point expression.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to execute I/O and other commands during motion.

Description

Executes point to point relative motion, in the selected local coordinate system that is specified in the *destination* point expression.

If a local coordinate system is not specified, relative motion will occur in local 0 (base coordinate system).

Arm orientation attributes specified in the *destination* point expression are ignored. The manipulator keeps the current arm orientation attributes. However, for a 6-Axis manipulator, the arm orientation attributes are automatically changed in such a way that joint travel distance is as small as possible.

The Till modifier is used to complete BGo by decelerating and stopping the robot at an intermediate travel position if the current Till condition is satisfied.

The Find modifier is used to store a point in FindPos when the Find condition becomes true during motion.

When Till is used and the Till condition is satisfied, the manipulator halts immediately and the motion command is finished. If the Till condition is not satisfied, the manipulator moves to the destination point.

When Find is used and the Find condition is satisfied, the current position is stored. Please refer to Find for details.

When parallel processing is used, other processing can be executed in parallel with the motion command.

CP parameters can start the acceleration of the next motion command when the deceleration starts. In this case the track will not come to the destination coordinate and moves to the next point.

See Also

Accel, BMove, Find, !....! Parallel Processing, Point Assignment, Speed, Till, TGo, TMove, Tool

BGo Example

```
> BGo XY(100, 0, 0, 0) 'Move 100mm in X direction
                       '(in the base coordinate system)

Function BGoTest

  Speed 50
  Accel 50, 50
  Power High

  P1 = XY(300, 300, -20, 0)
  P2 = XY(300, 300, -20, 0) /L
  Local 1, 0, 0, 0, 45

  GoP1
  Print P*
  BGo XY(0, 50, 0, 0)
  Pring P*

  Go P2
  Print P*
  BGo XY(0, 50, 0, 0)
  Print P*

  BGo XY(0, 50, 0, 0) /1
  Print P*

Fend

[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  350.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /L /0
 X:  300.000 Y:  350.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /L /0
 X:  264.645 Y:  385.355 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /L /0
```

BMove Statement

> S

Executes linear interpolation relative motion, in the selected local coordinate system

Syntax

BMove *destination* [**ROT**] [**CP**] [*searchExpr*] [*!...!*]

Parameters

destination	The target destination of the motion using a point expression.
ROT	Optional. :Decides the speed/acceleration/deceleration in favor of tool rotation.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to execute I/O and other commands during motion.

Description

Executes linear interpolated relative motion, in the selected local coordinate system that is specified in the *destination* point expression.

If a local coordinate system is not specified, relative motion will occur in local 0 (base coordinate system).

Arm orientation attributes specified in the *destination* point expression are ignored. The manipulator keeps the current arm orientation attributes. However, for a 6-Axis manipulator, the arm orientation attributes are automatically changed in such a way that joint travel distance is as small as possible.

BMove uses the SpeedS speed value and AccelS acceleration and deceleration values. Refer to *Using BMove with CP* below on the relation between the speed/acceleration and the acceleration/deceleration. If, however, the ROT modifier parameter is used, **BMove** uses the SpeedR speed value and AccelR acceleration and deceleration values. In this case SpeedS speed value and AccelS acceleration and deceleration value have no effect.

Usually, when the move distance is 0 and only the tool orientation is changed, an error will occur. However, by using the ROT parameter and giving priority to the acceleration and the deceleration of the tool rotation, it is possible to move without an error. When there is not an orientational change with the ROT modifier parameter and movement distance is not 0, an error will occur.

Also, when the tool rotation is large as compared to move distance, and when the rotation speed exceeds the specified speed of the manipulator, an error will occur. In this case, please reduce the speed or append the ROT modifier parameter to give priority to the rotational speed/acceleration/deceleration.

The Till modifier is used to complete BMove by decelerating and stopping the robot at an intermediate travel position if the current Till condition is satisfied.

The Find modifier is used to store a point in FindPos when the Find condition becomes true during motion.

When Till is used and the Till condition is satisfied, the manipulator halts immediately and the motion command is finished. If the Till condition is not satisfied, the manipulator moves to the destination point.

When Find is used and the Find condition is satisfied, the current position is stored. Please refer to Find for details.

When parallel processing is used, other processing can be executed in parallel with the motion command.

Notes

Using BMove with CP

The CP parameter causes the arm to move to *destination* without decelerating or stopping at the point defined by *destination*. This is done to allow the user to string a series of motion instructions together to cause the arm to move along a continuous path while maintaining a specified speed throughout all the motion. The **BMove** instruction without CP always causes the arm to decelerate to a stop prior to reaching the point *destination*.

See Also

AccelS, BGo, Find, !....! Parallel Processing, Point Assignment, SpeedS, TGo, Till, TMove, Tool

BMove Example

```
> BMove XY(100, 0, 0, 0)    'Move 100mm in the X
                            'direction (in the local coordinate system)

Function BMoveTest

  Speed 50
  Accel 50, 50
  SpeedS 100
  AccelS 1000, 1000
  Power High

  P1 = XY(300, 300, -20, 0)
  P2 = XY(300, 300, -20, 0) /L
  Local 1, 0, 0, 0, 45

  Go P1
  Print P*
  BMove XY(0, 50, 0, 0)
  Print P*

  Go P2
  Print P*
  BMove XY(0, 50, 0, 0)
  Print P*

  BMove XY(0, 50, 0, 0) /1
  Print P*

Fend
```

```
[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  350.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /L /0
 X:  300.000 Y:  350.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /L /0
 X:  264.645 Y:  385.355 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /L /0
```

Boolean Statement

S

Declares variables of type Boolean. (1 byte whole number).

Syntax

Boolean *varName* [**(***subscripts***)**], [*varName* [**(***subscripts***)**]**...**]

Parameters

varName	Variable name which the user wants to declare as type **Boolean**.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

Boolean is used to declare variables as type **Boolean**. Variables of type **Boolean** can contain one of two values, 0 = False and -1 = True. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

See Also

Byte, Double, Global, Integer, Long, Real, String

Boolean Statement Example

```
Boolean partOK
Boolean A(10)          'Single dimension array of boolean
Boolean B(10, 10)      'Two dimension array of boolean
Boolean C(10, 10, 10)  'Three dimension array of boolean

partOK = CheckPart()
If Not partOK Then
   Print "Part check failed"
EndIf
```

BOpen Statement

S

Opens file for read / write binary access.

Syntax

BOpen *fileName* **As** #*fileNumber*
.
.
Close #*fileNumber*

Parameters

fileName	String expression that specifies valid path and file name.
fileNumber	Integer expression representing values from 30 - 63.

Description

Opens the specified file and identifies it by the specified file number. This statement is used for accessing the specified file in binary mode. **Close** closes the file and releases the file number.

You must use the ReadBin and WriteBin commands to read and write data in binary mode.

fileNumber identifies the file while it is open and cannot be used to refer to a different file until the current file is closed. *fileNumber* is used by other file operations such as Print#, Read, Write, Seek, and Close.

If the specified file does not exist on disk, the file will be created and the data will be written into it. If the specified file already exists on disk, the data will be written and read starting from the beginning of the existing data.

The read/write position (pointer) of the file can be changed using the Seek command. When switching between read and write access, you must use Seek to reposition the file pointer.

It is recommended that you use the FreeFile function to obtain the file number so that more than one task are not using the same number.

See Also

Close, AOpen, FreeFile, ReadBin, ROpen, WOpen, WriteBin

BOpen Statement Example

```
Real data
Integer fileNum, i

fileNum = FreeFile
BOpen "TEST.DAT" As #fileNum
For i = 0 To 100
   WriteBin #fileNum, i
Next i
Close #fileNum

fileNum = FreeFile
BOpen "TEST.DAT" As #fileNum
Seek #fileNum, 10
ReadBin #fileNum, data
Print "data = ", data
Close #fileNum
```

Brake Statement

>

Turns brake on or off for specified joint of the current robot.

Syntax

Brake *status, jointNumber*

Parameters

status The keyword **On** is used to turn the brake on. The keyword **Off** is used to turn the brake off.

jointNumber The joint number from 1 to 6.

Description

The Brake command is used to turn brakes on or off for one joint. It can only be executed as a monitor command. This command is intended for use by maintenance personel only. It can only be used for 6-Axis robots.

⚠ WARNING	Use extreme caution when turning off a brake. Ensure that the joint is properly supported, otherwise the joint can fall and cause damage to the robot and personel.

Before releasing the brake, be ready to use the emergency stop switch so that you can immediately press it. When the controller is in emergency stop status, the motor brakes are locked. Be aware that the robot arm may fall by its own weight when the brake is turned off with Brake command.

See Also

Motor, Power, Reset, Robot, SFree, SLock

Brake Example

```
> brake on, 1
> brake off, 1
```

BSet Function

F

Sets a bit in a number and returns the new value.

Syntax

BSet (*number*, *bitNum*)

Parameters

number	Specifies the value to set the bit with an expression or numeric value.
bitNum	Specifies the bit (positive integer) to be set by an expression or numeric value.

Return Values

Returns the bit set value of the specified numeric value (integer).

See Also

BClr, BTst

BSet Example

```
flags = BSet(flags, 1)
```

BTst Function

F

Returns the status of 1 bit in a number.

Syntax

BTst (*number*, *bitNum*)

Parameters

number Specifies the number for the bit test with an expression or numeric value.

bitNum Specifies the bit (positive integer) to be tested.

Return Values

Returns the bit test results (integer 1 or 0) of the specified numeric value.

See Also

BClr, BSet

BTst Example

```
If BTst(flags, 1) Then
    Print "Bit 1 is set"
End If
```

Byte Statement

S

Declares variables of type Byte. (1 byte whole number).

Syntax

Byte *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameters

varName	Variable name which the user wants to declare as type **Byte**.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

Byte is used to declare variables as type **Byte**. Variables of type **Byte** can contain whole numbers ranging in value from -128 to +127. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

See Also

Boolean, Double, Global, Integer, Long, Real, String

Byte Example

The following example declares a variable of type **Byte** and then assigns a value to it. A bitwise And is then done to see if the high bit of the value in the variable test_ok is On (1) or Off (0). The result is printed to the display screen. (Of course in this example the high bit of the variable test_ok will always be set since we assigned the variable the value of 15.)

```
Function Test
    Byte A(10)          'Single dimension array of byte
    Byte B(10, 10)      'Two dimension array of byte
    Byte C(10, 10, 10)  'Three dimension array of byte
    Byte test_ok
    test_ok = 15
    Print "Initial Value of test_ok = ", test_ok
    test_ok = (test_ok And 8)
    If test_ok <> 8 Then
       Print "test_ok high bit is ON"
    Else
       Print "test_ok high bit is OFF"
    End If
Fend
```

Calib Statement

> S

Replaces the current arm posture pulse values with the current CalPls values.

Syntax

Calib *joint1,* [*joint2*], [*joint3*], [*joint4*] , [*joint5*] , [*joint6*]

Parameters

joint	Integer number from 1-6 which represents the joint number to calibrate. While normally only 1 joint may need calibration at a time, up to all 6 joints may be calibrated with the **Calib** command at the same time.

Description

Automatically calculates and specifies the offset (Hofs) value. This offset is necessary for matching the origin for each robot joint motor to the corresponding robot mechanical origin.

The **Calib** command should be used when the motor pulse value has changed. The most common occurrence for use is after changing a motor. Normally, the calibration position pulse values would match the CalPls pulse values. However, after maintenance operations such as changing the motors, these two sets of values will no longer match, and therefore calibration becomes necessary.

Calibration may be accomplished by moving the arm to a desired calibration position, and then executing the **Calib** command. By executing **Calib**, the calibration position pulse value is changed to the CalPls value, (the correct pulse value for the calibration position)

In order to perform a proper calibration, Hofs values must be determined. To have Hofs values automatically calculated, move the arm to the desired calibration position, and execute **Calib**. The controller automatically calculates Hofs values based on the calibration pulse values and on the CalPls pulse values.

Notes

Use caution when using the Calib command

Calib is intended to be used for maintenance purposes only. Execute **Calib** only when necessary.

Executing **Calib** causes the Hofs value to be replaced. Because unintended Hofs value changes can cause unpredictable robot motion, use caution in executing **Calib** only when necessary.

Potential Errors

No Joint Number Specified Error

If the joint number is not specified with the **Calib** command, an error will occur.

See Also

CalPls, Hofs

Calib Example

The following example is done from Monitor.

```
> CalPls   'Display current CalPls values
   65523 43320
   -1550 21351
> Pulse    'Display current Pulse values
   65526 49358
   1542  21299
> Calib 2  'Execute calibration for joint 2 only
> Pulse    'Display (changed) Pulse values
   65526 43320
   -1542 21299
>
```

Call Statement

S

Calls a user function.

Syntax

Call *funcName* [(*argList*)]

Parameters

funcName	The name of a Function which is being called.
argList	Optional. List of arguments that were specified in the Function declaration.

Description

The **Call** instruction causes the transfer of program control to a function (defined in Function...Fend). This means that the **Call** instruction causes program execution to leave the current function and transfer to the function specified by **Call**. Program execution then continues in that function until an Exit Function or Fend instruction is reached. Control is then passed back to the original calling function at the next statement after the **Call** instruction.

You may omit the Call keyword and argument parentheses. For example, here is a call statement used with or without the Call keyword:

```
Call MyFunc(1, 2)
MyFunc 1, 2
```

To execute a subroutine within a function, use GoSub...Return.

See Also

Function, GoSub

Call Statement Example

<File1: MAIN.PRG>

```
Function main
    Call InitRobot(1)
Fend
```

<File2: INIT.PRG>

```
Function InitRobot(robotNumber As Integer)

    Integer savRobot

    savRobot = Robot
    Robot robotNumber
    Motor On
    If Not MCalComplete Then
        MCal
    End If
    Robot savRobot
Fend
```

CalPls Statement

> S

Specifies and displays the position and orientation pulse values for calibration.

Syntax

(1) CalPls *j1Pulses, j2Pulses, j3Pulses, j4Pulses,* [*j5Pulses*], [*j6Pulses*]
(2) CalPls

Parameters

j1Pulses	First joint pulse value. This is a long integer expression.
j2Pulses	Second joint pulse value. This is a long integer expression.
j3Pulses	Third joint pulse value. This is a long integer expression.
j4Pulses	Fourth joint pulse value. This is a long integer expression.
j5Pulses	Optional. Fifth joint pulse value. This is a long integer expression.
j6Pulses	Optional. Sixth joint pulse value. This is a long integer expression.

Return Values

When parameters are omitted, displays the current **CalPls** values.

Description

Specifies and maintains the correct position pulse value(s) for calibration.

CalPls is intended to be used for maintenance, such as after changing motors or when motor zero position needs to be matched to the corresponding arm mechanical zero position. This matching of motor zero position to corresponding arm mechanical zero position is called calibration.

Normally, the calibration position Pulse values match the **CalPls** pulse values. However, after performing maintenance operations such as changing motors, these two sets of values no longer match, and therefore calibration becomes necessary.

Calibration may be accomplished by moving the arm to a certain calibration position and then executing Calib. By executing Calib, the calibration position pulse value is changed to the **CalPls** value (the correct pulse value for the calibration position.)

Hofs values must be determined to execute calibration. To have Hofs values automatically calculated, move the arm to the desired calibration position, and execute Calib. The controller automatically calculates Hofs values based on calibration position pulse values and on the **CalPls** values

Notes

CalPls Values Cannot be Changed by cycling power

CalPls values are not initialized by turning main power to the controller off and then on again. The only method to modify the **CalPls** values is to execute the Calib command.

See Also

Calib, Hofs

CalPls Statement Example

The following example is executed from the monitor window.

```
> CalPls   'Display current CalPls values
65523, 43320, -1550, 21351
> Pulse
PULSE: 1: 65526 pls 2: 49358 pls 3:  -1542 pls 4: 21299 pls
> Calib 4
> Pulse
PULSE: 1: 65526 pls 2: 49358 pls 3:  -1542 pls 4: 21351 pls
>
```

CalPls Function

F

Returns calibration pulse value specified by the CalPls Statement.

Syntax

CalPls(*joint*)

Parameters

joint Integer expression representing a robot joint number or 0 to return CalPls status.

Return Values

Integer value containing number of calibration pulses. When *joint* is 0, returns 1 or 0 depending on if CalPls has been executed.

See Also

CalPls statement

CalPls Function Example

This example uses the **CalPls** function in a program:

```
Function DisplayCalPlsValues
   Integer i

   Print "CalPls Values:"
   For i = 1 To 4
      Print "Joint ", i, " CalPls = ", CalPls(i)
   Next i
Fend
```

Chain Statement

S

Stops execution of the current program group and begins execution of the specified program group.

Syntax

Chain *groupName*

Parameters

groupName	A string expression representing the name of the program group to execute. The program group must reside in the current project.

Description

Chain causes execution of the current program in the current program group to stop and then begins execution of the 1st program in the program group specified by the **Chain** instruction. (The 1st program in the program group is the program defined as the main program for that program group.)

Keep in mind that the user has the option to specify separate point files for each program group. This means that when the **Chain** instruction causes a new program group to be used a new point file may also be used (depending on whether or not the user specified separate point files for each program group).

Chain is useful for branching execution from one program group to another. For example, the user may have 10 different product types each of which has its own program and possibly also own set of teach points. In this case **Chain** can be used to branch to the proper program based on the current product type.

Chain can also be used to restart the current program group. This is useful when a critical error has occurred and you want to start the program from the beginning. You can execute **Chain** from inside an error handler to accomplish this.

Notes

I/O Conditions are Maintained

Because a software reset does not occur when executing the **Chain** instruction, the I/O output and memory I/O conditions are maintained.

OnErr Exception

If the user executes the **Chain** instruction with a *groupName* that does not exist an error occurs. The OnErr instruction cannot be used in this instance to handle the error.

See Also

Restart, Xqt

Chain Function Example

Use the Chain statement in your programs to start another program group in the project. When the Chain statement executes, it loads the points for the program group being chained to and then starts the main program of the group.

Program Groups must be enabled to use Chain. You can enable Program Groups from the Edit Command in the Project Menu.

Here is a simple example that uses Chain.

Program Group: MAINGRP
Program Name: MAIN.PRG

```
Function Main

mainLoop:
   Print "Main Menu"
   Print "1) Run widget 1"
   Print "2) Run widget 2"
   Print "3) Quit
   Input choice
   Select choice
      Case 1
         Chain "widget1"
      Case 2
         Chain "widget2"
      Case 3
         End
      Default
         GoTo mainLoop
   Send
Fend
```

Program Group: WIDGET1
Program Name: WID1.PRG

```
Function Widget1
   Print "Executing widget 1"
   Chain "MAINGRP"       ' Start main group again
Fend
```

Program Group: WIDGET2
Program Name: WID2.PRG

```
Function Widget2
   Print "Executing widget 2"
   Chain "MAINGRP"      ' Start main group again
Fend
```

ChDir Command

> Monitor Window Abbr. - Cd

Changes and displays the current directory.

Syntax

(1) ChDir *pathName*
(2) ChDir

Parameters

pathName	String expression representing the name of the new default path.

Description

(1) Changes the default directory to the specified directory

(2) Displays the current directory. When the pathname is omitted, the current directory is displayed. This is used to display the current directory when it is not known.

At power on, the root directory becomes the current directory.

See Also

ChDrive, Dir

ChDir Example

The following examples are done from the monitor window.

```
> ChDir \         'Change current directory to the root directory
> ChDir..         'Change current directory to parent dir

> Cd \TEST\H55    'Change current directory to \H55 in \TEST

> Cd              'Display current directory
A:\TEST\H55\
```

ChDrive Statement

S

Change the default disk drive for file operations.

Syntax

ChDrive *drive*

Parameters

drive String expression or literal containing a valid drive letter.

See Also

ChDir, CurDrive$

ChDrive Statement Example

```
ChDrive "d"
```

ChkCom Function

F

Returns number of characters in the reception buffer of a communication port

Syntax

ChkCom (*portNumber)*

Parameters

portNumber Integer expression for port number to check.

Return Values

Number of characters received (integer).

If the port cannot receive characters, the following negative values are returned to report the current port status:

-2 Port is used by another task
-3 Port is not open

See Also

CloseCom, OpenCom, Read, Write

ChkCom Example

```
Integer numChars

numChars = ChkCom(1)
```

ChkNet Function

F

Returns number of characters in the reception buffer of a network port

Syntax

ChkNet (*portNumber)*

Parameters

portNumber Integer expression for port number to check.

Return Values

Number of characters received (integer).

If the port cannot receive characters, the following negative values are returned to report the current port status:

-1 Port is open but communication has not been established
-2 Port is used by another task
-3 Port is not open

See Also

CloseNet, OpenNet, Read, Write

ChkNet Example

```
Integer numChars

numChars = ChkNet(128)
```

Chr$ Function

F

Returns the character specified by a numeric ASCII value.

Syntax

Chr$*(number)*

Parameters

number An integer expression between 1 and 255.

Return Values

Returns a character that corresponds with the specified ASCII code specified by the value of *number*.

Description

Chr$ returns a character string (1 character) having the ASCII value of the parameter *number*. When the *number* specified is outside of the range 1-255 an error will occur.

See Also

Asc, Instr, Left$, Len, Mid$, Right$, Space$, Str$, Val

Chr$ Function Example

The following example declares a variable of type String and then assigns the string "ABC" to it. The Chr$ instruction is used to convert the numeric ASCII values into the characters "A", "B" and "C". The &H means the number following is represented in hexadecimal form. (&H41 means Hex 41)

```
Function Test
    String temp$
    temp$ = Chr$(&H41) + Chr$(&H42) + Chr$(&H43)
    Print "The value of temp = ", temp$
Fend
```

Clear

S

Erases the position data in memory for the current robot.

Syntax

Clear

Description

Clear initializes the position data area for the current robot. Use this instruction to erase point definitions which reside in memory before teaching new points.

See Also

New, Plist

Clear Statement Example

The example below shows simple examples of using the **Clear** command (from the monitor window). Notice that no teach points are shown when initiating the Plist command once the **Clear** command is given.

```
>P1=100,200,-20,0/R
>P2=0,300,0,20/L
>Plist
P1=100,200,-20,0/R
P2=0,300,0,20/L
>clear
>Plist
>
```

Close Statement

S

Closes a file that has been opened with AOpen, BOpen, ROpen, UOpen, or WOpen.

Syntax

Close #*fileNumber*

Parameters

fileNumber Integer expression whose value is from 30 - 63.

Description

Closes the file referenced by file handle *fileNumber* and releases it.

See Also

AOpen, BOpen, FreeFile, Input #, Print #, ROpen, UOpen, WOpen

Close Example

This example opens a file, writes some data to it, then later opens the same file and reads the data into an array variable.

```
Real data(100)
Integer fileNumber, i

For i = 0 To 100
    data(i) = i
Next i

fileNumber = FreeFile
WOpen "TEST.VAL" As #fileNumber
For i = 0 To 100
    Print #fileNumber , data(i)
Next i
Close #fileNumber

fileNumber = FreeFile
ROpen "TEST.VAL" As #fileNumber
For i = 0 to 100
    Input #fileNumber , data(i)
Next i
Close #fileNumber
```

CloseCom Statement

S

Close the communication port previously opened with OpenCom.

Syntax

CloseCom #*portNum* | **All**

Parameters

portNum Integer expression for port number to close.

See Also

ChkCom, OpenCom

CloseCom Statement Example

```
CloseCom #1
```

CloseNet Statement

S

Close the network port previously opened with OpenNet.

Syntax

CloseNet #*portNum*

Parameters

portNum Integer expression for port number to close. Range is 128 - 147.

See Also

ChkNet, OpenNet

CloseNet Statement Example

```
CloseNet #1
```

ClrScr Statement

S

Clears the EPSON RC+ Run, Operator, or Monitor window text area.

Syntax

ClrScr

Description

ClrScr clears either the current EPSON RC+ Run or Operator window text area, depending on where the program was started from.

If **ClrScr** is executed from a program that was started from the Monitor window, the monitor window text area is cleared.

ClrScr Example

If this example is run from the Run window or Operator window, the text area of the window will be cleared when **ClrScr** executes.

```
Function main
    Integer i

    Do
        For i = 1 To 10
            Print i
        Next i
        Wait 3
        ClrScr
    Loop
Fend
```

Cnv_AbortTrack Statement

S

Aborts a motion command to a conveyor queue point.

Syntax

Cnv_AbortTrack [*stopZheight*]

Parameters

stopZheight	Optional. Real expression that specifies the Z position the robot should move to after aborting the track.

Description

When a motion command to a conveyor queue point is in progress, **Cnv_AbortTrack** can be executed to abort it.

If *stopZHeight* is ommitted, the robot is decelerated to a stop depending on the current QP setting.

If *stopZHeight* is specified, the robot will move up to this value only if the Z axis position at the time of abort is below *stopZHeight*.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_RobotConveyor Statement

Cnv_AbortTrack Statement Example

```
' Task to monitor robot whose part being tracked has gone downstream

Function WatchDownstream

  Robot 1
  Do
    If g_TrackInCycle And Cnv_QueLen(1, CNV_QUELEN_DOWNSTREAM) > 0 Then
      ' Abort tracking for current robot and move robot Z axis to 0
      g_AbortTrackInCycle = TRUE
      Cnv_AbortTrack 0
      g_AbortTrackInCycle = FALSE
    EndIf
    Wait .01
  Loop
Fend
```

Cnv_Downstream Function

F

Returns the downstream limit for the specified conveyor.

Syntax

Cnv_Downstream (*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

Real value in millimeters.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Upstream

Cnv_Downstream Statement Example

```
Print "Downstream limit: ", Cnv_Downstream(1)
```

Cnv_Fine Statement

S

Sets the value of Cnv_Fine for one conveyor.

Syntax

Cnv_Fine *conveyorNumber* [, *fineValue*]

Parameters

conveyorNumber	Integer expression representing the conveyor number.
fineValue	Optional. Real expression that specifies the distance at which tracking is completed in millimeters. A value of 0 means that Cnv_Fine is not used. If ommitted, the current Cnv_Fine setting is displayed.

Description

Use **Cnv_Fine** to specify the distance from the part that is acceptable before the tracking operation is considered to be complete.

The default value of 0 mm is automatically set when the following conditions occur:
Conveyor is created.
SPEL Drivers are started.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Fine Function

Cnv_Fine Statement Example

```
Cnv_Fine 1, 5
```

Cnv_Fine Function

F

Returns the current Cnv_Fine setting.

Syntax

Cnv_Fine(*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

Real value of Cnv_Fine in millimeters.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Fine Statement

Cnv_Fine Function Example

```
Real f

f = Cnv_Fine(1)
```

Cnv_Name$ Function

F

Returns the name of the specified conveyor.

Syntax

Cnv_Name$ (*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

A string containing the conveyor name.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Number

Cnv_Name$ Function Example

```
Print "Conveyor 1 Name: ", Cnv_Name$(1)
```

Cnv_Number Function

F

Returns the number of a conveyor specified by name.

Syntax

Cnv_Number (*conveyorName*)

Parameters

conveyorName String expression representing the conveyor name.

Return Values

Integer conveyor number.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Name$

Cnv_Number Function Example

```
Integer cnvNum

cnvNum = Cnv_Number("Main Conveyor")
```

Cnv_Point Function

F

Returns a robot point in the specified conveyor's coordinate system derived from sensor coordinates.

Syntax

Cnv_Point (*conveyorNumber, sensorX*, *sensorY [, sensorU]*)

Parameters

conveyorNumber	Integer expression representing the conveyor number.
sensorX	Real expression for the sensor X coordinate.
sensorY	Real expression for the sensor Y coordinate.
sensorU	Optional. Real expression for the sensor U coordinate.

Return Values

Robot point in conveyor coordinate system.

Description

The **Cnv_Point** function must be used to create points that can be added to a conveyor queue. For vision conveyors, *sensorX* and *sensorY* are the vision coordinates from the camera. For sensor conveyors, *sensorX* and *sensorY* can be 0, since this is the origin of the conveyor's coordinate system.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Speed

Cnv_Point Function Example

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
  VGet FindParts.Part.CameraXYU(i), found, x, y, u
  Cnv_QueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_PosErr Function

F

Returns deviation in current tracking position compared to tracking target.

Syntax

Cnv_PosErr (*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

Real value in millimeters.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_MakePoint

Cnv_PosErr Function Example

```
Print "Conveyor 1 position error: ", Cnv_PosErr(1)
```

Cnv_Pulse Function

F

Returns the current position of a conveyor in pulses.

Syntax

Cnv_Pulse (*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

Long value of current pulses for specified conveyor.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Trigger

Cnv_Pulse Function Example

```
Print "Current conveyor position: ", Cnv_Pulse(1)
```

Cnv_QueAdd Statement

S

Adds a robot point to a conveyor queue.

Syntax

Cnv_QueAdd *conveyorNumber*, *pointData* [, *userData*]

Parameters

conveyorNumber	Integer expression that specifies the number of the conveyor to use.
pointData	The robot point to add to the conveyor queue.
userData	Optional. Real expression used to store user data along with the point.

Description

pointData is added to the end of the specified conveyor's queue. It is registered together with the currently latched conveyor pulse position.

If the distance between *pointData* and the previous point in the queue is at or below that specified by Cnv_QueReject, the point data will not be added to the queue, and no error will occur.

The maximum queue data value is 1023.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_RobotConveyor Statement

Cnv_QueAdd Statement Example

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
  VGet FindParts.Part.CameraXYU(i), found, x, y, u
  Cnv_QueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_QueGet Function

F

Returns a point from the specified conveyor's queue.

Syntax

Cnv_QueGet(*conveyorNumber* [, *index*])

Parameters

conveyorNumber	Integer expression representing the conveyor number.
index	Optional. Integer expression representing the index of the queue data to retrieve.

Return Values

A robot point in the specified conveyor's coordinate system.

Description

Use **Cnv_QueGet** to retrieve points from the conveyor queue. When *queNumber* is omitted, the first point in the queue is returned. Otherwise, the point from the specified queNumber is returned.

Cnv_QueGet does not delete the point from the queue. Instead, you must use Cnv_QueRemove to delete it.

To track a part as the conveyor moves, you must use **Cnv_QueGet** in a motion command statement. For example:

```
Jump Cnv_QueGet(1) ' this tracks the part
```

You cannot assign the result from **Cnv_QueGet** to a point and then track it by moving to the point.

```
P1 = Cnv_QueGet(1)
Jump P1  ' this does not track the part
```

When you assign the result from **Cnv_QueGet** to a point, the coordinate values correspond to the position of the part when the point assignment was executed.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueLen, Cnv_QueRemove

Cnv_QueGet Function Example

```
' Jump to the first part in the queue and track it
Jump Cnv_QueGet(1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1
Cnv_QueRemove 1
```

Cnv_QueLen Function

F

Returns the number of items in the specified conveyor's queue.

Syntax

Cnv_QueLen(*conveyorNumber* [, *paramNumber*])

Parameters

conveyorNumber Integer expression representing the conveyor number.

paramNumber Optional. Integer expression that specifies which data to return the length for.

Symbolic constant	Value	Meaning
CNV_QUELEN_ALL	0	Returns total number of items in queue.
CNV_QUELEN_UPSTREAM	1	Returns number of items upstream.
CNV_QUELEN_PICKUPAREA	2	Returns number of items in pickup area.
CNV_QUELEN_DOWNSTREAM	3	Return number of items downstream.

Return Values

Integer number of items.

Description

Cnv_QueLen is used to find out how many items are available in the queue. Typically, who will want to know how many items are in the pick up area.

You can also use Cnv_QueLen as an argment to the Wait statement.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueGet

Cnv_QueLen Function Example

```
Do
  Do While Cnv_QueLen(1, CNV_QUELEN_DOWNSTREAM) > 0
    Cnv_QueRemove 1, 0
  Loop
  If Cnv_QueLen(1, CNV_QUELEN_PICKUPAREA) > 0 Then
    Jump Cnv_QueGet(1, 0) C0
    On gripper
    Wait .1
    Cnv_QueRemove 1, 0
    Jump place
    Off gripper
    Jump idlePos
  EndIf
Loop
```

Cnv_QueList Statement

S

Displays a list of items in the specified conveyor's queue.

Syntax

Cnv_QueList *conveyorNumber,* [*numOfItems*]

Parameters

conveyorNumber	Integer expression representing the conveyor number.
numOfItems	Optional. Integer expression to specify how many items to display. If omitted, all items are displayed.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueGet

Cnv_QueList Statement Example

```
Cnv_QueList 1
```

Cnv_QueMove Statement

S

Moves data from upstream conveyor queue to downstream conveyor queue.

Syntax

Cnv_QueMove *conveyorNumber,* [*index*], [*userData*]

Parameters

conveyorNumber	Integer expression representing the conveyor number.
index	Optional. Integer expression that specifies the index of the first item in the queue to move.
userData	Optional. Real expression used to store user data along with the item.

Description

Cnv_QueMove is used to move one or more items from a conveyor queue to its associated downstream conveyor queue. If *index* is specified, then all items from *index* to the end of the queue are moved.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueGet

Cnv_QueMove Statement Example

```
Cnv_QueMove 1
```

Cnv_QueReject Statement

S

Sets and displays the queue reject distance for a conveyor.

Syntax

Cnv_QueReject *conveyorNumber* [, *rejectDistance*]

Parameters

conveyorNumber	Integer expression representing the conveyor number.
rejectDistance	Optional. Real expression specifying the minimum distance between parts allowed in the queue in millimeters. If omitted, the current *rejectDistance* is displayed.

Description

Use **Cnv_QueReject** to specify the miminum distance between parts to prevent double registration in the queue. As parts are scanned by the vision system, they will be found more than once, but they should only be registered once. **Cnv_QueReject** helps the system filter out double registration.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueReject Function

Cnv_QueReject Statement Example

```
Cnv_QueReject 1, 20
```

Cnv_QueReject Function

F

Returns the current part reject distance for a conveyor.

Syntax

Cnv_QueReject (*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

Real value in millimeters.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueReject Statement

Cnv_QueReject Function Example

```
Real rejectDist

RejectDist = Cnv_QueReject(1)
```

Cnv_QueRemove Statement S

Removes items from a conveyor queue.

Syntax

Cnv_QueRemove *conveyorNumber* [, *index* | All]

Parameters

conveyorNumber	Integer expression representing the conveyor number.
index	Optional. Integer expression specifying the index of the first item to remove or specify All to remove all.

Description

Use Cnv_QueRemove to remove one or more items from a conveyor queue. Typically, you remove items from the queue after you are finished with the data.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueAdd Statement

Cnv_QueRemove Statement Example

```
Jump Cnv_QueGet(1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1

' Remove the data from the conveyor
Cnv_QueRemove 1
```

Cnv_QueUserData Statement

S

Sets and displays user data associated with a queue entry.

Syntax

Cnv_QueUserData *conveyorNumber,* [*index*], [*userData*]

Parameters

conveyorNumber	Integer expression representing the conveyor number.
index	Optional. Integer expression specifying the index of the item number in the queue.
userData	Optional. Real expression specifying user data.

Description

Cnv_QueUserData is used to store your own data with each item in a conveyor queue. User data is optional. It is not necessary for normal operation.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueUserData Function

Cnv_QueUserData Statement Example

```
Cnv_QueUserData 1, 1, angle
```

Cnv_QueUserData Function

F

Returns the user data value associated with an item in a conveyor queue.

Syntax

Cnv_QueUserData (*conveyorNumber* [, *index*])

Parameters

conveyorNumber	Integer expression representing the conveyor number.
index	Optional. Integer expression specifying the index of the item number in the queue.

Return Values

Real value.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueUserData Statement

Cnv_QueUserData Function Example

```
' Add to queue
Cnv_QueAdd 1, Cnv_Point(1, x, y), angle

' Remove from queue
angle = Cnv_QueUserData(1)  ' default to queue index of 0
Jump Cnv_QueGet(1) :U(angle)
Cnv_QueRemove 1
```

Cnv_RobotConveyor Function

F

Returns the conveyor being tracked by a robot.

Syntax

Cnv_RobotConveyor [(*robotNumber*)]

Parameters

robotNumber Integer expression representing the robot number.

Return Values

Integer conveyor number. 0 = no conveyor being tracked.

Description

When using multiple robots, you can use Cnv_RobotConveyor to see which conveyor a robot is currently tracking.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_MakePoint Statement

Cnv_RobotConveyor Function Example

```
Integer cnvNum

cnvNum = Cnv_RobotConveyor(1)
```

Cnv_Speed Function

F

Returns the current speed of a conveyor.

Syntax

Cnv_Speed (*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

Real value in millimeters per second.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Pulse

Cnv_Speed Statement Example

```
Print "Conveyor speed: ", Cnv_Speed(1)
```

Cnv_Trigger Statement

S

Latches current conveyor position for the next Cnv_QueAdd statement.

Syntax

Cnv_Trigger *conveyorNumber*

Parameters

conveyorNumber Integer expression representing the conveyor number.

Description

Cnv_Trigger is a software trigger command that must be used if there is no hardware trigger wired to the PG board for the conveyor encoder.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_QueAdd

Cnv_Trigger Statement Example

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
  VGet FindParts.Part.CameraXYU(i), found, x, y, u
  Cnv_QueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_Upstream Function

F

Returns the upstream limit for the specified conveyor.

Syntax

Cnv_Upstream (*conveyorNumber*)

Parameters

conveyorNumber Integer expression representing the conveyor number.

Return Values

Real value in millimeters.

Note

This command will only work if the Conveyor Tracking option is installed.

See Also

Cnv_Downstream

Cnv_Upstream Statement Example

```
Print "Upstream limit: ", Cnv_Upstream(1)
```

Cont Statement

S

The **Cont** command resumes task execution after a Pause statement has been executed or if a pause has occurred by safeguard open.

Syntax

Cont

Description

Use **Cont** to resume all tasks that have been paused by the Pause statement or by safe guard. When the safeguard is open while tasks are running, all robots will decelerate to a stop and all motors will turn off. After the safeguard has been closed, you can use Cont to resume the cycle. Cont operation after the safeguard has been closed depends on the EPSON RC+ SPEL Options Auto Recover setting.

If Auto Recover is on and Cont is executed, then robot motors are turned on and the robots are slowly moved back to the position they were in when the safeguard was open.

If Auto Recover is off (manual recover) and Cont is executed, then a dialog box is displayed with a Recover button and Continue button. The operator must hold the Recover button down to allow the robot motors to turn back on and slowly move back to their original safeguard open position. Then the operator can click the Continue button to continue the cycle.

See Also

Pause, Recover

Cont Statement Example

```
Function main
    Xqt 2, monitor, NoPause
    Do
        Jump P1
        Jump P2
    Loop
Fend

Function monitor
    Do
        If Sw(pswitch) = On then
            Pause
            Wait Sw(pswitch) = Off
            Cont
        End If
    Loop
Fend
```

Copy Command

>

Copies a file to another location.

Syntax

Copy *source*, *destination*

Parameters

source	Pathname and filename of the source location of the file to copy.
destination	Pathname and filename of the destination to copy the specified source file to.

Description

Copies the specified *source* filename to the specified *destination* filename.

An error occurs if the destination already exists.

Wildcard characters (*, ?) are not allowed in specified filenames.

The same pathname and filename may not be specified for both source and destination files.

When used in the monitor window, quotes and comma may be omitted.

See Also

ChDir, Dir, MkDir

Copy Command Example

```
Function BackupData
    Copy "c:\data\test.dat", "c:\databack\test.dat"
Fend
```

The following example is done from the Monitor window.

```
> copy test.dat test2.dat
```

Cos Function

F

Returns the cosine of a numeric expression.

Syntax

Cos(*number***)**

Parameters

number Numeric expression in Radians.

Return Values

Numeric value in radians representing the cosine of the numeric expression *number*.

Description

Cos returns the cosine of the numeric expression. The numeric expression (*number)* must be in radian units. The value returned by the **Cos** function will range from -1 to 1

To convert from degrees to radians, use the DegToRad function.

See Also

Abs, Atan, Atan2, Int, Mod, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Cos Function Example

The following example shows a simple program which uses **Cos**.

```
Function costest
   Real x
   Print "Please enter a value in radians"
   Input x
   Print "COS of ", x, " is ", Cos(x)
Fend
```

The following examples use **Cos** from the Monitor window.

Display the cosine of 0.55:

```
>print cos(0.55)
 0.852524522059506
>

Display cosine of 30 degrees:

>print cos(DegToRad(30))
 0.866025403784439
>
```

CP Statement

S

Sets CP (Continuous Path) motion mode.

Syntax

CP { **On | Off** }

Parameters

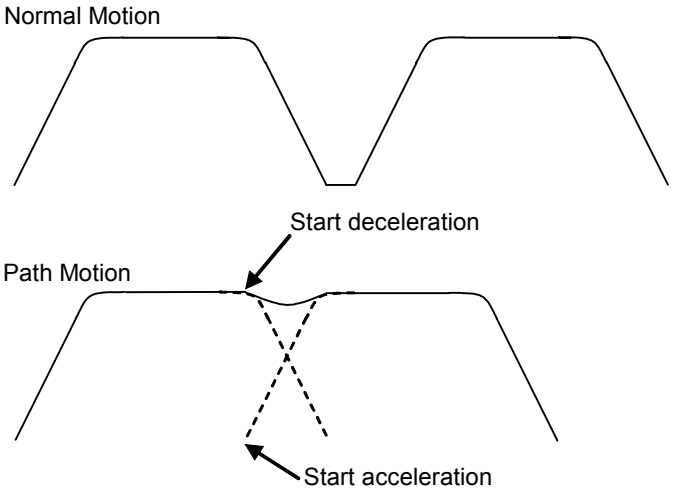
On | Off The keyword On is used to enable path motion. The keyword Off is used to disable path motion.

Description

CP (Continuous Path) motion mode can be used for the Arc, Arc3, Go, Jump, Jump3, Jump3CP, and Move robot motion instructions.

When CP is On, motion command executes the next statement as the deceleration starts. The motion command does not change whether the CP parameter is specified in each motion command.
As a result, themotion track compounds when the next motion starts during the deceleration.

When CP is Off, this function is active only when CP parameter is specified in each motion command.



Two CP motions (Arc, Arc3, Go, Jump, Jump3, Jump3CP, Move), or two PTP motions (Go, Jump) compound the motion track by CP On.
In contrast, continuous track of a CP motion and a PTP motion do not compound the motion track and decelerates before the next command.

CP will be Off in the following cases
- Controller startup
- Reset
- All task stop
- Switching the Auto / Programming operation mode
- MOTOR ON
- SFree, SLock

See Also

CP Function, Arc, Move, Go

CP Statement Example

```
CP On
Move P1
Move P2
CP Off
```

CP Function

F

Returns status of path motion.

Syntax

CP

Return Values

0 = Path motion off, 1 = Path motion on.

See Also

CP Statement

CP Function Example

```
If CP = Off Then
    Print "CP is off"
EndIf
```

Ctr Function

F

Returns the counter value of the specified Hardware Input counter.

Syntax

Ctr(*bitNumber*)

Parameters

bitNumber	Number of the Hardware Input bit set as a counter. Only 16 counters can be active at the same time.

Return Values

The current count of the specified Hardware Input Counter.

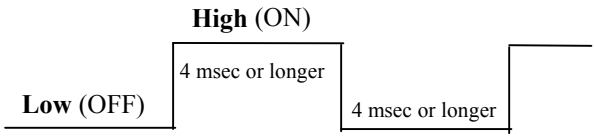
Description

Ctr works with the CTReset statement to allow Hardware inputs to be used as counters.

Each time a hardware input specified as a counter is switched from the Off to On state that input causes the counter to increment by 1.

The **Ctr** function can be used at any time to get the current counter value for any counter input. Any of the Hardware Inputs can be used as counters. However, only 16 counters can be active at the same time.

Counter Pulse Input Timing Chart



See Also

CTReset

Ctr Function Example

The following example shows a sample of code which could be used to get a hardware input counter value.

```
CTReset 3 'Reset counter for input 3 to 0
On 0      'Turn an output switch on

Do While Ctr(3) < 5
Loop
Off 0     'When 5 input cycles are counted for Input 3 turn
          'switch off (output 0 off)
```

CTReset Statement

> S

Resets the counter value of the specified input counter and enables the input to be a counter input.

Syntax

CTReset(*bitNumber*)

Parameters

bitNumber	Number of the input bit set as a counter. This must be an integer expression representing a valid input bit. Only 16 counters can be active at the same time.

Description

CTReset works with the CTR function to allow inputs to be used as counters. CTReset sets the specified input bit as a counter and then starts the counter. If the specified input is already used as a counter, it is reset and started again.

Notes

Turning Off Power and Its Effect on Counters

Turning off main power releases all counters.

Using the Ctr Function

Use the Ctr Function to retrieve current Hardware Input counter values.

See Also

Ctr

CTReset Example

The following example shows a sample of code which could be used to get a hardware input counter value.

```
CTReset 3   'Reset Counter 3 to 0
On 0        'Turn an output switch on
Do While Ctr(3) < 5
Loop
Off 0       'When 5 input cycles are counted for Input 3 turn
            'switch off (output 0 off)
```

CtrlDev Statement

>

Specifies the current control device.

Syntax

CtrlDev [*device*]

Parameters

device **PC** - The PC can start programs. (Default.)

Remote - Programs can be run using remote I/O control.

OP - Programs can be run using OP500RC Operator Pendant.

Description

CtrlDev specifies the controlling device.

PC	Allows the PC to start, pause, continue, stop programs.
Remote	Allows external equipment to start, pause, continue programs using discrete inputs and outputs.
OP	Allows the OP500RC Operator Pendant to start, pause, continue programs.

When no parameter is specified, **CtrlDev** displays the current setting.

You can only execute the **CtrlDev** statement from the Monitor window. You cannot execute the **CtrlDev** statement from within a program. You can also set the control device by selecting Control Device from the Run menu in the EPSON RC+ development environment.

Notes

Verinit resets CtrlDev to default setting

When executing Verinit, the **CtrlDev** value is initialized to its default value: PC.

See Also

CtrlDev Function

CtrlDev Statement Example

The following example shows the syntax required to set the current control device. The following shows the instructions executed from the monitor window.

```
> ctrldev remote   ' External devices can start programs

> ctrldev
Remote
>
```

CtrlDev Function

F

Returns the current control device number.

Syntax

CtrlDev

Return Values

1	**PC**
2	**Remote**
3	**OP**

See Also

CtrlDev Statement

CtrlDev Function Example

```
Print "The current control device is: ", CtrlDev
```

CtrlInfo Function

F

Returns the controller information.

Syntax

CtrlInfo (*paramNumber*)

Parameters

paramNumber Parameter number for the desired information.

Parameter	Information
0	EPSON RC+ Version in numeric format as follows: major version * 1000 + minor version * 100 + revision * 10 + Service Pack number
1	Reserved
2	Reserved
3	Reserved
4	Dryrun status 1 = Dryrun is enabled
5	Reserved
6	Number of robots
10	All drives SMART test result: 0 = Pass, 1 = Fail
11	Primary master drive SMART test result: 0 = Pass, 1 = Fail
12	Primary slave drive SMART test result: 0 = Pass, 1 = Fail
13	Secondary master drive SMART test result: 0 = Pass, 1 = Fail
14	Secondary slave drive SMART test result: 0 = Pass, 1 = Fail
15 - 19	Reserved
20	Approximate amount of memory in use as a percentage
21	Total physical memory for Windows
22	Total available memory for Windows

Return Values

Long value of the desired parameter data

See Also

CtrlDev Command, CtrlDev Function

CtrlInfo Function Example

```
Print "The number of robots is: ", CtrlInfo(5)
```

CurDir$ Function

F

Returns a string representing the current path.

Syntax

CurDir$

Return Values

A string that includes the current drive and path.

See Also

ChDir, CurDrive$

CurDir$ Function Example

```
Print "The current directory is: ", CurDir$
```

CurDrive$ Function

F

Returns a string representing the current drive.

Syntax

CurDrive$

Return Values

A string that contains the current drive letter.

See Also

ChDrive, CurDir$

CurDrive$ Function Example

```
Print "The current drive is: ", CurDrive$
```

CurPos Function

F

Returns the current position of the specified robot.

Syntax

CurPos [(*robotNumber*)]

Parameters

robotNumber	Optional. Specifies which robot to retrieve position data for. If omitted, returns the current position of the current robot.

Return Values

A robot point representing the current position of the specified robot.

See Also

InPos, FindPos

CurPos Function Example

```
Function main

   Xqt showPosition
   Do
      Jump P0
      Jump P1
   Loop
Fend

Function showPosition

   Do
      P99 = CurPos
      Print CX(P99), CY(P99)
   Loop
Fend
```

Curve Statement

S

Defines the data and points required to move the arm along a curved path. Many data points can be defined in the path to improve precision of the path.

Syntax

Curve *fileName, closure, mode, numAxes, pointList*

Parameters

fileName A string expression for the path and name of the file in which the point data is stored. The specified *fileName* will have the extension CRV appended to the end so no extension is to be specified by the user. When the **Curve** instruction is executed, *fileName* will be created.

closure Specifies whether or not the defined **Curve** is Closed or left Open at the end of the curved motion. This parameter must be set to one of two possible values, as shown below.

C - Closed Curve

O - Open Curve

When specifying the open curve, the **Curve** instruction creates the data to stop the arm at the last point of the specified point series. When specifying the closed curve, the **Curve** instruction creates the data required to continue motion through the final specified point and then stopping motion after returning the arm to the starting point of the specified point series for the **Curve** instruction.

mode Specifies whether or not the arm is automatically interpolated in the tangential direction of the U-Axis. It can also specify the ECP number in the upper four bits.

Mode Setting		Tangential Correction	ECP Number
Hexadecimal	Decimal		
&H00	0	No	0
&H10	16		1
&H20	32		2
…	…		…
&HA0	160		10
&HB0	176		11
&HC0	192		12
&HD0	208		13
&HE0	224		14
&HF0	240		15
&H02	2	Yes	0
&H12	18		1
&H22	34		2
…	…		…
&HA2	162		10
&HB2	178		11
&HC2	194		12
&HD2	210		13
&HE2	226		14
&HF2	242		15

When specifying tangential correction, **Curve** uses only the U-Axis coordinate of the starting point of the point series. Tangential correction continuously maintains tool alignment tangent to the curve in the XY plane. It is specified when installing tools such as cutters that require continuous tangential alignment. When specifying a closed curve (using the *closure* parameter) with Automatic Interpolation in the tangential direction of the U-Axis, the U-Axis rotates 360 degrees from the start point. Therefore, before executing the CVMove instruction, set the U-Axis movement range using the Range instruction so the 360 degree rotation of the U-Axis does not cause an error.

When using ECP, specify the ECP number in the upper four bits.

numAxes Integer number 2, 3, 4, or 6 which specifies the number of axes controlled during the curve motion as follows:

2 - Generate a curve in the XY plane with no Z Axis movement or U Axis rotation.

3 - Generate a curve in the XYZ space with no U axis rotation.

4 - Generate a curve in the XYZ space with U-Axis rotation.

6 - Generate a curve in the XYZ space with U, V, and W axes rotation (6-Axis robots only).

The axes not selected to be controlled during the **Curve** motion maintain their previous encoder pulse positions and do not move during **Curve** motion.

pointList { point expression | P(*start*:*finish*) } [, *output command*] ...
This parameter is actually a series of Point Numbers and optional output statements either separated by commas or an ascended range of points separated by a colon. Normally the series of points are separated by commas as shown below:

```
Curve "MyFile", O, 0, 4, P1, P2, P3, P4
```

Sometimes the user defines a series of points using an ascending range of points as shown below:

```
Curve "MyFile", O, 0, 4, P(1:4)
```

In the case shown above the user defined a curve using points P1, P2, P3, and P4. *output command* is optional and is used to control output operation during curve motion. The command can be On or Off for digital outputs or memory outputs. Entering an output command following any point number in the point series causes execution of the output command when the arm reaches the point just before the output command. A maximum of 16 output commands may be included in one **Curve** statement. In the example below, the "On 2" command is executed just as the arm reaches the point P2, then the arm continues to all points between and including P3 and P10.

```
Curve "MyFile", C, 0, 4, P1, P2, ON 2, P(3:10)
```

Description

Curve creates data that moves the manipulator arm along the curve defined by the point series *pointList* and stores the data in a file on the controller. The CVMove instruction uses the data in the file created by **Curve** to move the manipulator in a continuous path type fashion.

Curve calculates independent X, Y, Z, U, V, W coordinate values for each point using a cubic spline function to create the trajectory. Therefore, if points are far apart from each other or the orientation of the robot is changed suddenly from point to point, the desired trajectory may not to be realized.

It is not necessary to specify speeds or accelerations prior to executing the **Curve** instruction. Arm speed and acceleration parameters can be changed anytime prior to executing CVMove by using the SpeedS or AccelS instructions.

Points defined in a local coordinate system may be used in the series to locate the curve at the desired position. By defining all of the specified points in the point series for the **Curve** instruction as points with local attributes, the points may be changed as points on the local coordinate system by the Local instruction following the **Curve** instruction.

Note

Use tangential correction when possible

It is recommended that you use tangential correction whenever possible, especially when using CVMove in a continuous loop throught the same points. If you do not use tangential correction, the robot may not follow the correct path at higher speeds.

Open Curve Min and Max Number of Points Allowed

Open Curves may be specified by using from 3 to 200 points.

Closed Curve Min and Max Number of Points Allowed

Closed Curves may be specified by using from 3 to 50 points.

Mode 1 and 3 no longer needed in SPEL+

Mode 1 and 3 were used in the SPEL language on SRC-3xx controllers to determine whether to decelerate or not after the last point was reached. In SPEL+, this feature has been replaced with the CP parameter for the CVMove statement.

Potential Errors

Attempt to Move Arm Outside Work Envelope

The **Curve** instruction cannot check the movement range for the defined curve path. This means that a user defined path may cause the robot arm to move outside the normal work envelope. In this case an "out of range" error will occur.

See Also

AccelS Function, Arc, CVMove, ECP, Move, SpeedS

Curve Statement Example

The following example designates the free curve data file name as MYCURVE.CRV, creates a curve tracing P1-P7, switches ON output port 2 at P2, and decelerates the arm at P7.

Set up curve

```
> curve "mycurve", O, 0, 4, P1, P2, On 2, P(3:7)
```

Move the arm to P1 in a straight line

```
> jump P1
```

Move the arm according to the curve definition called mycurve

```
> cvmove "mycurve"
```

CVMove Statement

S

Performs the continuous spline path motion defined by the **Curve** instruction.

Syntax

CVMove *fileName* [**CP**] [*searchExpr*]

Parameters

fileName	String expression for the path and name of the file to use for the continuous path motion data. This file must be previously created by the Curve instruction and stored on a PC hard disk.
CP	Optional. Specifies continuous path motion after the last point.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}

Description

CVMove performs the continuous spline path motion defined by the data in the file *fileName*, which is located on the PC hard disk. The file must be previously created with the Curve command. Floppy disks or other slow media should not be used, otherwise hesitation will occur as data is loaded from the file.

Multiple files may exist at the same time on the system. If there is no file name extension, then CRV is assumed.

The user can change the speed and acceleration for the continuous path motion for **CVMove** by using the SpeedS and AccelS instructions.

When the Curve instruction has been previously executed using points with Local definitions, you can change the operating position by using the Local instruction.

When executing CvMove, be careful that the robot doesn't collide with peripheral equipment. When you attempt to change the hand orientation of the 6-axis robot between adjacent points suddenly, due to the nature of cubic spline function, the 6-axis robot may start changing its orientation from the previous and following points and move in an unexpected trajectory. Verify the trajectory thoroughly prior to a CvMove execution and be careful that the robot doesn't collide with peripheral equipment. Specify points closely each other and at equal interval. Do not change the hand orientation between adjacent points suddenly.

CP parameters can start the acceleration of the next motion command when the deceleration starts. In this case the track will not come to the destination coordinate and moves to the next point.

Notes

New Instruction Name is CVMove:

In older versions of SPEL, the instruction was called CVMOV but it was later modified to **CVMove** since most users could better identify the **CVMove** instruction name.

See Also

AccelS Function, Arc, Curve, Move, SpeedS, Till, TillOn

CVMove Statement Example

The following example designates the free curve data file name as MYCURVE.CRV, creates a curve tracing P1-P7, switches ON output port 2 at P2, and decelerates the arm at P7.

Set up curve

```
> curve "mycurve", O, 0, 4, P1, P2, On 2, P(3:7)
```

Move the arm to P1 in a straight line

```
> jump P1
```

Move the arm according to the curve definition called mycurve

```
> cvmove "mycurve"
```

CX, CY, CZ, CU, CV, CW Statements F

Sets the coordinate of a point.

Syntax

CX(*point***)** = *value*
CY(*point***)** = *value*
CZ(*point***)** = *value*
CU(*point***)** = *value*
CV(*point***)** = *value*
CW(*point***)** = *value*

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Real expression representing the new coordinate value in millimeters.

See Also

CX, CY, CZ, CU, CV, CW Functions

CX, CY, CZ, CU, CV, CW StatementS Example

```
CX(pick) = 25.34
```

CX, CY, CZ, CU, CV, CW Functions

F

Retrieves a coordinate value from a point

Syntax

CX(*point***)**
CY(*point***)**
CZ(*point***)**
CU(*point***)**
CV(*point***)**
CW(*point***)**

Parameters

point Point expression.

Return Values

Returns the specified coordinate value. The return values for CX, CY, CZ are real numbers in millimeters. The return values for CU, CV, CW are real numbers in degrees.

Description

Used to retrieve an individual coordinate value from a point.

To obtain the coordinate from the current robot position, use **P*** or **Here** for the point parameter.

See Also

Point expression
CX, CY, CZ, CU, CV, CW Statements

CX, CY, CZ, CU, CV, CW Functions Example

The following example extracts the X axis coordinate value from point "pick" and puts the coordinate value in the variable *x*.

```
Function cxtest
   Real x
   x = CX(pick)
   Print "The X Axis Coordinate of point 'pick' is", x
Fend
```

Date Statement

>

Specifies and displays the current date in the controller.

Syntax

Date *yyyy, mm, dd*
Date

Parameters

yyyy	Integer expression for year.
mm	Integer expression for month.
dd	Integer expression for day.

Return Values

When the **Date** command is entered without any parameters, the current date is displayed.

Description

Specifies the current **Date** for the controller. This date is used for the files inside the controller. All files residing in the controller are date stamped. **Date** automatically calculates the day of the week for the **Date** display.

See Also

Time, Date$

Date Example

The following examples are done from the monitor window.

```
> Date
1999/09/27

> Date 1999,10,1

> Date
1999/10/01
```

Date$ Function

F

Returns the system date.

Syntax

Date$

Return Values

A string containing the date in the format *yyyy/mm/dd*.

Description

Date$ is used to get the controller system date in a program statement. To set the system date, you must use the Date statement.

See Also

Date, GetDate, Time, Time$

Date$ Function Example

```
Function LogErr(errNum As Integer)
  Integer fileNumber

  fileNumber = FreeFile
  AOpen "errlog.dat" As #fileNumber
  Print #fileNumber , "Error ", errNum, " ", ErrMsg$(errNum), " ", Date$, "
", Time$
  Close #fileNumber
Fend
```

Declare Statement

S

Declares an external function in a dynamic link library (DLL).

Syntax

Declare *funcName, "dllFile", "alias"* [, (*argList*)] **As** *type*

Parameters

funcName	The name of the function as it will be called from your program.
dllFile	The path and name of the library file. This must be a literal string (characters delimited by quotation marks). You may also use a macro defined by #define. If there is no path specified, then RC+ will look for the file in the current project directory. If not found, then it is assumed that the file is in the Windows system32 directory. The file extension can be omitted, but is always assumed to be .DLL.
alias	The actual name of the function in the DLL or the function index. The name is case sensitive. The alias must be a literal string (characters delimited by quotation marks). If you use an index, you must use a # character before the index. You may also use a macro defined by #define.
arglist	Optional. List of the DLL arguments. See syntax below.
type	Required. You must declare the type of function.

The arglist argument has the following syntax:

[{**ByRef | ByVal**}] *varName* [( )] **As** *type*

ByRef	Optional. Specify ByRef when you want any changes in the value of the variable to be seen by the calling function.
ByVal	Optional. Specify ByVal when you do not want any changes in the value of the variable to be seen by the calling function. This is the default.
varName	Required. Name of the variable representing the argument; follows standard variable naming conventions. Arrays can be passed for all data types except String and must be ByRef.
type	Required. You must declare the type of argument.

Description

Use Declare to call DLL functions from any program in the current program group. Declare must be used outside of functions.

The Declare statement checks that the DLL file and function exist at compile time.

Passing Numeric Variables ByVal

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a As Long) As Long
VC++  long _stdcall MyDllFunc(long a);
```

Passing String Variables ByVal

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a$ As String) As Long
VC++  long _stdcall MyDllFunc(char *a);
```

Passing Numeric Variables ByRef

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a As Long) As
Long
VC++  long _stdcall MyDllFunc(long *a);
```

Passing String Variables ByRef

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a$ As String) As
Long
VC++  long _stdcall MyDllFunc(char *a);
```

When you pass a string using ByRef, you can change the string in the DLL. Maximum string length is 256 characters. You must ensure that you do not exceed the maximum length.

You must also ensure that space is allocated for the string before calling the DLL. It is best to allocate 256 bytes by using the Space$ function, as shown in the following example.

```
Declare ChangeString, "mystuff.dll", "ChangeString", (ByRef a$ As String) As
Long


    Function main
        String a$

        ' Allocate space for string before calling DLL
        a$ = Space$(256)
        Call ChangeString(a$)
        Print "a$ after DLL call is: ",  a$
    Fend
```

Passing Numeric Arrays ByRef

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a() As Long) As
Long
VC++  long _stdcall MyDllFunc(long *a);
```

Returning Values from DLL Function

The DLL function can return a value for any data type, including String. However, for a string, you must return a pointer to a string allocated in the DLL function. And the function name must end in a dollar sign, as with all SPEL+ string variables and functions. Note that the alias doesn't have a dollar sign suffix.

For example:

```
    Declare ReturnLong, "mystuff.dll", "ReturnLong", As Long
    Declare ReturnString$, "mystuff.dll", "ReturnString", As String

    Function main

        Print "ReturnLong = ", ReturnLong()
        Print "ReturnString$ = ", ReturnString$()
    Fend
```

See Also

Function...Fend

Declare Statement Example

```
' Declare a DLL function. Since there is no path specified,
' the file can be in the current project directory or in
' the Windows system32 directory

Declare MyDLLTest, "mystuff.dll", "MyDLLTest" As Long

Function main
    Print MyDLLTest
Fend

' Declare a DLL function with two integer arguments
' and use a #define to define the DLL file name

#define MYSTUFF "mystuff.dll"

Declare MyDLLCall, MYSTUFF, "MyTestFunc", (var1 As Integer, var2 As
Integer) As Integer

' Declare a DLL function using a path and index.

Declare MyDLLTest, "c:\mydlls\mystuff.dll", "#1" As Long
```

DegToRad Function

> F

Converts degrees to radians.

Syntax

DegToRad(*degrees***)**

Parameters

degrees Real expression representing the degrees to convert to radians.

Return Values

A double value containing the number of radians.

See Also

ATan, ATan2, RadToDeg Function

DegToRad Function Example

```
s = Cos(DegToRad(x))
```

Del Command

>

Deletes one or more files.

Syntax

Del *fileName*

Parameters

fileName	The path and name of the file(s) to delete. The filename should be specified with an extension.

Description

Deletes the specified file(s) from the PC hard disk.

Filenames and extensions may contain wildcard characters such as (* or ?). If *.* is entered in the place of the filename, the following message occurs:

Are You Sure(Y/N) ?

To delete all files in the current directory enter: Y
To cancel the **Del** command enter: N

See Also

Kill

Del Example

The following examples are done from the monitor window using the **Del** command:

```
> Del *.pnt  'Deletes all point files from the current directory
> Del \BAK\*.*  'Deletes all files from the BAK subdirectory
```

Dir Command

>

Displays the contents of the specified directory.

Syntax

Dir [*filename*]

Parameters

fileName	The path and name of the file to search for in the specified path. The filename is used only to display specific files in the specified directory rather than displaying all the files. The filename and extension may contain wildcard characters (*, ?).

Return Values

Displays the contents of the specified directory.

Description

Displays filename, directory name, file size and date, and time stamp for specified directories and files. For those users familiar with DOS, the SPEL+ **Dir** command works similar to the dir command in DOS.

Notes

When path, or filename is omitted:

- If the path is omitted, **Dir** displays the file(s) in the current directory.
- If the filename is omitted, **Dir** displays all files in the current directory. This is equivalent to entering *.* in the place of a filename.

See Also

ChDir, ChDrive

Dir Command Example

The following examples are done from the Monitor window.

```
> Dir

> Dir TEST.*

> Dir *.DAT
```

Dist Function

F

Returns the distance between two robot points.

Syntax

Dist (*point1*, *point2*)

Parameters

point1, point2 Specifies two robot point expressions.

Return Values

Returns the distance between both points (real value in mm).

See Also

CU, CV, CW, CX, CY, CZ

Dist Function Example

```
Real distance

distance = Dist(P1, P2)
```

Do...Loop Statement

S

Repeats a block of statements while a condition is True or until a condition becomes True.

Syntax

Do [{ **While** | **Until** } *condition*]
    [*statements*]
[**Exit Do**]
    [*statements*]
Loop

Or, you can use this syntax:

Do
    [*statements*]
[**Exit Do**]
    [*statements*]
Loop [{ **While** | **Until** } *condition*]

The Do Loop statement syntax has these parts:

Part	Description
condition	Optional. Numeric expression or string expression that is True or False. If *condition* is Null, condition is treated as False.
statements	One or more statements that are repeated while, or until, *condition* is True.

Description

Any number of **Exit Do** statements may be placed anywhere in the **Do...Loop** as an alternate way to exit a **Do...Loop**. **Exit Do** is often used after evaluating some condition, for example, **If...Then**, in which case the **Exit Do** statement transfers control to the statement immediately following the **Loop**.

When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs. Nesting of **Do…Loop** statements is supported up to 256 levels deep including other statements (**If…Then…Else…EndIf**, **Select…Send**, and **While…Wend**).

See Also

For...Next, While...Wend

Do Example

```
ROpen "test.dat" As #30
Do While Not Eof(30)
    Line Input #30, tLine$
    Print tLine$
Loop
Close #30
```

Double Statement

S

Declares variables of type Double. (8 byte double precision number).

Syntax

Double *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameters

varName	Variable name which the user wants to declare as type **Double**.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

Double is used to declare variables as type **Double**. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

See Also

Boolean, Byte, Global, Integer, Long, Real, String

Double Example

The following example shows a simple program which declares some variables using **Double**.

```
Function doubletest
  Double var1
  Double A(10)          'Single dimension array of double
  Double B(10, 10)      'Two dimension array of double
  Double C(10, 10, 10) 'Three dimension array of double
  Double arrayvar(10)
  Integer i
  Print "Please enter a Number:"
  Input var1
  Print "The variable var1 = ", var1
  For I = 1 To 5
    Print "Please enter a Number:"
    Input arrayvar(i)
    Print "Value Entered was ", arrayvar(i)
  Next I
Fend
```

EClr Statement

S

Note: The EClr statement is obsolete and is no longer necessary.

Clears the error status after an error occurs.

Syntax

EClr

Description

EClr is used to clear the error status after an error occurs and is normally used in an error handling subroutine. Normally when an error occurs and there is no OnErr statement (i.e. error handling is not used), the task terminates and the appropriate error code is displayed. However, if an error handler is used to intercept the error (through using the OnErr instruction), then the error must be cleared to allow a successful return from the error handler.

See Also

Err, OnErr, Return

EClr Example

The following example shows a simple utility program which checks whether points P0-P399 exist. If the point does not exist, then a message is printed on the screen to let the user know this point does not exist. The program uses the CX instruction to test each point for whether or not it has been defined. When a point is not defined control is transferred to the error handler and a message is printed on the screen to tell the user which point was undefined.

```
Function eclrtest
   Integer i, errnum
   Real temp

   OnErr Goto eHandle
   For i = 0 To 399
     temp = CX(P(i))
   Next i
   End
'
'
'*******************************************
'* Error Handler                           *
'*******************************************
eHandle:
   errnum = Err(0)
   EClr
   ' Check if using undefined point
   If errnum = 78 Then
      Print "Point number P", i, " is undefined!"
   Else
      Print "ERROR: Error number ", errnum, " Occurred."
   End If
   EResume Next
Fend
```

ECP Statement

> S

Selects or displays the current ECP (external control point).

Syntax

(1) ECP *ECPNumber*
(2) ECP

Parameters

ECPNumber	Optional. Integer expression from 0-15 representing which of 16 ECP definitions to use with subsequent motion instructions. ECP 0 makes the ECP selection invalid.

Return Values

Displays current **ECP** when used without parameters.

Description

ECP selects the external control point specified by the ECPnumber (*ECPNumber*).

Note

This command will only work if the External Control Point option is installed.

Power Off and Its Effect on the ECP Selection

Turning main power off clears the ECP selection.

See Also

ECPSet

ECP Statement Example

```
>ecpset 1, 100, 200, 0, 0
>ecp 1
```

ECP Function

Returns the current ECP (external control point) number.

Syntax

ECP

Return Values

Integer containing the current ECP number.

Note

This command will only work if the External Control Point option is installed.

See Also

ECP Statement

ECP Function Example

```
Integer savECP

savECP = ECP
ECP 2
Call Dispense
ECP savECP
```

ECPClr Statement

> S

Clears (undefines) an external control point.

Syntax

ECPClr *ECPNumber*

Parameters

ECPNumber	Integer expression representing which of the 15 external control points to clear (undefine). (ECP0 is the default and cannot be cleared.)

Note

This command will only work if the External Control Point option is installed.

See Also

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLSet, Verinit

ECPClr Example

```
ECPClr 1
```

ECPSet Statement

> S

Defines or displays an external control point.

Syntax

(1) ECPSet *ECPNum*, *ECPPoint*
(2) ECPSet

Parameters

ECPNum	Integer number from 1-15 representing which of 15 external control points to define.
ECPPoint	**P***number* or **P**(*expr*) or point label or point expression.

Return Values

When parameters are omitted, displays the current **ECPSet** definitions.

Description

Defines an external control point.

Note

This command will only work if the External Control Point option is installed.

ECPSet Example

```
ECPSet 1, P1
ECPSet 2, 100, 200, 0, 0
```

ECPSet Function

F

Returns a point containing the external control point definition for the specified ECP.

Syntax

ECPSet(*ECPNumber*)

Parameters

ECPNumber Integer expression representing the number of the ECP to retrieve.

Return Values

A point containing the ECP definition.

Note

This command will only work if the External Control Point option is installed.

See Also

ECPSet Statement

ECPSet Function Example

```
P1 = ECPSet(1)
```

Elbow Statement

> S

Sets the elbow orientation of a point.

Syntax

(1) Elbow *point*, [*value*]
(2) Elbow

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Integer expression. 1 = Above (/A) 2 = Below (/B)

Return Values

When both parameters are omitted, the elbow orientation is displayed for the current robot position.
If *value* is ommited, the elbow orientation for the specified point is displayed.

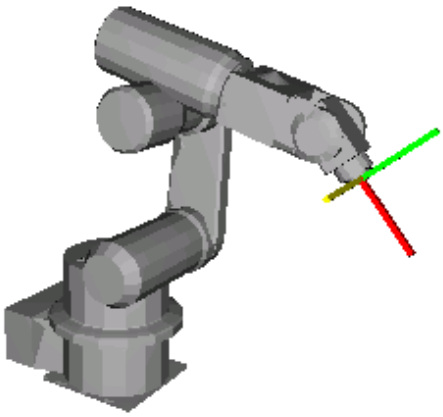
See Also

Elbow Function, Hand, J4Flag, J6Flag, Wrist

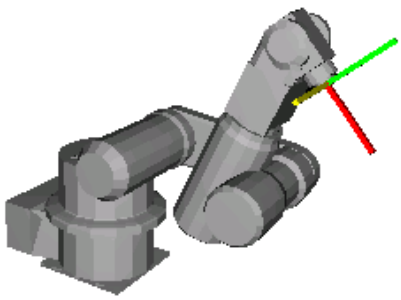
Elbow Statement Example

```
Elbow P0, Below
Elbow pick, Above
Elbow P(myPoint), myElbow

P1 = 0.000,  490.000,  515.000,   90.000,  -40.000,  180.000
```



```
Elbow P1, Above
Go P1
```



```
Elbow P1, Below
Go P1
```

Elbow Function

F

Returns the elbow orientation of a point.

Syntax

Elbow [(*point*)]

Parameters

point Optional. Point expression. If *point* is ommitted, then the elbow orientation of the current robot position is returned.

Return Values

1 Above (/A)
2 Below (/B)

See Also

Elbow Statement, Hand, Wrist, J4Flag, J6Flag

Elbow Function Example

```
Print Elbow(pick)
Print Elbow(P1)
Print Elbow
Print Elbow(P1 + P2)
```

ENetIO_AnaGetConfig Statement

Configures an Ethernet I/O analog input or output module.

Syntax

ENetIO_AnaGetConfig (*channel, gain, offset, loScale, hiScale*)

Parameters

channel	Integer expression or I/O label representing one of the analog input or output ports.
gain	Real variable to receive the input gain. 0 indicates default setting.
offset	Real variable to receive the input offset. 0 indicates default setting.
loScale	Real variable to receive the lowest value.
hiScale	Real variable to receive the highest value.

Description

Use this command to read the configuration for an analog module channel.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_AnaSetConfig

ENetIO_AnaGetConfig Statement Example

```
Real gain, offset, loScale, hiScale

ENetIO_AnaGetConfig 8, gain, offset, loScale, hiScale
loScale = -5
hiScale = 5
ENetIO_AnaSetConfig 8, gain, offset, loScale, hiScale
```

ENetIO_AnaIn Function

Returns the value of the specified EtherNet I/O analog input.

Syntax

ENetIO_AnaIn(*channel*)

Parameters

channel Integer expression or I/O label representing one of the analog input ports.

Return Values

Returns an real value whose range depends on the configuration of the module.

Description

ENetIO_AnaIn is used to read the value of one channel of an analog input module.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_AnaOut, ENetIO_AnaSetConfig, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_AnaIn Function Example

```
Print ENetIO_AnaIn(0)
```

ENetIO_AnaOut Statement

Sets the output level for an analog output channel.

Syntax

ENetIO_AnaOut *channel, value*

Parameters

channel	Integer expression or I/O label representing one of the analog output ports.
value	Real expression representing the value to set the analog output channel. Valid range depends on the configuration of the module.

Description

ENetIO_AnaOut set the value of one channel of an analog output module.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_AnaIn, ENetIO_AnaSetConfig, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_AnaOut Function Example

```
ENetIO_AnaOut 8, 2.5
```

ENetIO_AnaSetConfig Statement

Configures an Ethernet I/O analog input or output module.

Syntax

ENetIO_AnaSetConfig *channel, gain, offset, loScale, hiScale*

Parameters

channel	Integer expression or I/O label representing one of the analog output ports.
gain	Real expression containing the input gain. Set to 0 for default. Normally, gain is used for very precise voltage calibration.
offset	Real expression containing the input offset. Set to 0 for default. Normally, offset is used for very precise voltage calibration.
loScale	Real expression containing the lowest value to be returned.
hiScale	Real expression containing the highest value to be returned.

Description

Use this command to set the configuration for an analog module channel.

You may want to read the configuration values first using ENetIO_AnaGetConfig, then change the required parameters, then call ENetIO_SetConfig.

Gain and Offset

Normally, gain and offset should be set to 0 (factory defaults). If you need to precisely calibrate a module, you can set offset and gain by using a calibrated source. For example, if you need to read -50mv to +50mv, but the module does not read these values when compared with a voltmeter, set the offset and gain so that the readings are accurate.

Scaling

Set the *loScale* and *hiScale* parameters to correspond with the units you need. For example, if you are using an analog input module that reads -10 to +10 volts, you could set *loScale* to 0 and *hiScale* to 100 to read percentage.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_AnaIn, ENetIO_AnaOut

ENetIO_AnaSetConfig Statement Example

```
Real gain, offset, loScale, hiScale

ENetIO_AnaGetConfig 8, gain, offset, loScale, hiScale
loScale = -5
hiScale = 5
ENetIO_AnaSetConfig 8, gain, offset, loScale, hiScale
```

ENetIO_ClearLatches Statement

S

Clears on and off latches for an Ethernet I/O digital input.

Syntax

ENetIO_ClearLatches(*bitNumber*)

Parameters

bitNumber Integer expression representing the bit number.

Description

ENetIO_ClearLatches clears both the on latch and the off latch for the specified point.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_ClearLatches Statement Example

```
ENetIO_ClearLatches 1
```

ENetIO_DigGetConfig Function

F

Returns the configuration for the specified EtherNet I/O digital input.

Syntax

ENetIO_DigGetConfig(*bitNumber*)

Parameters

bitNumber Integer expression representing the bit number.

Return Values

An long integer number containing the current digital input configuration.

0 = counter disabled
1 = Up counter enabled
4 = Quadrature encoder counter enabled
xxgg0004 = Quadrature encoder counter with index. xx is the digital input used for the encoder index (Z phase), and gg specifies leading edge (01) or trailing edge (02) index trigger.

Description

Use **ENetIO_DigGetConfig** to read the current digital input configuration.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_AnaGetConfig, ENetIO_AnaSetConfig, ENetIO_DigSetConfig, ENetIO_AnaOut, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_ReadCount, ENetIO_Sw

ENetIO_DigGetConfig Function Example

```
Long config

config = ENetIO_DigGetConfig(0)
Print "Configuration for input 0 = ", Hex$(config)
```

ENetIO_DigSetConfig Statement

S

Sets the configuration for the specified EtherNet I/O digital input.

Syntax

ENetIO_DigSetConfig(*bitNumber, config*)

Parameters

bitNumber	Integer expression representing the bit number.
config	Long integer expression representing the configuration.
0 =	Counter disabled
1 =	Up counter enabled
4 =	Quadrature encoder counter enabled.
xxgg0004 =	Quadrature encoder counter with index, where xx is the digital input for the encoder index (Z phase), and gg is leading edge (01) or trailing edge (02) index trigger.

Description

Use ENetIO_DigSetConfig to set a digital input's configuration.

Use ENetIO_ReadCount to read the counter value.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_AnaGetConfig, ENetIO_AnaSetConfig, ENetIO_AnaOut, ENetIO_DigGetConfig, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_ReadCount, ENetIO_Sw

ENetIO_DigSetConfig Statement Example

```
' Enable counter for bit 0 digital input
ENetIO_DigSetConfig 0, 1

' Enable quadrature counter for bit 0 digital input
' This input is on a quadrature encoder module
ENetIO_DigSetConfig 0, 4

' Enable quadrature counter with index for bit 0 digital input
' This input is on a quadrature encoder module
' The encoder index input (bit 8) is on a fast input module
ENetIO_DigSetConfig 0, &H08010004
```

ENetIO_In Function

F

Returns the status of the specified EtherNet I/O Input port. Each port contains 4 EtherNet I/O Input channels.

Syntax

ENetIO_In(*portNumber*)

Parameters

portNumber Integer expression representing one of the input ports. Each port contains 4 input channels.

Return Values

Returns an integer value between 0-15. The return value is 4 bits, with each bit corresponding to 1 EtherNet I/O Input channel.

Description

ENetIO_In provides the ability to look at the value of 4 EtherNet I/O Input channels at the same time.

Since 4 channels are checked at a time, the return values range from 0-15. Please review the chart below to see how the integer return values correspond to individual EtherNet I/O Input channels.

ENet Input Channel Result

Return Value	3	2	1	0
1	Off	Off	Off	On
5	Off	On	Off	On
15	On	On	On	On

Note

This command will only work if the EtherNet I/O option is installed.

Response times for Ethernet I/O can vary and depend on several factors, including network load, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, consider using please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_In Function Example

```
Print ENetIO_In(1)
```

ENetIO_IONumber Function

F

Returns the bit number of the specified Ethernet I/O label.

Syntax

ENetIO_IONumber(*label*)

Parameters

label String expression that specifies the Ethernet I/O label.

Return Values

Returns the bit number of the specified Ethernet I/O label. If there is no such Ethernet I/O label, an error will be generated.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

IONumber, FBusIO_IONumber

ENet_IONumber Function Example

```
Integer ENetIObit

ENetIObit = ENetIO_IONumber("myENetIOLabel")

ENetIObit = ENetIO_IONUmber("Station" + Str$(station) + "InCycle")
```

ENetIO_Off

S

Turns an EtherNet I/O output off.

Syntax

ENetIO_Off *bitNumber* [*, time*]

Parameters

bitNumber Integer expression representing the bit number to turn off.

time Optional Parameter. Specifies a time interval in seconds for the output to remain Off. After the time interval expires, the Output is turned back on. (Minimum time interval is 0.01 seconds)

Description

ENetIO_Off turns off (sets to 0) the specified EtherNet I/O Output. If the time interval parameter is specified, the output bit specified by *bitNumber* is switched off, and then switched back on after the time interval elapses. If prior to executing ENetIO_Off, the Output bit was already off, then it is switched On after the time interval elapses.

Note

This command will only work if the EtherNet I/O option is installed.

Response times for Ethernet I/O can vary and depend on several factors, including network load, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, consider using please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_Off Statement Example

```
ENetIO_Off 1
```

ENetIO_On

S

Turns an EtherNet I/O output on.

Syntax

ENetIO_On *bitNumber* [*, time*]

Parameters

bitNumber	Integer expression representing the bit number to turn on.
time	Optional. Specifies a time interval in seconds for the output to remain on. After the time interval expires, the output is turned back off. (Minimum time interval is 0.01 seconds)

Description

ENetIO_On turns on the specified EtherNet I/O output bit. If the *time* interval parameter is specified, the output bit specified by *bitNumber* is switched on, and then switched back off after the *time* interval elapses.

Note

This command will only work if the EtherNet I/O option is installed.

Response times for Ethernet I/O can vary and depend on several factors, including network load, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, consider using please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_On Statement Example

```
ENetIO_On 1
```

ENetIO_Oport Function

F

Returns the state of the specified EtherNet I/O output.

Syntax

ENetIO_Oport(*bitNumber***)**

Parameters

bitNumber Number representing one of the EtherNet I/O outputs.

Return Values

Returns the specified output bit status as either a 0 or 1.

0: Off status
1: On status

Description

ENetIO_Oport provides a method to return the state of an EtherNet I/O output.

Note

This command will only work if the EtherNet I/O option is installed.

Response times for Ethernet I/O can vary and depend on several factors, including network load, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, consider using please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Out, ENetIO_Sw

ENetIO_Oport Statement Example

```
Print ENetIO_Oport(5)
```

ENetIO_Out Statement

S

Simultaneously sets 4 EtherNet I/O outputs.

Syntax

ENetIO_Out *portNumber, outData*

Parameters

portNumber Integer expression between representing one of the output groups (each group contains 4 outputs) which make up the EtherNet I/O outputs.

outData Integer expression between 0-15 representing the output pattern for the output group selected by *portNumber*. If represented in hexadecimal form the range is from &H0 to &HF.

Description

ENetIO_Out simultaneously sets 4 EtherNet I/O outputs using the combination of the *portNumber* and *outData* values specified by the user to determine which outputs will be set.

Once a port number is selected (i.e. a group of 4 outputs has be selected), a specific output pattern must be defined. This is done using the *outData* parameter. The *outData* parameter may have a value between 0-15 and may be represented in Hexadecimal or Integer format. (i.e. &H0-&HF or 0-15)

The table below shows some of the possible output combinations and their associated *outData* values.

Output Settings

outData Value	3	2	1	0
01	Off	Off	Off	On
02	Off	Off	On	Off
03	Off	Off	On	On
08	On	Off	Off	Off
09	On	Off	Off	On
15	On	On	On	On

Note

This command will only work if the EtherNet I/O option is installed.

Response times for Ethernet I/O can vary and depend on several factors, including network load, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, consider using please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Sw

ENetIO_Out Example

```
ENetIO_Out 0,&H0F
```

ENetIO_ReadCount Function

F

Returns the current counter value for the specified EtherNet I/O Input.

Syntax

ENetIO_ReadCount(*bitNumber, clearCounter*)

Parameters

bitNumber Integer expression representing the bit number.

clearCounter Boolean expression specifying whether to clear the counter after reading or not.

Return Values

A long number containing the current counter value.

Description

You use ENetIO_ReadCount to read the current value of an input counter. Use **ENetIO_DigSetConfig** to enable an input counter.

Note

This command will only work if the EtherNet I/O option is installed.

See Also

ENetIO_AnaGetConfig, ENetIO_AnaSetConfig, ENetIO_AnaOut, ENetIO_DigGetConfig, ENetIO_DigSetConfig, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_ReadCount Function Example

```
Long count

count = ENetIO_ReadCount(0, TRUE)
```

ENetIO_Sw Function

F

Returns or displays the selected EtherNet I/O input port status.

Syntax

ENetIO_Sw(*bitNumber***)**

Parameters

bitNumber Number representing one of the EtherNet I/O inputs.

Return Values

Returns a 1 when the specified EtherNet I/O input is On and a 0 when the specified EtherNet I/O input is Off.

Description

ENetIO_Sw provides a status check for the EtherNet I/O inputs. The EtherNet I/O input checked with the **ENetIO_Sw** instruction has 2 states (1 or 0). These indicate whether the input is On or Off.

Note

This command will only work if the EtherNet I/O option is installed.

Response times for Ethernet I/O can vary and depend on several factors, including network load, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, consider using please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

ENetIO_AnaIn, ENetIO_AnaOut, ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out

ENetIO_Sw Function Example

```
Print ENetIO_Sw(5)
```

ENetIO_SwLatch Function

F

Reads EtherNet I/O digital input on or off latch.

Syntax

ENetIO_SwLatch(*bitNumber, latchType*)

Parameters

bitNumber Integer expression or I/O label representing one of the EtherNet I/O digital inputs.

latchType **On | Off**

Description

Each digital input has two latches. The on latch is set if the input ever transitioned from off to on. The off latch is set if the input ever transitioned from on to off. Use **ENetIO_SwLatch** to read a latch state.

Use **ENetIO_ClearLatches** to clear both latches.

Note

This command will only work if the EtherNet I/O option is installed.

Response times for Ethernet I/O can vary and depend on several factors, including network load, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, consider using please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

ENetIO_In, ENetIO_Off, ENetIO_On, ENetIO_Oport, ENetIO_Out, ENetIO_Sw

ENetIO_SwLatch Function Example

```
Boolean latchStatus

latchStatus = ENetIO_SwLatch(0, On)
```

Eof Function

F

Returns end of file status.

Syntax

Eof (*fileNumber)*

Parameters

fileNumber Integer expression representing the file number to check.

Return Values

True if file pointer is at end of file, otherwise False.

See Also

Lof

Eof Example

```
ROpen "test.dat" As #30
Do While Not Eof(30)
    Line Input #30, tLine$
    Print tLine$
Loop
Close #30
```

EPrint Statement

> S

The **EPrint** (end print) command sends data to the PC printer after using the LPrint command.

Syntax

EPrint

Description

Use **EPrint** after executing the last **LPrint** statement.

See Also

LPrint, Print

EPrint Example

```
Function dataPrint
    Integer i

    For i = 1 to 10
        LPrint testDat$(i)
    Next i
    EPrint   ' Send the data to the PC printer
Fend
```

Era Function

F

Returns the joint number for which an error occurred.

Syntax

Era(*taskNum*)

Parameters

taskNum	Integer expression representing a task number from 0-32. 0 specifies the current task.

Return Values

The joint number that caused the error in the range 0-6 as described below:

0 - The current error was not caused by a servo axis.
1 - The error was caused by joint number 1
2 - The error was caused by joint number 2
3 - The error was caused by joint number 3
4 - The error was caused by joint number 4
5 - The error was caused by joint number 5
6 - The error was caused by joint number 6

Description

Era is used when an error occurs to determine if the error was caused by one of the robot joints and to return the number of the joint which caused the error. If the current error was not caused by any joint, **Era** returns zero.

See Also

EClr, Erl, Err, Errhist, ErrMsg$, Ert, OnErr, Trap

Era Function Example

The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erl: where the error occurred; Era: if a joint caused the error.

```
Trap Error Call TrapErrHandler     'Set up for when error occurs
  .
  .
Function TrapErrHandler(errNum As Integer) 'Error handling routine

  Integer errTask

  Print "The Error code is ", errNum
  Print "The Error Message is ", ErrMsg$(errNum)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number at which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused error is ", Era(errTask)
    EndIf
Fend
```

Erase Statement

S

Clear all the elements of an array or a variable.

Syntax

Erase { *varName* | **All** }

Parameters

varName	Name of array or variable to clear.
All	Clears all arrays and variables in the current project.

Description

Erase can be used to clear any variable. Number variables are set to 0 and string variables are set to empty strings.

Erase All is typically used to reset all variables after a Chain is executed. However, when a program is started from the Run window or Operator window, all variables are automatically cleared.

See Also

Redim, UBound

Erase Statement Example

```
Integer i, a(10)

For i= 1 To 10
    a(i) = i
Next i

' Clear all elements of a
Erase a
```

EResume Statement

S

Resumes execution after an error-handling routine is finished.

Syntax

EResume [{ *lineNumber* | *label* | **Next** }]

Description

EResume
If the error occurred in the same procedure as the error handler, execution resumes with the statement that caused the error. If the error occurred in a called procedure, execution resumes at the Call statement in the procedure containing the error handler.

EResume Next
If the error occurred in the same procedure as the error handler, execution resumes with the statement immediately following the statement that caused the error. If the error occurred in a called procedure, execution resumes with the statement immediately following the Call statement that last in the procedure containing the error handler.

EResume { *lineNumber* | *label* }
If the error occurred in the same procedure as the error handler, execution resumes at the statement containing the line number or label.

See Also

OnErr

EResume Statement Example

```
Function LogData(data$ As String)

   OnErr GoTo LogDataErr
   Integer answer
   String msg$

   WOpen "a:\test.txt" As #30
   Print #30, data$
   Close #30

   Exit Function

LogDataErr:
   msg$ = "Cannot open file on floppy." + CRLF
   msg$ = msg$ + "Please insert floppy in drive."
   MsgBox msg$, MB_OK, "Error", answer
   EResume
Fend
```

Erf$ Function

F

Returns the name of the function in which the error occurred.

Syntax

Erf$(*taskNumber*)

Parameters

taskNumber Integer expression representing a task number from 0-32. 0 specifies the current task.

Return Values

The name of thefunction where the last error occurred.

Description

Erf$** is used with OnErr and Trap Error. If an error occurs in a program included within OnErr...Return, it is sometimes difficult to determine the source of the error. Erl returns the line number in which the error occurred. Using **Erf$ combined with Err, Ert, Erl and Era the user can determine much more about the error which occurred.

See Also

EClr, Era, Erl, Err, Errhist, ErrMsg$, Ert, OnErr, Trap

Erf$ Function Example

The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erf$: the name of the function the error occurred in; Erl: the line number where the error occurred; Era: if a joint caused the error....

```
Trap Error Call TrapErrHandler        'Set up for when error occurs
  .
  .
Function TrapErrHandler(errNum As Integer) 'Error handling routine

  Integer errTask

  Print "The Error code is ", errNum
  Print "The Error Message is ", ErrMsg$(errNum)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number at which error occurred is ", errTask
    Print "Function at which error occurred is ", Erf$(errTask)
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused error is ", Era(errTask)
    EndIf
Fend
```

Erl Function

F

Returns the line number in which the error occurred.

Syntax

Erl(*taskNumber*)

Parameters

taskNumber	Integer expression representing a task number from 0-32. 0 specifies the current task.

Return Values

The line number where the last error occurred.

Description

Erl is used with OnErr and Trap Error. If an error occurs in a program included within OnErr...Return, it is sometimes difficult to determine the source of the error. Erl returns the line number in which the error occurred. Using **Erl** combined with Err, Ert and Era the user can determine much more about the error which occurred.

See Also

EClr, Era, Erf$, Err, Errhist, ErrMsg$, Ert, OnErr, Trap

Erl Function Example

The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erl: where the error occurred; Era: if a joint caused the error....

```
Trap Error Call TrapErrHandler        'Set up for when error occurs
  .
  .
Function TrapErrHandler(errNum As Integer) 'Error handling routine

  Integer errTask

  Print "The Error code is ", errNum
  Print "The Error Message is ", ErrMsg$(errNum)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number at which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused error is ", Era(errTask)
    EndIf
Fend
```

Err Function

F

Returns the most recent error status.

Syntax

Err [(*taskNumber*)]

Parameters

taskNumber	Optional. Integer expression representing a task number from 0-32. 0 specifies the current task.

Return Values

Returns a numeric error code in integer form.

Description

Err allows the user to read the current error code. This along with the SPEL$^+$ Error Handling capabilities allows the user to determine which error occurred and react accordingly. Err is used with OnErr. If an error occurs in a program included within OnErr...Return, it may be difficult to determine the error number. However, by using the Err functions this process is made simple.

See Also

EClr, Era, Erf$, Erl, ErrHist, ErrMsg$, EResume, Ert, OnErr, Return, Trap

Err Example

The following example shows a simple utility program which checks whether points P0-P399 exist. If the point does not exist, then a message is printed on the screen to let the user know this point does not exist. The program uses the CX instruction to test each point for whether or not it has been defined. When a point is not defined control is transferred to the error handler and a message is printed on the screen to tell the user which point was undefined.

```
Function errtest
   Integer i, errnum
   Real x

   OnErr Goto eHandle
   For i = 0 To 399
     x = CX(P(i))
   Next i
   Exit Function
'
'
'*****************************************
'* Error Handler                         *
'*****************************************
eHandle:
   errnum = Err
   ' Check if using undefined point
   If errnum = 78 Then
      Print "Point number P", i, " is undefined!"
   Else
      Print "ERROR: Error number ", errnum, " Occurred."
   End If
   EResume Next
Fend
```

ErrHist Command

>

Displays a history of recent errors.

Syntax

ErrHist [{**CLEAR** | *filter*}]

Parameters

CLEAR	Optional. Clears entire history.
filter	Optional. Query filter for history listing. You can specify any string. Any history line that contains the filter string will be displayed.

Description

Displays the most recent errors. The following items are displayed:

- Date and Time Error Occurred
- EPSON RC+ version
- Error Type: System, RunTime, Monitor
- Error Number
- Error Message

These additional items are displayed if applicable:

- Function Name
- Task Number
- Line Number
- Robot Number
- Joint Number

Notes

Errors caught by error handler are not recorded

- If OnErr is used, any error which is caught by OnErr is not recorded.

See Also

EClr, Era, Erl, Err, Errhist, ErrMsg$, Ert, OnErr, Trap

Errhist Example

```
> errhist
11/15/1999 08:17:37, v3.0.0, RunTime, main, 150, Command not allowed with
motors off., Task 1, Line 3, Robot 1
11/15/1999 08:18:24, v3.0.0, RunTime, main, 125, The arm reached the limit
of motion range., Task 1, Line 5, Robot 1, Joint 4
```

Display any errors that contain the word "joint" by using the optional filter parameter.

```
> errhist joint
11/15/1999 08:18:24, v3.0.0, RunTime, main, 125, The arm reached the limit
of motion range., Task 1, Line 5, Robot 1, Joint 4
>
```

ErrMsg$ Function

F

Returns the error message which corresponds to the specified error number.

Syntax

ErrMsg$(*errNumber*)

Parameters

errNumber Integer expression containing the error number to get the message for.

Return Values

Returns the error message which is described in the Error Codes table.

See Also

EClr, Era, Erl, Err, Errhist, Ert, OnErr, Trap

ErrMsg$ Example

The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erl: where the error occurred; Era: if a joint caused the error....

```
Trap Error Call TrapErrHandler      'Set up for when error occurs
  .
  .
Function ER_TrapErrHandler(errNum As Integer) 'Error handling routine

  Integer errTask

  Print "The Error code is ", errNum
  Print "The Error Message is ", ErrMsg$(errNum)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number at which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused error is ", Era(errTask)
    EndIf
Fend
```

Error Statement

S

Generates a user error.

Syntax

Error *errorNumber*

Parameters

errorNumber	Integer expression representing a valid error number. User error numbers start at 30000.

Description

Use the Error statement to generate system or user defined errors. You can define user error labels and descriptions by using the User Error Editor in the EPSON RC+ development environment.

See Also

EClr, Era, Erl, Err, ErrHist, OnErr

Error Statement Example

```
#define ER_VAC 30000

If Sw(vacuum) = Off Then
    Error ER_VAC
EndIf
```

Ert Function

> F

Returns the task number in which an error occurred.

Syntax

Ert

Return Values

The task number in which the error occurred.

Description

Ert is used when an error occurs to determine in which task the error occurs. The number returned will be between 1and 32.

See Also

EClr, Era, Erl, Err, ErrHist, ErrMsg$, OnErr, Trap

Ert Function Example

The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erl: where the error occurred; Err: what error occurred; Era: if a joint caused the error....

```
Trap Error Call TrapErrHandler      'Set up for when error occurs
  .
  .
Function ER_TrapErrHandler(errNum As Integer) 'Error handling routine

  Integer errTask

  Print "The Error code is ", errNum
  Print "The Error Message is ", ErrMsg$(errNum)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused error is ", Era(errTask)
    EndIf
Fend
```

EStopOn Function

F

Returns the current Emergency Stop status.

Syntax

EStopOn

Return Values

True if the Emergency Stop circuit has been broken (ON), otherwise False.

Description

EStopOn returns True after the emergency stop circuit has been tripped. Even after completing the circuit again (usually by pulling out the emergency stop switch), **EStopOn** will return True. You must execute Reset to clear the condition.

EStopOn is equivalent to `((Stat(0) And &H100000) <> 0)`.

See Also

PauseOn, SafetyOn, Stat

EStopOn Function Example

```
Function main

   Trap Emergency Call eTrap

   Do
      Print "Program running..."
      Wait 1
   Loop

Fend

Function eTrap

   String msg$
   Integer ans
   msg$ = "Estop Is on.  Please clear it and Retry"

   Do While EStopOn
      MsgBox msg$, MB_RETRYCANCEL, "Clear E-stop and click Retry", ans
      If ans = IDRETRY Then
         Reset
      Else
         Exit Function
      EndIf
      Wait 1
   Loop
   ' You must get acknowledgement from the user before restarting
   MsgBox msg$, MB_YESNO + MB_ICONQUESTION, "Ready to Restart?", ans
    If ans = IDYES Then
      Restart
    EndIf
Fend
```

Eval Function

F

Executes a monitor statement and returns error status.

Syntax

Eval(*command* [, *reply$*])

Parameters

command	A string expression containing a monitor mode command.
reply$	Optional. A string variable which will contain the reply from the command.

Return Values

The error code returned from executing the command.

Description

Use Eval to execute monitor commands from another device. For example, you can input commands over RS-232 or TCP/IP and then execute them.

Use the *reply$* parameter to retrieve the reply from the command. For example, if the command was "Print Sw(1)", then *reply$* would contain a 1 or 0.

See Also

Error Codes

Eval Function Example

This example shows how to execute a command being read over RS-232. After the command is executed, the error code is returned to the host. For example, the host could send a command like "motor on".

```
Integer errCode
String cmd$

OpenCom #1
Do
  Line Input #1, cmd$
  errCode = Eval(cmd$)
  Print #1, errCode
Loop
```

Exit Statement

S

Exits a loop construct or function.

Syntax

Exit { **Do** | **For** | **Function** }

Description

The Exit statement syntax has these forms:

Statement	Description
Exit Do	Provides a way to exit a Do...Loop statement. It can be used only inside a Do...Loop statement. **Exit Do** transfers control to the statement following the Loop statement. When used within nested Do...Loop statements, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs.
Exit For	Provides a way to exit a For loop. It can be used only in a For...Next loop. **Exit For** transfers control to the statement following the Next statement. When used within nested For loops, Exit For transfers control to the loop that is one nested level above the loop where **Exit For** occurs.
Exit Function	Immediately exits the Function procedure in which it appears. Execution continues with the statement following the statement that called the Function.

See Also

Do...Loop, For...Next, Function...Fend

Exit Statement Example

```
For i = 1 To 10
    If Sw(1) = On Then
        Exit For
    EndIf
    Jump P(i)
Next i
```

FbusIO_GetBusStatus Function

Returns the status of the specified Fieldbus.

Syntax

FbusIO_GetBusStatus(*busNumber*)

Parameters

busNumber Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.

Return Values

0 - OK
1 - Disconnected
2 - Power off

Description

FbusIO_GetBusStatus can be used to verify the general status of the Fieldbus.

Note

This command will only work if the Fieldbus option is installed.

See Also

FbusIO_GetDeviceStatus

FbusIO_GetBusStatus Function Example

```
Long sts
sts = FbusIO_GetBusStatus(1)
```

FbusIO_GetDeviceStatus Function

Returns the status of the specified Fieldbus device.

Syntax

FbusIO_GetDeviceStatus(*busNumber, deviceID*)

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.

Return Values

0 - OK
1 - Disconnected
2 - Power off
3 - Syncronization error. Device is booting, or has incorrect baud rate.

Description

FbusIO_GetDeviceStatus can be used to verify the general status of a Fieldbus device.

Note

This command will only work if the Fieldbus option is installed.

See Also

FbusIO_GetBusStatus

FbusIO_GetDeviceStatus Function Example

```
Long sts
sts = FbusIO_GetDeviceStatus(1, 10)
```

FbusIO_In Function

Returns the status of the specified Fieldbus 8 bit input port.

Syntax

FbusIO_In(*busNumber*, *deviceID, portNumber*)

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
portNumber	Integer expression representing one of the input ports. Keep in mind that each port contains 8 input bits.

Return Values

Returns an integer value between 0-255. The return value is 8 bits, with each bit corresponding to 1 Fieldbus input bit.

Description

FbusIO_In provides the ability to read the value of 8 Fieldbus input bits at the same time.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_Off, FbusIO_On, FbusIO_Oport, FbusIO_Out, FbusIO_Sw

FbusIO_In Function Example

```
' Print the data from port 3 of device 2 on bus 1
   Print FbusIO_In(1, 2, 3)
```

FbusIO_InW Function

Returns the status of the specified Fieldbus 16 bit input port.

Syntax

FbusIO_InW(*busNumber*, *deviceID, portNumber*)

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
portNumber	Integer expression representing one of the input ports. Keep in mind that each port contains 16 input bits.

Return Values

Returns an integer value between 0-65535. The return value is 16 bits, with each bit corresponding to 1 Fieldbus Input bit.

Description

FbusIO_InW provides the ability to look at the value of 16 Fieldbus input bits at the same time.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_Off, FbusIO_On, FbusIO_Oport, FbusIO_Out, FbusIO_Sw,

FbusIO_InW Function Example

```
' Print the data from port 3 of device 2 on bus 1
   Print FbusIO_InW(1, 2, 3)
```

FbusIO_IONumber Function

F

Returns the bit number of the specified Fieldbus I/O label.

Syntax

FbusIO_IONumber*(busNumber, deviceID, label*)

Parameters

busNumber Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.

deviceID Integer expression representing the Fieldbus ID of the device.

label String expression that specifies the Fieldbus I/O label.

Return Values

Returns the bit number of the specified FieldBus I/O label. If there is no such Fieldbus I/O label, an error will be generated.

Note

This command will only work if the Fieldbus option is installed.

See Also

ENetIO_IONumber, IONumber

FbusIO_IONumber Function Example

```
Integer FbusIObit

FbusIObit = FbusIO_IONumber(1, 1, "myFbusIOLabel")

FbusIObit = FbusIO_IONUmber(1, 1, "Station" + Str$(station) + "InCycle")
```

FbusIO_Off Statement

Turns a Fieldbus output bit off.

Syntax

FbusIO_Off *busNumber*, *deviceID, bitNumber* [, *time*]

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
bitNumber	Integer expression representing the bit number to turn off.
time	Optional Parameter. Specifies a time interval in seconds for the output to remain Off. After the time interval expires, the output is turned back on. (Minimum time interval is 0.01 seconds)

Description

FbusIO_Off turns off (sets to 0) the specified Fieldbus output. If the time interval parameter is specified, the output bit specified by outnum is switched off, and then switched back on after the time interval elapses. If prior to executing FbusIO_Off, the Output bit was already off, then it is switched On after the time interval elapses.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_In, FbusIO_On, FbusIO_Oport, FbusIO_Out, FbusIO_Sw

FbusIO_Off Statement Example

```
' Turn off bit 8 of device 2 on bus 1
FbusIO_Off 1, 2, 8
```

FbusIO_On Statement

Turns a Fieldbus output bit on.

Syntax

FbusIO_On *busNumber*, *deviceID, bitNumber* [*, time*]

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
bitNumber	Integer expression representing the bit number to turn on.
time	Optional Parameter. Specifies a time interval in seconds for the output to remain On. After the time interval expires, the Output is turned back off. (Minimum time interval is 0.01 seconds)

Description

FbusIO_On turns On (sets to 1) the specified Fieldbus output. If the *time* interval parameter is specified, the output bit specified by *bitNumber* is switched On, and then switched back Off after the *time* interval elapses.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_In, FbusIO_Off, FbusIO_Oport, FbusIO_Out, FbusIO_Sw

FbusIO_On Statement Example

```
' Turn on bit 8 of device 2 on bus 1
FbusIO_On 1, 2, 8
```

FbusIO_Oport Function

Returns the state a Fieldbus output.

Syntax

FbusIO_Oport(*busNumber*, *deviceID, bitNumber***)**

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
bitNumber	Number representing one of the Fieldbus outputs.

Return Values

Returns the specified output bit status as either a 0 or 1.

0: Off status
1: On status

Description

FbusIO_Oport provides a status check for the Fieldbus outputs.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_In. FbusIO_InW, FbusIO_Off, FbusIO_On, FbusIO_Out, FbusIO_OutW, FbusIO_Sw

FbusIO_Oport Function Example

```
' Print status of bit 5 for device 2 on bus 1
Print FbusIO_Oport(1, 2, 5)
```

FbusIO_Out Statement

Simultaneously sets 8 Fieldbus outputs.

Syntax

FbusIO_Out *busNumber*, *deviceID, portNumber, outData*

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
portNumber	Integer expression between representing one of the output groups (each group contains 8 outputs) which make up the Fieldbus outputs.
outData	Integer number between 0-255 representing the output pattern for the output group selected by *portNumber*. If represented in hexadecimal form the range is from &H0 to &HFF. The lower digit represents the least significant digits (or the 1st 4 outputs) and the upper digit represents the most significant digits (or the 2nd 4 outputs).

Description

FbusIO_Out simultaneously sets 8 Fieldbus outputs using the combination of the *portNumber* and *outData* values specified by the user to determine which outputs will be set.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_In, FbusIO_InW, FbusIO_On, FbusIO_Oport, FbusIO_Off, FbusIO_OutW, FbusIO_Sw

FbusIO_Out Statement Example

```
' Set the value of output port 0 for device 2 on bus 1
FbusIO_Out 1, 2, 0,&H0F
```

FbusIO_OutW Statement

Simultaneously sets 16 Fieldbus output bits.

Syntax

FbusIO_OutW *busNumber*, *deviceID, portNumber, outData*

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
portNumber	Integer expression between representing one of the output groups (each group contains 16 outputs) which make up the Fieldbus outputs.
outData	Long number between 0-65535 representing the output pattern for the output group selected by *portNumber*. If represented in hexadecimal form the range is from &H0 to &HFFFF.

Description

FbusIO_OutW simultaneously sets 16 Fieldbus output bits using the combination of the *portNumber* and *outData* values specified by the user to determine which outputs will be set.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_In, FbusIO_InW, FbusIO_On, FbusIO_Oport, FbusIO_Off, FbusIO_Out, FbusIO_Sw

FbusIO_OutW Statement Example

```
' Set the value of output port 0 for device 2 on bus 1
FbusIO_OutW 1, 2, 0,&H0FFF
```

FbusIO_SendMsg Statement

Sends an explicit message to a Fieldbus device and returns the reply.

Syntax

FbusIO_SendMsg(*busNumber*, *deviceID, msgParam, sendData(), recvData()***)**

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
msgParam	Integer expression for message parameter. Not used with DeviceNet.
sendData	Array of type Byte containing data that is send to the device. This array must be dimensioned to the number of bytes to send. If there are no bytes to send, specify 0.
recvData	Array of type Byte that contains the data received from the device. This array will automatically be redimensioned to the number of bytes received.

Description

FBusIO_SendMsg is used to query one Fieldbus device. Refer to the device manufacturer for information on messaging support.

Note

This command will only work if the Fieldbus option is installed.

See Also

FbusIO_In, FbusIO_InW, FbusIO_On, FbusIO_Oport, FbusIO_Off, FbusIO_Out, FbusIO_OutW

FbusIO_SendMsg Statement Example

```
' Send explicit message to DeviceNet device
Byte sendData(5)
Byte recvData(0)
Integer i

sendData(0) = &H0E  ' Command
sendData(1) = 1     ' Class
sendData(3) = 1     ' Instance
sendData(5) = 7     ' Attribute
' msgParam is 0 for DeviceNet
FbusIO_SendMsg 1, 1, 0, sendData(), recvData()
' Display the reply
For i = 0 to UBound(recvData)
  Print recvData(i)
Next i

' Send message to Profibus device
Byte recvData(0)
Integer i

' msgParam is the service number
FbusIO_SendMsg 2, 1, 56, 0, recvData()
' Display the reply
For i = 0 to UBound(recvData)
  Print recvData(i)
Next i
```

FbusIO_Sw Function

Returns or displays the selected Fieldbus input port status.

Syntax

FbusIO_Sw(*busNumber*, *deviceID, bitNumber***)**

Parameters

busNumber	Integer expression representing the Fieldbus system number. This was set in Setup | System Configuration | I/O Systems.
deviceID	Integer expression representing the Fieldbus ID of the device.
bitNumber	Number representing one of the Fieldbus inputs.

Return Values

Returns a 1 when the specified Fieldbus input is On and a 0 when the specified Fieldbus input is Off.

Description

FbusIO_Sw provides a status check for the Fieldbus inputs. The Fieldbus input checked with the **FbusIO_Sw** instruction has 2 states (1 or 0). These indicate whether the input is On or Off.

Note

This command will only work if the Fieldbus option is installed.

Response times for Fieldbus I/O can vary and depend on several factors, including baud rate, scan rate, number and types of devices, number of SPEL+ tasks, etc. When the fastest and most consistent response times are required, please use EPSON Standard digital I/O, which incorporates interrupt driven inputs and outputs.

See Also

FbusIO_In, FbusIO_InW, FbusIO_On, FbusIO_Oport, FbusIO_Off, FbusIO_Out

FbusIO_Sw Function Example

```
Print FbusIO_Sw(5)
```

FileDateTime$ Function

F

Returns the date and time of a file.

Syntax

FileDateTime$(*fileName*)

Parameters

fileName	A string expression containing the file name to check. The drive and path can also be included.

Return Values

Returns a string containing the date and time of the last modification in the following format:

m/d/yyyy hh:mm:ss

See Also

FileExists, FileLen

FileDateTime$ Function Example

```
Print FileDateTime$("c:\data\myfile.txt")
```

FileExists Function

F

Checks if a file exists.

Syntax

FileExists(*fileName*)

Parameters

fileName	A string expression containing the file name to check. The drive and path can also be included.

Return Values

True if the file exists, False if not.

See Also

FolderExists

FileExists Function Example

```
If FileExists("d:\data\mydata.dat") Then
    Kill "d:\data\mydata.dat"
EndIf
>
```

FileLen Function

F

Returns the length of a file.

Syntax

FileLen(*fileName*)

Parameters

fileName	A string expression containing the file name to check. The drive and path can also be included.

Return Values

Returns the number of bytes in the file.

See Also

FileDateTime$, FileExists

FileLen Function Example

```
Print FileLen("c:\data\mydata.txt")
```

Find Statement

S

Specifies or displays the condition to store coordinates during motion.

Syntax

Find [*condition*]

Parameters

condition Conditional expression that causes the coordinates to be stored.

See Also

FindPos, Go, Jump, PosFound

Find Statement Example

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
End If
```

FindPos Function

F

Returns a robot point stored by Fine during a motion command.

Syntax

FindPos

Return Values

A robot point that was stored during a motion command using Find.

See Also

Find, Go, Jump, PosFound, CurPos, InPos

FindPos Function Example

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
End If
```

Fine Statement

> S

Specifies and displays the positioning accuracy for target points.

Syntax

(1) Fine *axis1, axis2, axis3, axis4,* [*axis5*], [*axis6*]
(2) Fine

Parameters

axis1	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 1st joint.
axis2	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 2nd joint.
axis3	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 3rd joint.
axis4	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 4th joint.
axis5	Optional. Integer expression ranging from (0-65535) which represents the allowable positioning error for the 5th joint.
axis6	Optional. Integer expression ranging from (0-65535) which represents the allowable positioning error for the 6th joint.

Return Values

When used without parameters, **Fine** displays the current fine values for each of the 4 or 6 axes, depending on the robot type.

Description

Fine specifies, for each joint, the allowable positioning error for detecting completion of any given move.

This positioning completion check begins after the CPU in the drive unit has completed sending the target position pulse to the servo system. Due to servo delay, the robot will not yet have reached the target position. This check continues to be executed every few milliseconds until each joint has arrived within the specified range setting. Positioning is considered complete when all axes have arrived within the specified ranges. Once positioning is complete program control is passed to the next command.

When relatively large ranges are used with the Fine instruction, the positioning will be confirmed relatively early in the move, which may cause the target positioning to be rough.

The default **Fine** settings depend on the robot type. Refer to your robot manual for details.

Notes

Cycle Times and the Fine Instruction

The **Fine** value does not affect the acceleration or deceleration control of the manipulator arm. However, smaller **Fine** values can cause the system to run slower because it may take the servo system extra time (a few milliseconds) to get within the acceptable position range. Once the arm is located within the acceptable position range (defined by the **Fine** instruction), the CPU executes the next user instruction. (Keep in mind that all activated axes must be in position before the CPU can execute the next user instruction.)

Initialization of Fine (by Motor On, SLock, SFree and Verinit Instructions)

Any time the following commands are used the Fine value is initialized to default values: SLock, SFree, Motor, Verinit instructions.

Make sure that you reset Fine values after one of the above commands execute.

Potential Errors

If **Fine** positioning is not completed within about 2 seconds, Error 151 will occur. This error normally means the servo system balance needs to be adjusted. **(Call your distributor for assistance)**

See Also

Accel, AccelR, AccelS, Arc, Go, Jump, Move, Speed, SpeedR, SpeedS, Pulse

Fine Statement Example

The examples below show the Fine statement used in a program function, and used from the monitor window.

```
Function finetest
    Fine 5, 5, 5, 5           'reduce precision to +/- 1 Pulse
    Go P1
    Go P2
Fend

> Fine 50, 50, 50, 50
>
> Fine
10, 10, 10, 10
```

Fine Function

F

Returns Fine setting for a specified joint.

Syntax

Fine(*joint*)

Parameters

joint Integer expression representing the joint number for which to retrieve the Fine setting.

Return Values

Real value.

See Also

Accel, AccelS, Arc, Go, Jump, Move, Speed, SpeedS, Pulse

Fine Function Example

This example uses the **Fine** function in a program:

```
Function finetst
    Integer a
    a = Fine(1)
Fend
```

Fix Function

F

Returns the integer portion of a real number.

Syntax

Fix(*number*)

Parameters

number Real expression containing number to fix.

Return Values

An integer value containing the integer portion of the real number.

See Also

Int

Fix Function Example

```
>print Fix(1.123)
 1
>
```

FmtStr Statement

S

Format a numeric or date/time expression.

Syntax

FmtStr *expression, strFormat, strOut$*

Parameters

expression	Expression to be formatted.
strFormat	Format specification string.
strOut$	Output string variable.

Description

Use FmtStr to format a numeric expression into a string.

Numeric Format Specifiers

Character	Description
None	Display the number with no formatting.
(0)	Digit placeholder. Display a digit or a zero. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification.
(#)	Digit placeholder. Display a digit or nothing. If the expression has a digit in the position where the # appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression.
(.)	Decimal placeholder. In some locales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the Number Format recognized by your system.
(%)	Percentage placeholder. The expression is multiplied by 100. The percent character (%) is inserted in the position where it appears in the format string.
(,)	Thousand separator. In some locales, a period is used as a thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (0 or #). Two adjacent thousand separators or a thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed." For example, you can use the format string "##0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent thousand separators in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system.

(:)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
(/)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.
(E- E+)	Scientific format. If the format expression contains at least one digit placeholder (0 or #) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents.
- + $ ( )	Display a literal character. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").
(\)	Display the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (\). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\).Examples of characters that can't be displayed as literal characters are the date-formatting and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, / and :), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !).
("ABC")	Display the string inside the double quotation marks (" "). To include a string in format from within code, you must use Chr$(34) to enclose the text (34 is the character code for a quotation mark ("")).

Date / Time Format Specifiers

Character	Description
(:)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
(/)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.
c	Display the date as ddddd and display the time as ttttt, in that order. Display only date information if there is no fractional part to the date serial number; display only time information if there is no integer portion.
d	Display the day as a number without a leading zero (1–31).
dd	Display the day as a number with a leading zero (01–31).
ddd	Display the day as an abbreviation (Sun-Sat).
dddd	Display the day as a full name (Sunday–Saturday).
ddddd	Display the date as a complete date (including day, month, and year), formatted according to your system's short date format setting. For Microsoft Windows, the default short date format is m/d/yy.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the long date setting recognized by your system. For Microsoft Windows, the default long date format is mmmm dd, yyyy.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday).
ww	Display the week of the year as a number (1–54).

m	Display the month as a number without a leading zero (1–12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01–12). If m immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan–Dec).
mmmm	Display the month as a full month name (January–December).
q	Display the quarter of the year as a number (1–4).
y	Display the day of the year as a number (1–366).
yy	Display the year as a 2-digit number (00–99).
yyyy	Display the year as a 4-digit number (100–9999).
h	Display the hour as a number without leading zeros (0–23).
hh	Display the hour as a number with leading zeros (00–23).
n	Display the minute as a number without leading zeros (0–59).
nn	Display the minute as a number with leading zeros (00–59).
s	Display the second as a number without leading zeros (0–59).
ss	Display the second as a number with leading zeros (00–59).
t t t t t	Display a time as a complete time (including hour, minute, and second), formatted using the time separator defined by the time format recognized by your system. A leading zero is displayed if the leading zero option is selected and the time is before 10:00 A.M. or P.M. For Microsoft Windows, the default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 P.M.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 P.M.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 P.M.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 P.M.
AMPM	Use the 12-hour clock and display the AM string literal as defined by your system with any hour before noon; display the PM string literal as defined by your system with any hour between noon and 11:59 P.M. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as defined by your system settings. For Microsoft Windows, the default format is AM/PM.

See Also

Left$, Right$, Str$

FmtStr Example

```
Function SaveData

    String d$, f$, t$

    ' Make file name in the format
    ' month, day, hour, minute
    d$ = Date$
    t$ = Time$
    d$ = d$ + " " + t$
    FmtStr d$, "mmddhhmm", f$
    f$ = f$ + ".dat"
    WOpen f$ as #30
    Print #30, "data"
    Close #30
Fend
```

FolderExists Function

F

Checks if a folder exists.

Syntax

FolderExists(*pathName*)

Parameters

pathName A string expression containing the path of the folder to check. The drive can also be included.

Return Values

True if the file exists, False if not.

See Also

FileExists

FolderExists Function Example

```
If Not FolderExists("d:\data") Then
    MkDir "d:\data"
EndIf
```

For...Next

S

The For...Next instructions are used together to create a loop where instructions located between For and Next are executed multiple times as specified by the user.

Syntax

For *var* **=** *initValue* **To** *finalValue* **[Step** *increment* **]**
statements
Next [*var*]

Parameters

var	The counting variable used with the For...Next loop. This variable is normally defined as an integer but may also be defined as a Real variable.
initValue	The initial value for the counter *var*.
finalValue	The final value of the counter *var*. Once this value is met, the For...Next loop is complete and execution continues starting with the statement following the Next instruction.
increment	An optional parameter which defines the counting increment for each time the Next statement is executed within the For...Next loop. This variable may be positive or negative. However, if the value is negative, the initial value of the variable must be larger than the final value of the variable. If the increment value is left out the system automatically increments by 1.
statements	Any valid SPEL$^+$ statements can be inserted inside the For...Next loop.

Description

For...Next executes a set of statements within a loop a specified number of times. The beginning of the loop is the **For** statement. The end of the loop is the Next statement. A variable is used to count the number of times the statements inside the loop are executed.

The first numeric expression (*initValue)* is the initial value of the counter. This value may be positive or negative as long as the *finalValue* variable and Step increment correspond correctly.

The second numeric expression (*finalValue*) is the final value of the counter. This is the value which once reached causes the For...Next loop to terminate and control of the program is passed on to the next instruction following the Next instruction.

Program statements after the **For** statement are executed until a Next instruction is reached. The counter variable (*var*) is then incremented by the Step value defined by the *increment* parameter. If the Step option is not used, the counter is incremented by 1 (one).

The counter variable (*var*) is then compared with the final value. If the counter is less than or equal to the final value, the statements following the **For** instruction are executed again. If the counter variable is greater than the final value, execution branches outside of the For...Next loop and continues with the instruction immediately following the Next instruction.

Nesting of For...Next statements is supported up to 16 levels deep. This means that a For...Next Loop can be put inside of another For...Next loop and so on until there are 16 For...Next loops.

Notes

Negative Step Values:

If the value of the Step increment (*increment*) is negative, the counter variable (*var*) is decremented (decreased) each time through the loop and the initial value must be greater than the final value for the loop to work.

Variable Following Next is Not Required:

The variable name following the Next instruction may be omitted. However, for programs that contain nested **For...Next** loops, it is recommended to include the variable name following the Next instruction to aid in quickly identifying loops.

See Also

Do...Loop, While...Wend

For...Next Example

```
Function fornext
    Integer ctr
    For ctr = 1 to 10
        Go Pctr
    Next ctr

    For ctr = 10 to 1 Step -1
        Go Pctr
    Next ctr
Fend
```

Force_Calibrate Statement

S

Sets zero offsets for all axes for the current force sensor.

Syntax

Force_Calibrate

Parameters

On | **Off** Torque Control can be either On or Off.

Description

You should call **Force_Calibrate** for each sensor when your application starts. This will account for the weight of the components mounted on the sensor.

Note

This command will only work if the Force Sensing option is installed.

See Also

Force_Sensor Statement

Force_Calibrate Statement Example

```
Force_Calibrate
```

Force_GetForces Statement

S

Returns the forces and torques for all force sensor axes in an array.

Syntax

Force_GetForces *array()*

Parameters

array()	Real array with upper bound of 6.

Return Values

The array elements are filled in as follows:

Fx Force: 1

Fy Force: 2

Fz Force: 3

Tx Torque: 4

Ty Torque: 5

Tz Torque: 6

Description

Use **Force_**GetForces to read all force and torque values at once.

Note

This command will only work if the Force Sensing option is installed.

See Also

Force_GetForce Statement

Force_GetForces Statement Example

```
Real fValues(6)
Force_GetForces fValues()
```

Force_GetForce Function

F

Returns the force for a specified axis.

Syntax

Force_GetForce (*axis)*

Parameters

axis Integer expression representing the axis.

Axis	Constant	Value
X Force	FORCE_XFORCE	1
Y Force	FORCE_YFORCE	2
Z Force	FORCE_ZFORCE	3
X Torque	FORCE_XTORQUE	4
Y Torque	FORCE_YTORQUE	5
Z Torque	FORCE_ZTORQUE	6

Return Values

Returns an real value.

Description

Use Force_GetForce to read the current force setting for one axis. The units are determined by the type of force sensor.

Note

This command will only work if the Force Sensing option is installed.

See Also

Force_GetForces

Force_GetForce Function Example

```
Print Force_GetForce(1)
```

Force_Sensor Statement

S

Sets the current force sensor for the current task.

Syntax

Force_Sensor *sensorNumber*

Parameters

sensorNumber Integer expression representing the sensor number.

Description

When using multiple force sensors on the same system, you must set the current force sensor before using other force sensing commands.

If your system has only one sensor, then you don't need to use Force_Sensor because the default sensor number is 1.

Note

This command will only work if the Force Sensing option is installed.

See Also

Force_Sensor Function

Force_Sensor Statement Example

```
Force_Sensor 1
```

Force_Sensor Function

F

Returns the current force sensor for the current task.

Syntax

Force_Sensor

Description

Force_Sensor returns the current sensor number for the current task. When a task starts, the sensor number is automatically set to 1.

Note

This command will only work if the Force Sensing option is installed.

See Also

Force_Sensor Statement

Force_Sensor Function Example

```
var = Force_Sensor
```

Force_SetTrigger Statement

F

Sets the force trigger for the Till command.

Syntax

Force_SetTrigger *axis, Threshold, CompareType*

Parameters

axis Integer expression containing the desired force sensor axis.

Axis	Constant	Value
X Force	FORCE_XFORCE	1
Y Force	FORCE_YFORCE	2
Z Force	FORCE_ZFORCE	3
X Torque	FORCE_XTORQUE	4
Y Torque	FORCE_YTORQUE	5
Z Torque	FORCE_ZTORQUE	6

Threshold Real expression containing the desired threshold in units for the sensor being used.

CompareType 0 = Less Than, 1 = Greater Than

Description

To stop motion with a force sensor, you must set the trigger for the sensor, then use Till Force in your motion statement.

You can set the trigger with multiple axes. Call Force_SetTrigger for each axis. To disable an axis, set the threshold at 0.

Note

This command will only work if the Force Sensing option is installed.

See Also

Force_Calibrate

Force_SetTrigger Statement Example

```
'Set trigger to stop motion when force is less than -1 on Z axis.
Force_SetTrigger 3, -1, 0
SpeedS 3
AccelS 5000
Move Place Till Force
```

Force_TC Statement

S

Switches Torque Control On or Off and displays the current mode status.

Syntax

(1) Force_TC { **On** | **Off** }
(2) Force_TC

Parameters

On | **Off** Torque Control can be either On or Off.

Return Values

Displays the current **Force_TC** mode setting when parameter is omitted.

Description

Use Force_TC to turn torque control on or off. You must also set the torque limits using Force_TCLim before torque control is effective.

If you move to a point under torque control, then execute Force_TC Off, you may get an error stating that the current position could not be maintained. This is because without torque control, the robot cannot move to the correct position.

See Also

Force_TCLim Statement

Force_TC Statement Example

```
Speed 5
' Set Z axis torque limit to 20 percent
Force_TCLim -1, -1, 20, -1
Force_TCSpeed 5
Force_TC On
Go Place
Force_TC Off
```

Force_TCLim Statement

S

Sets the torque limits for all axes for torque control.

Syntax

Force_TCLim [*J1TorqueLimit*, *J2TorqueLimit*, *J3TorqueLimit*, *J4TorqueLimit*, [*J5TorqueLimit*], [*J6TorqueLimit*]]

Parameters

J1TorqueLimit	Integer expression representing the torque limit value for joint 1 in percent. Set to -1 to disable.
J2TorqueLimit	Integer expression representing the torque limit value for joint 2 in percent. Set to -1 to disable.
J3TorqueLimit	Integer expression representing the torque limit value for joint 3 in percent. Set to -1 to disable.
J4TorqueLimit	Integer expression representing the torque limit value for joint 4 in percent. Set to -1 to disable.
J5TorqueLimit	Optional. Integer expression representing the torque limit value for joint 5 in percent. Set to -1 to disable.
J6TorqueLimit	Optional. Integer expression representing the torque limit value for joint 6 in percent. Set to -1 to disable.

Return Values

Displays the torque limit values defined for each axis when parameters are omitted.

Description

The torque limit settings are effective when TC mode is On. These settings are a percentage of total torque.

If a setting is too low, the robot may not be able to move. In this case, the motion command will stop before the destination is reached.

Force_TC On must be re-executed after setting Force_TCLim.
The limits are automatically disabled (set to -1) when a Reset, Motor On, or SLock occurs.

See Also

Force_TC Statement, Force_TCLim Function

Force_TCLim Statement Example

```
Speed 5
' Set Z axis torque limit to 20 percent
Force_TCLim -1, -1, 20, -1
Force_TCSpeed 5
Force_TC On
Go Place
```

Force_TCLim Function

F

Returns the current torque limit setting for a specified joint.

Syntax

Force_TCLim(*jointNumber*)

Parameters

jointNumber Integer expression representing the joint number to retrieve the torque limit setting.

Return Values

Integer containing the current torque limit. -1 indicates that torque limit is disabled.

See Also

Force_TCLim Statement

Force_TCLim Function Example

```
Print "The current torque limit for Z is: ", Force_TCLim(3)
```

Force_TCSpeed Statement

F

Sets the speed for torque control.

Syntax

Force_TCSpeed [*speed*]

Parameters

speed Integer expression representing the torque control speed. Range is from 1 - 100.

Description

The Force_TCSpeed is the speed that the robot will move after reaching the torque limit.

See Also

Force_TCLim, Force_TCSpeed Function

Force_TCSpeed Statement Example

```
Speed 5
' Set Z axis torque limit to 20 percent
Force_TCLim -1, -1, 20, -1
Force_TCSpeed 5
Force_TC On
Go Place
```

Force_TCSpeed Function

F

Returns current torque control speed setting.

Syntax

Force_TCSpeed

Return Values

Current Force_TCSpeed setting.

See Also

Force_TCSpeed Statement

Force_TCSpeed Function Example

```
Integer var
var = Force_TCSpeed
```

FreeFile Function

F

Returns a file number that is currently not being used.

Syntax

FreeFile

Return Values

Integer between 30 and 63.

See Also

AOpen, BOpen, ROpen, WOpen

FreeFile Function Example

```
Integer f
f = FreeFile
ROpen "test.dat" As #f
```

Function...Fend

S

A function is a group of program statements which includes a **Function** statement as the first statement and a Fend statement as the last statement.

Syntax

Function *funcName* [(*argList*)] [**As** *type*]
statements
Fend

Parameters

funcName	The name which is given to the specific group of statements bound between the **Function** and **Fend** instructions. The function name must contain alphanumeric characters and may be up to 32 characters in length. Underscores are also allowed.
argList	Optional. List of variables representing arguments that are passed to the Function procedure when it is called. Multiple variables are separated by commas.

The arglist argument has the following syntax:

[{**ByRef | ByVal**}] *varName* [( )] **As** *type*

ByRef	Optional. Specify ByRef when you want any changes in the value of the variable to be seen by the calling function.
ByVal	Optional. Specify ByVal when you do not want any changes in the value of the variable to be seen by the calling function. This is the default.
varName	Required. Name of the variable representing the argument; follows standard variable naming conventions.
As *type*	Required. You must declare the type of argument.

Return Values

Value whose data type is specified with the **As** clause at the end of the function declaration.

Description

The **Function** statement indicates the beginning of a group of SPEL$^+$ statements. To indicate where a function ends we use the **Fend** statement. All statements located between the **Function** and **Fend** statements are considered part of the function.

The **Function...Fend** combination of statements could be thought of as a container where all the statements located between the **Function** and **Fend** statements belong to that function. Multiple functions may exist in one program file.

See Also

Call, Fend, Halt, Quit, Return, Xqt

Function...Fend Example

The following example shows 3 functions which are within a single file. The functions called task2 and task3 are executed as background tasks while the main task called main executes in the foreground.

```
Function main
  Xqt 2, task2 'Execute task2 in background
  Xqt 3, task3 'Execute task3 in background
  '....more statements here
Fend

Function task2
  While TRUE
    On 1
    On 2
    Off 1
    Off 2
  Wend
Fend

Function task3
  While TRUE
    On 10
    Wait 1
    Off 10
  Wend
Fend
```

GetCurrentUser$ Function

F

Returns the current EPSON RC+ user.

Syntax

GetCurrentUser$

Return Values

String containing the current user logID.

Note

This command will only work if the Security option is installed.

See Also

LogIn Statement

GetCurrentUser$ Function Example

```
String currUser$

currUser$ = GetCurrentUser$
```

Global Statement

S

Declares variables with the global scope. Global variables can be accessed from anywhere.

Syntax

Global [**Preserve**] *dataType varName* [**(***subscripts***)**] [**,** *varName* [**(***subscripts***)**] **,** ...]

Parameters

Preserve	Optional. If Preserve is specified, then the variable retains its values by saving its contents to disk. The values are cleared only with Erase or if the number of dimensions is changed. The values are preserved between sessions or project changes.
dataType	Data type including Boolean, Integer, Long, Real, Double, Byte, or String.
varName	Variable name. Names may be up to 24 characters in length.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

Global variables are variables which can be used in more than 1 file within the same project. They are cleared whenever a function is started from the Run window or Operator window unless they are declared with the Preserve option.

Global variables can used in the monitor window after the project build is complete.

Global variables can also be used with the VB Guide option.

It is recommended that global variable names begin with a "g_" prefix to make it easy to recognize globals in a program. For example:

```
Global Long g_PartsCount
```

See Also

Boolean, Byte, Double, Integer, Long, Real, String

Global Statement Example

The following example shows 2 separate program files. The first program file defines some global variables and initializes them. The second file then uses these global variables.

FILE1 (MAIN.PRG)

```
Global Integer status1
Global Real numsts

Function Main
    Integer I

    status1 = 10
```

The following example shows 2 separate program files. The first program file defines some global variables and initializes them. The second file then also uses these global variables.

FILE1 (MAIN.PRG)

```
Global Integer g_Status
Global Real g_MaxValue

Function Main

    g_Status = 10
    g_MaxValue = 1.1
    .
    .
Fend
```

FILE2 (TEST.PRG)

```
Function Test

    Print "status1 = , g_Status
    Print "MaxValue = , g_MaxValue
    .
    .
Fend
```

Go Statement

> S

Moves the arm using point to point motion from the current position to the specified point or X,Y,Z,U, V, W position. The Go instruction can move any combination of 1-6 joints at the same time.

Syntax

Go destination [**CP**] [searchExpr] [!...!]

Parameters

destination	The target destination of the motion using a point expression.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to execute I/O and other commands during motion.

Description

Go simultaneously moves all joints of the robot arm using point to point motion. The destination for the Go instruction can be defined in a variety of ways:

- Using a specific point to move to. For example: **Go P1**.
- Using an explicit coordinate position to move to. For example: **Go XY(50, 400, 0, 0)**.
- Using a point with a coordinate offset. For example: **Go P1 +X(50)**.
- Using a point but with a different coordinate value. For example: **Go P1 :X(50)**.

The Speed instruction determines the arm speed for motion initiated by the Go instruction. The Accel instruction defines the acceleration.

The **Go** instruction cannot check whether or not there is an area in which the arm cannot physically move to prior to starting the move itself. Therefore, it is possible for the controller to find a prohibited position along the way to a target point. In this case, the arm may abruptly stop which may cause shock and a servo out condition of the arm. Even though the target position is within the range of the arm, there are some moves which will not work because the arm cannot physically make it to some of the intermediate positions required during the move.

The **Go** instruction always causes the arm to decelerate to a stop prior to reaching the final destination of the move.

Notes

Difference between Go and Move

The Move instruction and the **Go** instruction each cause the robot arm to move. However, the primary difference between the 2 instructions is that the **Go** instruction causes point to point motion where as the Move instruction causes the arm to move in a straight line. The **Go** instruction is used when the user is primarily concerned with the orientation of the arm when it arrives on point. The Move instruction is used when it is important to control the path of the robot arm while it is moving.

Difference between Go and Jump

The Jump instruction and the **Go** instruction each cause the robot arm to move in a point to point type fashion. However, the JUMP instruction has 1 additional feature. Jump causes the robot end effector to first move up to the LimZ value, then in a horizontal direction until it is above the target point, and then finally down to the target point. This allows Jump to be used to guarantee object avoidance and more importantly to improve cycle times for pick and place motions.

Proper Speed and Acceleration Instructions with Go

The Speed and Accel instructions are used to specify the speed and acceleration of the manipulator during motion caused by the **Go** instruction. Pay close attention to the fact that the Speed and Accel instructions apply to point to point type motion (like that for the **Go** instruction) while linear and circular interpolation motion uses the SpeedS and AccelS instructions.

Using Go with the Optional Till Modifier

The optional Till modifier allows the user to specify a condition to cause the robot to decelerate to a stop at an intermediate position prior to completing the motion caused by the **Go** instruction. If the Till condition is not satisfied, the robot travels to the target position. The Go with Till modifier can be used in 2 ways as described below:

(1) Go with Till Modifier

Checks if the current Till condition becomes satisfied. If satisfied, this command completes by decelerating and stopping the robot at an intermediate position prior to completing the motion caused by the **Go** instruction.

(2) Go with Till Modifier, Sw(Input bit number) Modifier, and Input Condition

This version of the Go with Till modifier allows the user to specify the Till condition on the same line with the Go instruction rather than using the current definition previously defined for Till. The condition specified is simply a check against one of the inputs. This is accomplished through using the Sw instruction. The user can check if the input is On or Off and cause the arm to stop based on the condition specified. This feature works almost like an interrupt where the motion is interrupted (stopped) once the Input condition is met. If the input condition is never met during the robot motion then the arm successfully arrives on the point specified by *destination.*

Using Go with the Optional Find Modifier

The optional Find modifier allows the user to specify a condition to cause the robot to record a position during the motion caused by the **Go** instruction. The Go with Find modifier can be used in 2 ways as described below:

(1) Go with Find Modifier:
Checks if the current Find condition becomes satisfied. If satisfied, the current position is stored in the special point FindPos.

(2) Go with Find Modifier, Sw(Input bit number) Modifier, and Input Condition:
This version of the Go with Find modifier allows the user to specify the Find condition on the same line with the Go instruction rather than using the current definition previously defined for Find. The condition specified is simply a check against one of the inputs. This is accomplished through using the Sw instruction. The user can check if the input is On or Off and cause the current position to be stored in the special point **FindPos**.

Go Instruction Always Decelerates to a Stop

The **Go** instruction always causes the arm to decelerate to a stop prior to reaching the final destination of the move.

Potential Errors

Attempt to Move Outside of Robots Work Envelope
When using explicit coordinates with the **Go** instruction, you must make sure that the coordinates defined are within the robots valid work envelope. Any attempt to move the robot outside of the valid work envelope will result in an error.

See Also

!...! Parallel Processing, Accel, Find, Jump, Move, Pass, Pn= (Point Assignment), Pulse, Speed, Sw, Till

Go Example

The example shown below shows a simple point to point move between points P0 and P1 and then moves back to P0 in a straight line. Later in the program the arm moves in a straight line toward point P2 until input #2 turns on. If input #2 turns On during the Move, then the arm decelerates to a stop prior to arriving on point P2 and the next program instruction is executed.

Function sample

```
  Integer i

  Home
  Go P0
  Go P1
  For i = 1 to 10
    Go P(i)
  Next i
  Go P2 Till Sw(2) = On
  If Sw(2) = On Then
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
  Else
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
  End If
Fend

Some syntax examples from the monitor window are shown below:

>Go P* +X(50)          ' Move only in the X direction 50 mm from
                       ' current position
>Go P1                 ' Simple example to move to point P1
>Go P1 :U(30)          ' Move to P1 but use +30 as the position for
                       ' the U joint to move to
>Go P1 /L              ' Move to P1 but make sure the arm ends up
                       ' in lefty position
>Go XY(50, 450, 0, 30) ' Move to position X=50, Y=450, Z=0, U=30

<Another Coding Example>
Till Sw(1)=0 And Sw(2) = On  ' Specifies Till conditions for
                             ' inputs 1 & 2
Go P1 Till                   ' Stop if current Till condition
                             ' defined on previous line is met
Go P2 Till Sw(2) = ON        ' Stop if Input Bit 2 is On
Go P3 Till                   ' Stop if current Till condition
                             ' defined on previous line is met
```

GoSub...Return

S

GoSub transfers program control to a subroutine. Once the subroutine is complete, program control returns back to the line following the **GoSub** instruction which initiated the subroutine.

Syntax

GoSub {*lineNum* | *label*}

{*lineNum* | *label***:**}
statements
Return

Parameters

lineNum	This is the line number to which program execution will jump to when the **GoSub** instruction is executed. Valid line numbers are between 0001-32767. Any valid line number is acceptable.
label	Rather than using a line number the user can also specify a label (recommended). When the user specifies a label (rather than a line number) the program execution will jump to the line on which this label resides. The label can be up to 24 characters in length. However, the first character must be an alphabet character (not numeric).

Description

The **GoSub** instruction causes program control to branch to the user specified statement line number or label. The program then executes the statement on that line and continues execution through subsequent line numbers until a Return instruction is encountered. The Return instruction then causes program control to transfer back to the line which immediately follows the line which initiated the **GoSub** in the first place. (i.e. the **GoSub** instruction causes the execution of a subroutine and then execution returns to the statement following the **GoSub** instruction.) Be sure to always end each subroutine with Return. Doing so directs program execution to return to the line following the **GoSub** instruction.

Potential Errors

Branching to Non-Existent Statement

If the **GoSub** instruction attempts to branch control to a non-existent line number or label then an Error 8 will be issued.

Nesting Level for GoSubs is Too Deep

GoSub instructions may be nested up to 16 deep. This means that from within a subroutine another **GoSub** can be executed (and so on) up to 16 times. However, once 16 nested **GoSub** instructions are executed, an attempt to execute an 17th **GoSub** will cause an Error 7 to occur.

Return Found Without GoSub

A Return instruction is used to "return" from a subroutine back to the original program which issued the **GoSub** instruction. If a Return instruction is encountered without a **GoSub** having first been issued then an Error 3 will occur. A stand alone Return instruction has no meaning because the system doesn't know where to Return to.

See Also

GoTo, OnErr, Return

GoSub Statement Example

The following example shows a simple function which uses a GoSub instruction to branch to a label and execute some I/O instructions then return.

```
Function main
    Integer var1, var2

    GoSub checkio  'GoSub using Label
    On 1
    On 2
    Exit Function

checkio:    'Subroutine starts here
    var1 = In(0)
    var2 = In(1)
    If var1 = 1 And var2 = 1 Then
        On 1
    Else
        Off 1
    EndIf
    Return  'Subroutine ends here
Fend
```

GoTo Statement

S

The **GoTo** instruction causes program control to branch unconditionally to a designated statement line number or label.

Syntax

GoTo { *lineNumber* | *label* }

Parameters

lineNumber	This is the line number to which program execution will jump to when the **GoTo** instruction is executed. Valid line numbers are between 0001-32767. Any valid line number is acceptable.
label	Rather than using a line number the user can also specify a line label. When the user specifies a label, program execution will jump to the line on which the label resides. The label can be up to 32 characters. However, the first character must be an alphabetic character (not numeric).

Description

The **GoTo** instruction causes program control to branch to the user specified statement line number or label. The program then executes the statement on that line and continues execution from that line on. **GoTo** is most commonly used for jumping to an exit label because of an error.

Notes

Using Too Many GoTo's

Please be careful with the **GoTo** instruction since using too many **GoTo**'s in a program can make the program difficult to understand. The general rule is to try to use as few **GoTo** instructions as possible. Some **GoTo**'s are almost always necessary. However, jumping all over the source code through using too many **GoTo** statements is an easy way to cause problems.

See Also

GoSub, OnErr

GoTo Statement Example

The following example shows a simple function which uses a GoTo instruction to branch to a line label.

```
Function main

    If Sw(1) = Off Then
        GoTo mainAbort
    EndIf
    Print "Input 1 was On, continuing cycle"
    .
    .
    Exit Function

mainAbort:
    Print "Input 1 was OFF, cycle aborted!"
Fend
```

Halt Statement

S

Temporarily suspends execution of a specified task.

Syntax

Halt *taskIdentifier*

Parameters

taskIdentifier	Task name or integer expression representing the task number. A task name is the function name used in an Xqt statement or a function started from the Run window or Operator window. If an integer expression is used, the range is from 1 to 32.

Description

Halt temporarily suspends the task being executed as specified by the task name or number.

To continue the task where it was left off, use Resume. To stop execution of the task completely, use Quit. To display the task status, click the Task Manager Icon on the EPSON RC+ Toolbar to run the Task manager.

See Also

Quit, Resume, Xqt

Halt Statement Example

The example below shows a function named "flicker" that is started by Xqt, then is temporarily stopped by Halt and continued again by Resume.

```
Function main
    Xqt 2, flicker   'Execute flicker function

  Do
    Wait 3           'Execute task flicker for 3 seconds
    Halt flicker

    Wait 3           'Halt task flicker for 3 seconds
    Resume flicker

  Loop
Fend
    '

Function flicker
  Do
    On 1
    Wait 0.2
    Off 1
    Wait 0.2
  Loop
Fend
```

Hand Statement

> S

Sets the hand orientation of a point.

Syntax

(1) Hand *point*, [*value*]
(2) Hand

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Integer expression. 1 = Righty (/R) 2 = Lefty (/L)

Return Values

When both parameters are omitted, the hand orientation is displayed for the current robot position.
If *value* is ommited, the hand orientation for the specified point is displayed.

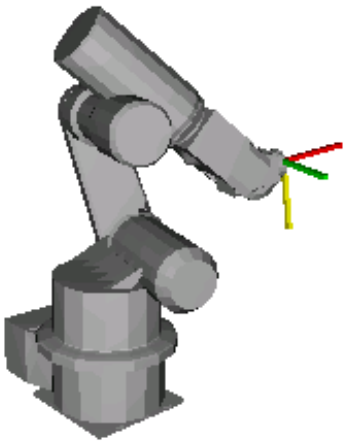
See Also

Elbow, Hand Function, J4Flag, J6Flag, Wrist

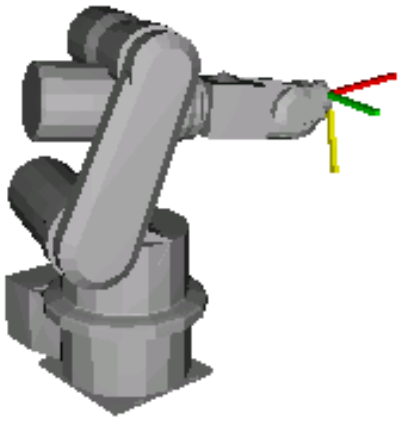
Hand Statement Example

```
Hand P0, Lefty
Hand pick, Righty
Hand P(myPoint), myHand

P1 = -364.474, 120.952, 469.384, 72.414, 1.125, -79.991
```



```
Hand P1, Righty
Go P1
```



```
Hand P1, Lefty
Go P1
```

Hand Function

F

Returns the hand orientation of a point.

Syntax

Hand [(*point*)]

Parameters

point Optional. Point expression. If *point* is ommitted, then the hand orientation of the current robot position is returned.

Return Values

1 Righty (/R)
2 Lefty (/L)

See Also

Elbow, Wrist, J4Flag, J6Flag

Hand Function Example

```
Print Hand(pick)
Print Hand(P1)
Print Hand
Print Hand(P1 + P2)
```

Here Statement

S

Teach a robot point at the current position.

Syntax

Here *point*

Parameters

point **P***number* or **P**(*expr*) or point label.

See Also

Here Function

Here Statement Example

```
Here P1    ' Same as P1 = P*
Here pick  ' Same as pick = P*
```

Here Function

F

Returns current robot position as a point.

Syntax

Here

Return Values

A point representing the current robot position.

Description

Use **Here** to retrieve the current position of the current manipulator. This is equivalent to P*.

See Also

Here Statement

Here Function Example

```
P1 = Here
```

Hex$ Function

> F

Returns a string representing a specified number in hexadecimal format.

Syntax

Hex$*(number)*

Parameters

number Integer expression.

Return Values

Returns a string containing the ASCII representation of the number in hexadecimal format.

Description

Hex$** returns a string representing the specified number in hexadecimal format. Each character is from 0-9 or A-F. **Hex$ is especially useful for examining the results of the Stat function over an RS232 port.

See Also

Str$, Stat, Val

Hex$ Function Example

```
> print hex$(stat(0))
A00000
> print hex$(255)
FF
```

Hofs Statement

> S

Displays or sets the offset pulses between the encoder origin and the home sensor.

Syntax

(1) Hofs *j1Pulses, j2Pulses, j3Pulses, j4Pulses*
(2) Hofs

Parameters

j1Pulses	Integer expression representing joint 1 offset pulses.
j2Pulses	Integer expression representing joint 2 offset pulses.
j3Pulses	Integer expression representing joint 3 offset pulses.
j4Pulses	Integer expression representing joint 4 offset pulses.

Return Values

Displays current Hofs values when used without parameters.

Description

Hofs displays or sets the home position offset pulses. When the robot returns the axes to their home positions, the controller checks the position by the home sensor and the encoder Z phase. Ordinarily, however, the values of the home sensor and the encoder Z phase are different. Therefore, **Hofs** is used to define this offset value. (**Hofs** specifies the offset from the encoder 0 point to the mechanical 0 point.)

Although the robot motion control is based on the zero point of the encoder mounted on each joint motor, the encoder zero point may not necessarily match the robot mechanical zero point. The **Hofs** offset pulse correction pulse is used to carry out a software correction to the mechanical 0 point based on the encoder 0 point.

Note

Hofs Values SHOULD NOT be Changed unless Absolutely Necessary

The Hofs values are correctly specified prior to delivery. There is a danger that unnecessarily changing the Hofs value may result in position errors and unpredictable motion. Therefore, it is **strongly recommended** that Hofs values not be changed unless absolutely necessary.

To Automatically Calculate Hofs Values

To have **Hofs** values automatically calculated, move the arm to the desired calibration position, and execute Calib. The controller then automatically calculates Hofs values based on the CalPls pulse values and calibration position pulse values.

Saving and Restoring Hofs

Hofs can be saved and restored using the Mkver and SetVer commands in the Maintenance dialog from the Tools menu.

See Also

Calib, CalPls, Home, Hordr, HTest, Mcal, Ver

Hofs Statement Example

These are simple examples on the monitor window that first sets the joint 1 home offset value to be -545, the joint 2 home offset value to be 514, and the joint 3 and the joint 4 Home offset values to be both 0. It then displays the current home offset values.

```
> hofs  -545, 514, 0, 0

> hofs
-545, 514, 0, 0
>
```

Hofs Function

F

Returns the offset pulses used for software zero point correction.

Syntax

Hofs(*jointNumber*)

Parameters

jointNumber Integer expression representing the joint number to retrieve the Hofs value for.

Return Values

The offset pulse value (integer value, in pulses).

See Also

Calib, CalPls, Home, Hordr, HTest, Mcal, Ver

Hofs Function Example

This example uses the **Hofs** function in a program:

```
Function DisplayHofs
   Integer i

   Print "Hofs settings:"
   For i = 1 To 4
      Print "Joint ", i, " = ", Hofs(i)
   Next i
Fend
```

Home Statement

> S

Moves the robot arm to the user defined home position.

Syntax

Home

Description

Executes low speed Point to Point motion to the Home (standby) position specified by HomeSet, in the homing order defined by Hordr. It is very important to note that the Home command does not calibrate the robot.

Calibration is done by the Mcal command for robots with incremental encoders. The **Home** command simply moves the arm at low speed to a standby position defined by HomeSet.

Normally, for SCARA and Cartesian robots, the Z joint (J3) returns first to the HomeSet position, then the J1, J2 and J4 joints simultaneously return to their respective HomeSet coordinate positions. The Hordr instruction can change this order of the axes returning to their home positions.

Note

Home Status Output:

When the robot is in its Home position, the controller's system Home output is turned ON.

Potential Errors

Attempting to Home without HomeSet Values Defined

Attempting to **Home** the robot without setting the HomeSet values will result in an Error 143 being issued.

See Also

Hofs, HTest, HOrdr

Home Example

The Home instruction can be used in a program such as this:

```
Function InitRobot
   Reset
   If Motor = Off Then
      Motor On
   EndIf
   Home
Fend
```

Or it can be issued from the Monitor window like this:

```
> home
>
```

HomeSet Statement

> S

Specifies and displays the Home position.

Syntax

(1) HomeSet *j1Pulses*, *j2Pulses*, *j3Pulses*, *j4Pulses*, [*j5Pulses*], [*j6Pulses*]
(2) HomeSet

Parameters

j1Pulses	The home position encoder pulse value for joint 1.
j2Pulses	The home position encoder pulse value for joint 2.
j3Pulses	The home position encoder pulse value for joint 3.
j4Pulses	The home position encoder pulse value for joint 4.
j5Pulses	Optional for 6-axis robots. The home position encoder pulse value for joint 5.
j6Pulses	Optional for 6-axis robots. The home position encoder pulse value for joint 6.

Return Values

Displays the pulse values defined for the current Home position when parameters are omitted.

Description

Allows the user to define a new home (standby) position by specifying the encoder pulse values for each of the robot joints.

Notes

Home Command Does Not Calibrate the Robot:

This note pertains to incremental encoder robots only.

While the **HomeSet** command sets the Home position encoder values, it is very important to remember that the Home command does not calibrate the robot. The Mcal command is used to calibrate the robot. (When main power is first turned on the robot must be calibrated using Mcal.)

Verinit and its Effect on HomeSet Values:

Executing Verinit deletes the current **HomeSet** values.

Potential Errors

Attempting to Home without HomeSet Values Defined:

Attempting to Home the robot without setting the **HomeSet** values will result in an Error 143 being issued.

Attempting to Display HomeSet Values without HomeSet Values Defined:

Attempting to display home position pulse values without HomeSet values defined causes an Error 143.

See Also

Home, Hordr, Mcal, Pls

HomeSet Example

The following examples are done from the monitor window:

```
> home
!!Error 143 : Home position is not defined

> homeset
!!Error 143 : Home position is not defined

> homeset 0,0,0,0  'Set Home position at 0,0,0,0
> homeset
  0  0
  0  0

> home  'Robot homes to 0,0,0,0 position
```

Using the Pls function, specify the current position of the arm as the Home position.

```
> homeset Pls(1), Pls(2), Pls(3), Pls(4)
```

HomeSet Function

F

Returns pulse values of the home position for the specified joint.

Syntax

HomeSet(*jointNumber)*

Parameters

jointNumber Integer expression representing the joint number to retrieve the HomeSet value for.

Return Values

Returns pulse value of joint home position. When *jointNumber* is 0, returns 1 when HomeSet has been set or 0 if not.

See Also

HomeSet Statement

HomeSet Function Example

This example uses the **HomeSet** function in a program:

```
Function DisplayHomeset

   Integer i

   If HomeSet(0) = FALSE Then
      Print "HomeSet is not defined"
   Else
      Print "HomeSet values:"
      For i = 1 To 4
         Print "J", i, " = ", HomeSet(i)
      Next i
   EndIf
Fend
```

Hordr Statement

> S

Specifies or displays the order of the axes returning to their Home positions.

Syntax

(1) Hordr *step1, step2, step3, step4*, [*step5*] ,[*step6*]
(2) Hordr

Parameters

step1	Bit pattern that defines which joints should home during the 1st step of the homing process.
step2	Bit pattern that defines which joints should home during the 2nd step of the homing process.
step3	Bit pattern that defines which joints should home during the 3rd step of the homing process.
step4	Bit pattern that defines which joints should home during the 4th step of the homing process.
step5	For 6 axis robots. Bit pattern that defines which joints should home during the 5th step of the homing process.
step6	For 6 axis robots. Bit pattern that defines which joints should home during the 6th step of the homing process.

Return Values

Displays current Home Order settings when parameters are omitted.

Description

Hordr specifies joint motion order for the Home command. (i.e. Defines which joint will home 1st, which joint will home 2nd, 3rd, etc.)

The purpose of the **Hordr** instruction is to allow the user to change the homing order. The homing order is broken into 4 or 6 separate steps, depending on robot type. The user then uses **Hordr** to define the specific joints which will move to the Home position during each step. It is important to realize that more than one joint can be defined to move to the Home position during a single step. This means that all joints can potentially be homed at the same time. For 4 axis robots, it is recommended that the Z joint normally be defined to move to the Home position first (in Step 1) and then allow the other joints to follow in subsequent steps. (See notes below)

The **Hordr** instruction expects that a bit pattern be defined for each of the steps. Each joint is assigned a specific bit. When the bit is set to 1 for a specific step, then the corresponding joint will home. When the bit is cleared to 0, then the corresponding axis will not home during that step. The joint bit patterns are assigned as follows:

Joint:	**1**	**2**	**3**	**4**	**5**	**6**
Bit Number:	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5
Binary Code:	&B0001	&B0010	&B0100	&B1000	&B10000	&B100000

Notes

Difference Between MCordr and Hordr

While at first glance the MCordr and **Hordr** commands may appear very similar there is one major difference which is important to understand. MCordr is used to define the Robot Calibration Joint Order (used with Mcal) while **Hordr** is used to define the Homing joint order (used with the Home command).

Verinit Instruction (Resetting the Default Hordr)

The **Hordr** values are initialized to the above values by executing the Verinit instruction.

Hordr values are maintained

The **Hordr** values are permanently saved and are not changed until either the user changes them or a Verinit instruction is issued.

See Also

Mcal, MCordr, Home, HomeSet

Hordr Statement Example

Following are some monitor window examples for a 4 axis robot:

This example defines the home order as J3 in the first step, J1 in second step, J2 in third step, and J4 in the fourth step. The order is specified with binary values.

```
>hordr  &B0100, &B0001, &B0010, &B1000
```

This example defines the home order as J3 in the first step, then J1, J2 and J4 joints simultaneously in the second step. The order is specified with decimal values.

```
>hordr  4, 11, 0, 0
```

This example displays the current home order in decimal numbers.

```
>hordr
4, 11, 0, 0
>
```

Hordr Function

> F

Returns Hordr value for a specified step.

Syntax

Hordr(*stepNumber*)

Parameters

stepNumber Integer expression representing which Hordr step to retrieve.

Return Values

Integer containing the Hordr value for the specified step.

See Also

Mcal, MCordr, Home, HomeSet

Hordr Function Example

```
Integer a
a = Hordr(1)
```

Hour Statement

>

Displays the accumulated controller operating time.

Syntax

Hour

Description

Displays the amount of time the PC has been turned on and running SPEL. (Accumulated Operating Time) Time is always displayed in units of hours.

See Also

Time

Hour Example

The following example is done from the Monitor window:

```
> hour
2560
>
```

Hour Function

F

Returns the accumulated controller operating time.

Syntax

Hour

Return Values

Returns accumulated operating time of the controller (real number, in hours).

See Also

Time

Hour Function Example

```
Print "Number of controller operating hours: ", Hour
```

HTest Statement

>

Displays the pulse count from the time SPEL+ detects that the home sensor is switched on until it detects the encoder Z phase signal.

Syntax

HTest

Return Values

Pulse count for each joint is displayed.

Description

Upon executing the Mcal instruction, the robot arm moves from the position where the calibration sensor is switched off (if the calibration sensor is switched on, the arm moves to switch the sensor off) to the position where the sensor is switched on, and continues to move until the encoder Z phase is detected within one rotation, then the arm is stopped. The **HTest** instruction displays the pulse count of each joint following this movement.

Before each robot shipment, the pulse count, which includes some errors of the home sensor detection, is adjusted for safety.

Potential Errors

HTest Value Exceeds Range:

When the **HTest** values are beyond the predetermined allowable range, an error will occur.

See Also

Hofs, Mcal

HTest Statement Example

Display current HTest values in the monitor window:

```
>htest
344, 386, 254, 211
>
```

HTest Function

F

Returns the HTEST value for the specified joint.

Syntax

HTest(*jointNumber*)

Parameters

jointNumber An integer expression representing the joint number for which to retrieve the HTest value.

Return Values

Returns the HTEST value of the specified joint (integer value, in pulses).

See Also

HTest Statement

HTest Function Example

This example uses the **HTest** function in a program:

```
Function DisplayHTest

   Integer i

   Print "HTest values:"
   For i = 1 To 4
      Print HTest(i)
   Next i
Fend
```

If…Then…Else…EndIf Statement S

Executes instructions based on a specified condition.

Syntax

(1) **If** *condition* **Then**
stmtT1
.
.
[ElseIf *condition* **Then]**
stmtT1
.
.
[Else]
stmtF1
.
.
EndIf

(2) **If** *condition* **Then** *stmtT1* **[;** *stmtT2***...]** **[Else** *stmtF1* **[;** *stmtF2***...]]**

Parameters

condition	Any valid test condition which returns a True (any number besides 0) or False result (returned as a 0). (See sample conditions below)
stmtT1	Executed when the condition is True. (Multiple statements may be put here in a blocked **If...Then...Else** style.)
stmtF1	Executed when the condition is False. (Multiple statements may be put here in a blocked **If...Then...Else** style.)

Description

(1) **If...Then...Else** executes stmtT1, etc. when the conditional statement is True. If the condition is False then stmtF1, etc. are executed. The **Else** portion of the **If...Then...Else** instruction is optional. If you omit the **Else** statement and the conditional statement is False, the statement following the EndIf statement will be executed. For blocked **If...Then...Else** statements the EndIf statement is required to close the block regardless of whether an **Else** is used or not.

Nesting of up to 256 levels are supported for the blocked **If...Then...Else** statement including other statements (Do…Loop, Select…Send, and While…Wend).

(2) **If...Then...Else** can also be used in a non blocked fashion. This allows all statements for the **If...Then...Else** to be put on the same line. Please note that when using **If...Then...Else** in a non blocked fashion, the EndIf statement is not required. If the If condition specified in this line is satisfied (True), the statements between the Then and Else are executed. If the condition is not satisfied (False), the statements following **Else** are executed. The **Else** section of the **If...Then...Else** is not required. If there is no **Else** keyword then control passes on to the next statement in the program if the **If** condition is False.

The logical output of the conditional statement is any number excluding 1 when it is True, and 0 when it is false.

Notes

Sample Conditions:

a = b	:a is equal to b
a < b	:b is larger than a
a >= b	:a is greater than or equal to b
a <> b	:a is not equal to b
a > b	:b is smaller than a
a <= b	:a is less than or equal to b

Logical operations And, Or and Xor may also be used.

See Also

Else, Select...Case, While...Wend

If/Then/Else Statement Example

<Single Line If...Then...Else>

The following example shows a simple function which checks an input to determine whether to turn a specific output on or off. This task could be a background I/O task which runs continuously.

```
Function main
    Do
       If Sw(0) = 1 Then On 1 Else Off 1
    Loop
Fend
```

<Blocked If...Then...Else>

The following example shows a simple function which checks a few inputs and prints the status of these inputs

```
If Sw(0) = 1 Then Print "Input0 ON" Else Print "Input0 OFF"
'
If Sw(1) = 1 Then
    If Sw(2) = 1 Then
       Print "Input1 On and Input2 ON"
    Else
       Print "Input1 On and Input2 OFF"
    EndIf
Else
    If Sw(2) = 1 Then
       Print "Input1 Off and Input2 ON"
    Else
       Print "Input1 Off and Input2 OFF"
    EndIf
EndIf
```

<Other Syntax Examples>

```
If x = 10 And y = 3 Then GoTo 50
If test <= 10 Then Print "Test Failed"
If Sw(0) = 1 Or Sw(1) = 1 Then Print "Everything OK"
```

ImportPoints Statement

> S

Imports a point file into the current project for the current robot.

Syntax

ImportPoints *sourcePath, projectFileName*

Parameters

sourcePath	String expression containing the specific path and file to import into the current project. The extension must be .PNT.
projectFileName	String expression containing the specific file to be imported to in the current project for the current robot The extension must be .PNT.

Description

ImportPoints copies a point file into the current project and adds it to the files for the current robot. The point file is then compiled and is ready for loading using the LoadPoints command. If the file already exists for the current robot, it will be overwritten and recompiled.

Potential Errors

File Does Not Exist

If *sourcePath* does not exist, an error 53 will be issued.

A Path Cannot be Specified

If *projectFileName* contains a path, an error will occur. Only a file name in the current project can be specified for *projectFileName*.

Point file for another robot.

If *projectFileName* is a point file for another robot, an error will occur

See Also

Dir, LoadPoints, Robot, SavePoints

ImportPoints Statement Example

```
Function main
  Robot 1
  ImportPoints "c:\mypoints\model1.pnt", "robot1.pnt"
  LoadPoints "robot1.pnt"
Fend
```

In Function

F

Returns the status of the specified input port. Each port contains 8 input channels.

Syntax

In(*portNumber*)

Parameters

portNumber Number between 0 - 63 representing one of the 64 I/O ports. Keep in mind that each port contains 8 I/O channels making a total of 512 input channels.

Return Values

Returns an integer value between 0-255. The return value is 8 bits, with each bit corresponding to 1 input channel.

Description

In provides the ability to look at the value of 8 input channels at the same time. The In instruction can be used to store the 8 I/O channels status into a variable or it can be used with the Wait instruction to Wait until a specific condition which involves more than 1 I/O channel is met.

Since 8 channels are checked at a time, the return values range from 0-255. Please review the chart below to see how the integer return values correspond to individual input channels.

The system supports 16 Inputs and 16 Outputs as standard for each drive unit. This means that Ports 0 and 1 of the In command are available for each system. If three drive units are used, then Ports 0 to 5 will be available. Ports 6 - 63 are available as the user installs additional I/O boards into the controller with 512 inputs and 512 outputs available.

Input Channel Result (Using Port #0)

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

Input Channel Result (Using Port #3)

Return Value	31	30	29	28	27	26	25	24
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

See Also

InBCD, MemIn, MemOff, MemOn, MemSw, Off, On, OpBCD, Oport, Out, Sw, Wait

In Function Example

For the example below lets assume that input channels 28,29,30 and 31 are all connected to sensory devices such that the application should not start until each of these devices are returning an On signal indicating everything is OK to start. The program example gets the current value of the last 8 inputs and then makes sure that channels 28, 29, 30, and 31 are each On before proceeding. If they are not On (i.e. returning a value of 1) an error message is given to the operator and the task is stopped.

In the program, the variable "var1" is compared against the number 239 because in order for inputs 28, 29, 30 and 31 to all be On, then the result of In(3) will be 240 or larger. (We don't care about Inputs 24, 25, 26, and 27 in this case so any values between 240-255 will allow the program to proceed.)

```
Function main
    Integer var1
    var1 = In(3)  'Get last 8 input bits
    If var1 > 239 Then
        Go P1
        Go P2
        'Execute other motion statements here
        '.
        '.
    Else
        Print "Error in initialization!"
        Print "Sensory Inputs not ready for cycle start"
        Print "Please check inputs 28,29,30 and 31 for"
        Print "proper state for cycle start and then"
        Print "start program again"
    EndIf
Fend
```

We cannot set inputs from the monitor window but we can check them. For the examples shown below, we will assume that the Input channels 1, 5, 15, and 30 are On. All other inputs are Off.

```
> print In(0)
34
> print In(1)
128
> print In(2)
0
> print In(3)
64
```

In($n) Function

F

Note: The In($) function is obsolete and has been replaced by the MemIn function from Epson RC+ 4.0.

Returns the status of the specified memory I/O port. Each port contains 8 memory bits.

Syntax

In($*portNumber*)

Parameters

portNumber Number between 0-63 representing one of the 64 memory I/O ports. Keep in mind that each port contains 8 memory I/O bits making a total of 512 memory I/O bits.

Note: The "$" must be put in front of the I/O number to indicate that this is memory I/O.

Return Values

Returns an integer value between 0 - 255. The return value is 8 bits, with each bit corresponding to 1 memory I/O bit.

Description

In $ provides the ability to look at the value of 8 memory I/O bits at the same time. The **In $** instruction can be used to store the 8 memory I/O bit status into a variable or it can be used with the Wait instruction to Wait until a specific condition which involves more than 1 I/O bit is met.

Since 8 bits are retrieved at a time, the return values range from 0-255. Please review the chart below to see how the integer return values correspond to individua memory I/O bits.

Memory I/O Bit Result (Using Port #0)

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

Memory I/O Bit Result (Using Port #31)

Return Value	255	254	253	252	251	250	249	248
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

See Also

In, INBCD, Off, Off $, On, On $, OpBCD, Oport, Out, Out $, Sw, Sw $, Wait

In $ Example

The program example below gets the current value of the first 8 memory I/O bits and then makes sure that all 8 bits are currently set to 0 before proceeding. If they are not 0 an error message is given to the operator and the task is stopped.

```
Function main
    Integer var1

    var1 = In($0)  'Get 1st 8 memory I/O bit values
    If var1 = 0 Then
        Go P1
        Go P2
    Else
        Print "Error in initialization!"
        Print "First 8 memory bits were not all set to 0"
    EndIf
Fend
```

Other simple examples from the Monitor window are as follows:

```
> out $0, 1
> print In($0)
1
> on $1
> print In($0)
3
> out $31,3
> print In($31)
3
> off $249
> print In($31)
1
>
```

InBCD Function

F

Returns the input status of 8 inputs using BCD format. (Binary Coded Decimal)

Syntax

InBCD(*portNumber***)**

Parameters

portNumber Integer number between 0-63 representing one of the 64 input groups (each group contains 8 inputs) which make up the 512 inputs available.

Return Values

Returns as a Binary Coded Decimal (0-9), the input status of the input port (0-99).

Description

InBCD simultaneously reads 8 input lines using the BCD format. The 512 user inputs are broken into 64 groups of 8. The *portNumber* parameter for the InBCD instruction defines which group of 8 inputs to read where *portNumber* = 0 means inputs 0-7, *portNumber* = 1 means inputs 8-15, etc.

The resulting value of the 8 inputs is returned in BCD format. The return value may have 1 or 2 digits between 0 and 99. The 1st digit (or 10's digit) corresponds to the upper 4 outputs of the group of 8 outputs selected by *portNumber*. The 2nd digit (or 1's digit) corresponds to the lower 4 outputs of the group of 8 outputs selected by *portNumber*.

Since valid entries in BCD format range from 0-9 for each digit, every I/O combination cannot be met. The table below shows some of the possible I/O combinations and their associated return values assuming that *portNumber* is 0.

Input Settings (Input number)

Return Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

Notice that the Binary Coded Decimal format only allows decimal values to be specified. This means that through using Binary Coded Decimal format it is impossible to retrieve a valid value if all inputs for a specific port are turned on at the same time when using the **InBCD** instruction. The largest value possible to be returned by **InBCD** is 99. In the table above it is easy to see that when 99 is the return value for InBCD, all inputs are not on. In the case of a return value of 99, inputs 0, 3, 4, and 7 are On and all the others are Off.

EPSON RC+ supports 16 Inputs and 16 Outputs as standard. This means that Ports 0 and 1 of the **InBCD** command are available with the standard system. Ports 2-63 become available as the user installs additional I/O boards into the controller.

Notes

Difference between InBCD and In

The **InBCD** and In instructions are very similar in the SPEL+ language. However, there is one major difference between the two. This difference is shown below:

- The **InBCD** instruction uses the Binary Coded Decimal format for specifying the return value format for the 8 inputs. Since Binary Coded Decimal format precludes the values of &HA, &HB, &HC, &HD, &HE or &HF from being used, all combinations for the 8 inputs cannot be satisfied.
- The In instruction works very similarly to the InBCD instruction except that In allows the return value for all 8 inputs to be used. (i.e. 0-255 vs. 0-99 for InBCD) This allows all possible combinations for the 8 bit input groups to be read.

See Also

In, MemOff, MemOn, MemOut, MemSw, Off, On, OpBCD, Oport, Out, Sw, Wait

InBCD Example

Some simple examples from the Monitor window are as follows:

Assume that inputs 0, 4, 10, 16, 17, and 18 are all On (The rest of the inputs are Off).

```
> Print InBCD(0)
11
> Print InBCD(1)
04
> Print InBCD(2)
07
>
```

Inertia Statement

>

Specifies total load inertia and eccentricity for current robot.

Syntax

Inertia [*loadInertia*], [*eccentricity*]
Inertia

Parameters

loadInertia Optional. Real expression that specifies total moment of inertia in kgm^2 around the center of the end effector joint, including end effector and part.

eccentricity Optional. Real expression that specifies eccentricity in mm around the center of the end effector joint, including end effector and part.

Return Values

When parameters are omitted, the current Inertia parameters are displayed.

Description

The **Inertia** statement is supported starting with E2 Series robots. If **Inertia** is used with older types of robots, errors will occur.

Use the **Inertia** statement to specify the total moment of inertia for the load on the end effector joint. This allows the system to more accurately compensate acceleration, deceleration, and servo gains for end effector joint. You can also specify the distance from the center of end effector joint to the center of gravity of the end effector and part using the *eccentricity* parameter.

See Also

Inertia Function

Inertia Statement Example

```
Inertia 0.02, 1
```

Inertia Function

F

Returns inertia parameter value.

Syntax

Inertia(*paramNumber***)**

Parameters

paramNumber	Integer expression which can have the following values:
	0: Causes function to return 1 if robot supports inertia parameters or 0 if not.
	1: Causes function to return load inertia in kgm2.
	2: Causes function to return eccentricity in mm.

Return Values

Real value of the specified setting.

See Also

Inertia Statement

Inertia Function Example

```
Real loadInertia, eccentricity

loadInertia = Inertia(1)
eccentricity = Inertia(2)
```

InPos Function

F

Returns the position status of the specified robot.

Syntax

InPos [(*robotNumber*)]

Parameters

robotNumber Optional. Specifies which robot to retrieve position status for. When omitted, the current robot is assumed.

Return Values

True if last motion command has been completed successfully, otherwise False.

See Also

CurPos, FindPos, WaitPos

InPos Function Example

```
Function main

  P0 = 0, -100, 0, 0
  P1 = 0, 100, 0, 0

  Xqt MonitorPosition, NoPause
  Do
    Jump P0
    Wait .5
    Jump P1
    Wait .5
  Loop

Fend

Function MonitorPosition

  Boolean oldInPos, pos

  Do
    Pos = InPos
    If pos <> oldInPos Then
      Print "InPos = ", pos
    EndIf
    oldInPos = pos
  Loop

Fend
```

Input Statement

> S

Allows numeric data to be received from the keyboard and stored in a variable(s).

Syntax

Input *varName* [*, varName, varName,...*]

Parameters

varName	Variable name. Multiple variables can be used with the **Input** command as long as they are separated by commas.

Description

Input receives numeric data from the controlling device and assigns the data to the variable(s) used with the **Input** instruction.

When executing the Input instruction, a (?) prompt appears at the controlling device. After inputting data press the return key (Enter) on the keyboard.

Notes

Rules for Numeric Input

When inputting numeric values and non-numeric data is found in the input other than the delimiter (comma), the **Input** instruction discards the non-numeric data and all data following that non-numeric data.

Rules for String Input

When inputting strings, numeric and alpha characters are permitted as data.

Other Rules for the Input Instruction

- When more than one variable is specified in the instruction, the numeric data input intended for each variable has to be separated by a comma (",") character.
- Numeric variable names and string variable names are allowed. However, the input data type must match the variable type.

Potential Errors

Number of variables and input data differ

For multiple variables, the number of input data must match the number of Input variable names. When the number of the variables specified in the instruction is different from the number of numeric data received from the keyboard, an Error 30 will occur.

See Also

Input #, Line Input, Line Input #, Print, String

Input Statement Example

This function shows some simple Input statement examples.

```
Function InputNumbers
    Integer A, B, C

    Print "Please enter 1 number"
    Input A
    Print "Please enter 2 numbers separated by a comma"
    Input B, C
    Print "A = ", A
    Print "B = ", B, "C = ", C
Fend

A sample session of the above program running is shown below:
(Use the Run menu or F5 key to start the program)

Please enter 1 number
?-10000
Please enter 2 numbers separated by a comma
?25.1, -99
-10000
25.1 -99
B = 25.1 C = -99
>
```

Input # Statement

> S

Allows string or numeric data to be received from a file or communications port and stored in a variable(s).

Syntax

Input #*handle, varName* [*, varName, varName,...*]

Parameters

handle	The file or communications handle specified in an open statement. File handles can be specified in ROpen, WOpen, and AOpen statements. Communication handles can be specified in OpenCom (RS232) and OpenNet (TCP/IP) statements.
varName	Variable name to receive the data.

Description

The **Input #** instruction receives numeric or string data from the device specified by *handle*, and assigns the data to the variable(s).

Notes

Rules for Numeric Input

When inputting numeric values and non-numeric data is found in the input other than the delimiter (comma), the **Input** instruction discards the non-numeric data and all data following that non-numeric data.

Rules for String Input

When inputting strings, numeric and alpha characters are permitted as data.

Other Rules for the Input Instruction

- When more than one variable is specified in the instruction, the numeric data input intended for each variable has to be separated by a comma (",") character.
- Numeric variable names and string variable names are allowed. However, the input data type must match the variable type.

Potential Errors

Number of variables and input data differ

When the number of the variables specified in the instruction is different from the number of numeric data received from the device, an Error 30 will occur.

See Also

Input, Line Input, Line Input #

Input # Statement Example

This function shows some simple Input # statement examples.

```
Function GetData
    Integer A
    String B$

    OpenCom #1
    Print #1, "Send"
    Input #1, A   'Accept numeric data from port #1
    Input #1, B$  'Get string data from port #1
    CloseCom #1
Fend
```

InputBox Statement

> S

Displays a prompt in a dialog box, waits for the user to input text or choose a button, and returns the contents of the box.

Syntax

InputBox *prompt, title, default, answer$*

Parameters

prompt	String expression displayed as a message in the dialog box.
title	String expression displayed in the title bar of the dialog box.
default	String expression displayed in the text box as the default response. If no default is desired, use an empty string ("").
answer$	A string variable which will contain what the user typed. If the user entered nothing or clicks Cancel, this string will be "@".

Description

InputBox displays the dialog and waits for the user to click OK or Cancel. *answer* is a string that contains what the user typed in.

See Also

MsgBox

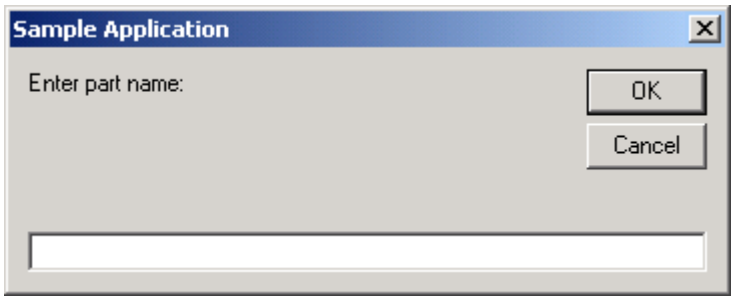
InputBox Statement Example

This function shows an InputBox example.

```
Function GetPartName$ As String
    String prompt$, title$, answer$

    prompt$ = "Enter part name:"
    title$ = "Sample Application"
    InputBox prompt$, title$, "", answer$
    If answer$ <> "@" Then
        GetPartName$ = answer$
    EndIf
Fend
```

The following picture shows the example output from the InputBox example code shown above.



InStr Function

F

Returns position of one string within another.

Syntax

InStr(*string, searchString*)

Parameters

string	String expression to be searched.
searchString	String expression to be searched for within *string*.

Return Values

Returns the position of the search string if the location is found, otherwise -1.

See Also

Mid$

Instr Function Example

```
Integer pos

pos = InStr("abc", "b")
```

Int Function

F

Converts a Real number to Integer. Returns the largest integer that is less than or equal to the specified value.

Syntax

Int*(number)*

Parameters

number A real number expression.

Return Values

Returns an Integer value of the real number used in *number*.

Description

Int(*number*) takes the value of *number* and returns the largest integer that is less than or equal to *number*.

Note

For Values Less than 1 (Negative Numbers)

If the parameter *number* has a value of less than 1 then the return value have a larger absolute value than *number*. (For example, if number = -1.35 then -2 will be returned.)

See Also

Abs, Atan, Atan2, Cos, Mod, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Int Function Example

Some simple examples from the Monitor window are as follows:

```
> Print Int(5.1)
5
> Print Int(0.2)
0
> Print Int(-5.1)
-6
>
```

Integer Statement

S

Declares variables of type Integer. (2 byte whole number).

Syntax

Integer *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameters

varName	Variable name which the user wants to declare as type integer.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

Integer is used to declare variables as type integer. Variables of type integer can contain whole numbers with values from -32768 to 32767. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

See Also

Boolean, Byte, Double, Global, Long, Real, String

Integer Statement Example

The following example shows a simple program that declares some variables using **Integer**.

```
Function inttest
    Integer A(10)         'Single dimension array of integer
    Integer B(10, 10)     'Two dimension array of integer
    Integer C(10, 10, 10) 'Three dimension array of integer
    Integer var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For I = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next I
Fend
```

InW Function

F

Returns the status of the specified input word port. Each word port contains 16 input bits.

Syntax

InW(*number*)

Parameters

number Specifies the word port numbers within the installed I/O board range using an integer expression.

Return Values

Returns the current status of inputs (long integers from 0 to 65535).

See Also

In, Out, OutW

InW Function Example

```
Long word0

word0 = InW(0)
```

IONumber Function

F

Returns the I/O number of the specified standard or memory I/O label.

Syntax

IONumber(*IOlabel*)

Parameters

IOlabel String expression that specifies the standard I/O or memory I/O label.

Return Values

Returns the bit number of the specified standard or memory I/O label. If there is no such I/O label, an error will be generated.

See Also

ENetIO_IONumber, FbusIO_IONumber

IONumber Function Example

```
Integer IObit

IObit = IONumber("myIO")

IObit = IONUmber("Station" + Str$(station) + "InCycle")
```

J4Flag Statement

> S

Sets the J4Flag attribute of a point.

Syntax

(1) J4Flag *point*, [*value*]
(2) J4Flag

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Optional. Integer expression. 0 (/J4F0) J4 range is -180 to +180 degrees 1 (/J4F1) J4 range is from -360 to -180 or +180 to +360 degrees

Return Values

The J4Flag attribute specifies the range of values for joint 4 for one point. If *value* is ommited, the J4Flag value for the specified point is displayed. When both parameters are omitted, the J4Flag value is displayed for the current robot position.

See Also

Elbow, Hand, J4Flag Function, J6Flag, Wrist

J4Flag Statement Example

```
J4Flag P0, 1
J4Flag P(mypoint), 0
```

J4Flag Function

F

Returns the J4Flag attribute of a point.

Syntax

J4Flag [(*point*)]

Parameters

point Optional. Point expression. If *point* is ommitted, then the J4Flag setting of the current robot position is returned.

Return Values

0 /J4F0
1 /J4F1

See Also

Elbow, Hand, Wrist, J4Flag Statement, J6Flag

J4Flag Function Example

```
Print J4Flag(pick)
Print J4Flag(P1)
Print J4Flag
Print J4Flag(Pallet(1, 1))
```

J6Flag Statement

> S

Sets the J6Flag attribute of a point.

Syntax

(1) J6Flag *point*, [*value*]
(2) J6Flag

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Integer expression. Range is 0 - 127 (/J6F0 - /J6F127). J6 range for the specified point is as follows: (-180 * (*value*+1) < J6 <= 180 * *value*) and (180 * *value* < J6 <= 180 * (*value*+1))

Return Values

The J6Flag attribute specifies the range of values for joint 6 for one point. If *value* is ommited, the J6Flag value for the specified point is displayed. When both parameters are omitted, the J6Flag value is displayed for the current robot position.

See Also

Elbow, Hand, J4Flag, J6Flag Function, Wrist

J6Flag Statement Example

```
J6Flag P0, 1
J6Flag P(mypoint), 0
```

J6Flag Function

F

Returns the J6Flag attribute of a point.

Syntax

J6Flag [(*point*)]

Parameters

point Optional. Point expression. If *point* is ommitted, then the J6Flag setting of the current robot position is returned.

Return Values

0 - 127 /J6F0 - /J6F127

See Also

Elbow, Hand, Wrist, J4Flag, J6Flag Statement

J6Flag Function Example

```
Print J6Flag(pick)
Print J6Flag(P1)
Print J6Flag
Print J6Flag(P1 + P2)
```

JA Function

F

Returns a robot point specified in joint angles.

Syntax

JA(*j1*, *j2*, *j3*, *j4*, [*j5*], [*j6*]**)**

Parameters

j1 - j6 Real expressions representing joint angles.

Return Values

A robot point whose location is determined by the specified joint angles.

Description

Use JA to specify a robot point using joint angles.

When the points returned from JA function specify a singularity of the robot, the joint angles of the robot do not always agree with the joint angles supplied to the JA function as arguments during the execution of a motion command for the points. To operate the robot using the joint angles specified for the JA function, avoid a singularity of the robot.

For example:

```
> go ja(0,0,0,90,0,-90)
> where
WORLD:  X:    0.000 mm  Y:  655.000 mm  Z:  675.000 mm  U:    0.000 deg V:  -90.000 deg W:  -90.000 deg
JOINT:  1:    0.000 deg 2:    0.000 deg 3:    0.000 deg 4:    0.000 deg 5:    0.000 deg 6:    0.000 deg
PULSE:  1:        0 pls 2:        0 pls 3:        0 pls 4:        0 pls 5:        0 pls 6:        0 pls

> go ja(0,0,0,90,0.001,-90)
> where
WORLD:  X:   -0.004 mm  Y:  655.000 mm  Z:  675.000 mm  U:    0.000 deg V:  -90.000 deg W:  -89.999 deg
JOINT:  1:    0.000 deg 2:    0.000 deg 3:    0.000 deg 4:   90.000 deg 5:    0.001 deg 6:  -90.000 deg
PULSE:  1:        0 pls 2:        0 pls 3:        0 pls 4:  2621440 pls 5:       29 pls 6: -1638400 pls
```

See Also

AglToPls, XY

JA Function Example

```
P10 = JA(60, 30, -50, 45)
Go JA(135, 90, -50, 90)
P3 = JA(0, 0, 0, 0, 0, 0)
```

JRange Statement

> S

Defines the permissible working range of the specified joint in pulses.

Syntax

JRange *jointNumber, lowerLimit, upperLmit*

Parameters

jointNumber	Integer expression between 1-6 representing the joint for which JRange will be specified.
lowerLmit	Long integer expression representing the encoder pulse count position for the lower limit range of the specified joint.
upperLmit	Long Integer expression representing the encoder pulse count position for the upper limit range of the specified joint.

Description

Defines the permissible working range for the specified joint with upper and lower limits in encoder pulse counts. **JRange** is similar to the Range command. However, the Range command requires that all joint range limits be set while the **JRange** command can be used to set each joint working limits individually thus reducing the number of parameters required. To confirm the defined working range, use the Range command.

Notes

Lower Limits Must Not Exceed Upper Limits:

The Lower limit defined in the JRange command must not exceed the Upper limit. A lower limit in excess of the Upper limit will cause an error, making it impossible to execute a motion command.

Factors Which can Change JRange:

Once **JRange** values are set they remain in place until the user modifies the values either by the Range or **JRange** commands. Neither turning main power off nor executing the Verinit command changes the **JRange** joint limit values.

Maximum and Minimum Working Ranges:

Refer to the specifications in the Robot manual for maximum working ranges for each robot model since these vary from model to model.

See Also

Range, JRange Function

JRange Statement Example

The following examples are done from the Monitor window:

```
> JRange 2, -6000, 7000       'Define the 2nd joint range

> JRange 1, 0, 7000           'Define the 1st joint range
```

JRange Function

F

Returns the permissible working range of the specified joint in pulses.

Syntax

JRange(*jointNumber, paramNumber*)

Parameters

jointNumber	Specifies reference joint number (integer from 1 - 6) by an expression or numeric value.
paramNumber	Integer expression containing one of two values: 1: Specifies lower limit value. 2: Specifies upper limit value.

Return Values

Range setting (integer value, pulses) of the specified joint.

See Also

Range, JRange Statement

JRange Function Example

```
Long i, oldRanges(3, 1)

For i = 0 To 3
    oldRanges(i, 0) = JRange(i + 1, 1)
    oldRanges(i, 1) = JRange(i + 1, 2)
Next i
```

JS Function

F

Jump Sense detects whether the arm stopped prior to completing a Jump, Jump3, or Jump3CP instruction which used a Sense input or if the arm completed the move.

Syntax

JS

Return Values

Returns a 1 or a 0.

1: When the arm was stopped prior to reaching its target destination because a Sense Input condition was met **JS** returns a 1.

0: When the arm completes the normal move and reaches the target destination as defined in the Jump instruction **JS** returns a 0.

Description

JS is used in conjunction with the Jump and Sense instructions. The purpose of the **JS** instruction is to provide a status result as to whether an input condition (as defined by the Sense instruction) is met during motion caused by the Jump instruction or not. When the input condition is met, **JS** returns a 1. When the input condition is not met and the arm reaches the target position, **JS** returns a 0.

JS is simply a status check instruction and does not cause motion or specify which Input to check during motion. The Jump instruction is used to initiate motion and the Sense instruction is used to specify which Input (if any) to check during Jump initiated motion.

Note

JS Works only with the Most Recent Jump, Jump3, Jump3CP Instruction:

JS can only be used to check the most recent Jump instruction's input check (which is initiated by the Sense instruction.) Once a 2nd Jump instruction is initiated, the **JS** instruction can only return the status for the 2nd Jump instruction. The **JS** status for the first Jump is gone forever. So be sure to always do any **JS** status check for Jump instructions immediately following the Jump instruction to be checked.

See Also

JT, Jump, Jump3, Jump3CP, Sense

JS Function Example

```
Function SearchSensor As Boolean
    Sense Sw(5) = On

    Jump P0
    Jump P1 Sense
    If JS = 1 Then
        Print "Sensor was found"
        SearchSensor = TRUE
    EndIf
Fend
```

JT Function

F

Returns the status of the most recent Jump, Jump3, or Jump3CP instruction for the current robot.

Syntax

JT

Return Values

JT returns a long with the following bits set or clear:

Bit 0	Set to 1 when rising motion has started or rising distance is 0.
Bit 1	Set to 1 when horizontal motion has started or horizontal distance is 0.
Bit 2	Set to 1 when descent motion has started or descent distance is 0.
Bit 16	Set to 1 when rising motion has completed or rising distance is 0.
Bit 17	Set to 1 when horizontal motion has completed or horizontal distance is 0.
Bit 18	Set to 1 when descent motion has completed or descent distance is 0.

Description

Use JT to determine the status of the most recent Jump command that was stopped before completion by Sense, Till, abort, etc.

See Also

JS, Jump, Jump3, Jump3CP, Sense, Till

JT Function Example

```
Function SearchTill As Boolean

    Till Sw(5) = On

    Jump P0
    Jump P1 Till
    If JT And 4 Then
        Print "Motion stopped during descent"
        SearchTill = TRUE
    EndIf
Fend
```

JTran Statement

S

Perform a relative move of one joint.

Syntax

JTran *jointNumber, distance*

Parameters

jointNumber	Integer expression representing which joint to move.
distance	Real expression representing the distance to move in degrees for rotational joints or millimeters for linear joints.

Description

Use **JTran** to move one joint a specified distance from the current position.

See Also

Go, Jump, Move, PTran

JTran Statement Example

```
JTran 1, 20
```

Jump Statement

> S

Moves the arm from the current position to the specified destination point using point to point motion by first moving in a vertical direction up, then horizontally and then finally vertically downward to arrive on the final destination point.

Syntax

Jump *destination* [**C***archNumber*] [**LimZ** *zLimit*] [CP] [*searchExpr*] [*!...!*]

Parameters

destination	The target destination of the motion using a point expression.
archNumber	Optional. The arch number (*archNumber*) specifies which Arch Table entry to use for the Arch type motion caused by the Jump instruction. *archNumber* must always be proceeded by the letter C. (Valid entries are C0-C7.)
zLimit	Optional. This is a Z limit value which represents the maximum position the Z joint will travel to during the **Jump** motion. This can be thought of as the Z Height Ceiling for the **Jump** instruction. Any valid Z joint Coordinate value is acceptable.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Sense, Till or Find expression. **Sense** | **Till** | **Find** **Sense Sw**(*expr*) = {**On** | **Off**} **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to the Jump instruction to cause I/O and other commands to execute during motion.

Description

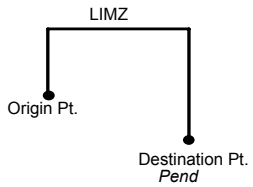
Jump moves the arm from the current position to *destination* using what is called Arch Motion. Jump can be thought of as 3 motions in 1. For example, when the Arch table entry defined by *archNumber* is 7, the following 3 motions will occur.

1) The move begins with only Z-joint motion until it reaches the Z joint height calculated by the Arch number used for the Jump command.
2) Next the arm moves horizontally (while still moving upward in Z) towards the target point position until the upper Z Limit (defined by LimZ) is reached. Then the arm begins to move downward in the Z direction (while continuing X, Y and U joint motion) until the final X, and Y and U joint positions are reached.
3) The **Jump** instruction is then completed by moving the arm down with only Z-joint motion until the target Z-joint position is reached.

The coordinates of *destination* (the target position for the move) must be taught previously before executing the **Jump** instruction. The coordinates cannot be specified in the **Jump** instruction itself. Acceleration and deceleration for the **Jump** is controlled by the Accel instruction. Speed for the move is controlled by the Speed instruction.

***archNumber* Details**

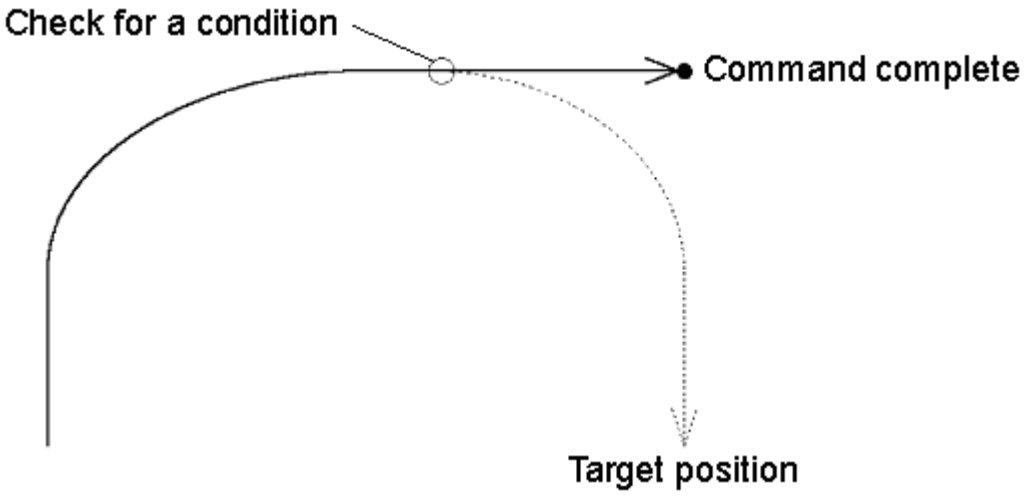
The Arch for the **Jump** instruction can be modified based on the *archNumber* value optionally specified with the **Jump** instruction. This allows the user to define how much Z to move before beginning the X, Y, and U joint motion. (This allows the user to move the arm up and out of the way of parts, feeders and other objects before beginning horizontal motion.) Valid *archNumber* entries for the **Jump** instruction are between C0-C7. The Arch table entries for C0-C6 are user definable with the Arch instruction. However, C7 is a special Arch entry which always defines what is called Gate Motion. Gate Motion means that the robot first moves Z all the way to the coordinate defined by LimZ before beginning any X, Y, or U joint motion. Once the LimZ Z limit is reached, X, Y and U joint motion begins. After the X, Y, and U joints each reaches its final destination position, then the Z joint can begin moving downward towards the final Z joint coordinate position as defined by *destination* (the target point). Gate Motion looks as follows:



LimZ Details
LimZ *zLimit* specifies the upper Z coordinate value for the horizontal movement plane in the current local coordinate system. The specified arch settings can cause the X, Y, and U joints to begin movement before reaching LimZ, but LimZ is always the maximum Z height for the move. When the LimZ optional parameter is omitted, the previous value specified by the LimZ instruction is used for the horizontal movement plane definition.

It is important to note that the LimZ *zLimit* height limit specification is the Z value for the local robot coordinate system. It is not the Z value for Arm or Tool. Therefore take the necessary precautions when using tools or hands with different operating heights.

Sense Details
The Sense optional parameter allows the user to check for an input condition or memory I/O condition before beginning the final Z motion downward. If satisfied, this command completes with the robot stopped above the target position where only Z motion is required to reach the target position. It is important to note that the robot arm does not stop immediately upon sensing the Sense input modifier.



The Js or Stat commands can then be used to verify whether the Sense condition was satisfied and the robot stopped prior to its target position or that the Sense condition was not satisfied and the robot continued until stopping at its target position.

Till Details
The optional Till qualifier allows the user to specify a condition to cause the robot to decelerate to a stop prior to completing the **Jump**. The condition specified is simply a check against one of the 512 inputs or one of the memory I/O. This is accomplished through using either the Sw or MemSw function. The user can check if the input is On or Off and cause the arm to decelerate and stop based on the condition specified.

The Stat function can be used to verify whether the Till condition has been satisfied and this command has been completed, or the Till condition has not been satisfied and the robot stopped at the target position.

Notes

Jump cannot be executed for 6-axis robots

Use Jump3 or Jump3CP for 6-axis robots.

Jump Motion trajectory changes depending on motion and speed

Jump motion trajectory is comprised of vertical motion and horizontal motion. It is not a continuous path trajectory. The actual Jump trajectory of arch motion is not determined by **Arch** parameters alone. It also depends on motion and speed.

Always use care when optimizing Jump trajectory in your applications. Execute Jump with the desired motion and speed to verify the actual trajectory.

When speed is lower, the trajectory will be lower. If Jump is executed with high speed to verify an arch motion trajectory, the end effector may crash into an obstacle with lower speed.

In a Jump trajectory, the depart distance increases and the approach distance decreases when the motion speed is set high. When the fall distance of the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the fall distance to be larger.

Even if Jump commands with the same distance and speed are executed, the trajectory is affected by motion of the robot arms. As a general example, for a SCARA robot the vertical upward distance increases and the vertical downward distance decreases when the movement of the first arm is large. When the vertical fall distance decreases and the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the fall distance to be larger.

Omitting *archNumber* Parameter

If the archnum optional parameter is omitted, the default Arch entry for use with the Jump instruction is C7. This will cause Gate Motion, as described above.

Difference between Jump and Jump3, Jump3CP

The Jump3 and Jump3CP instructions can be used for 6-axis robots. On the other hand the Jump instruction cannot be used for 6-axis robots. For SCARA and CARTESIAN robots, using the Jump instruction shortens the joint motion time for depart and approach motion. The depart and approach motions in Jump3 can be executed along the Z axis and in other directions.

Difference between Jump and Go

The Go instruction is similar to **Jump** in that they both cause Point to Point type motion, however there are many differences. The most important difference is that the Go instruction simply causes Point to Point motion where all joints start and stop at the same time (they are synchronized). Jump is different since it causes vertical Z movement at the beginning and end of the move. Jump is ideal for pick and place type applications.

Decelerating to a Stop With the Jump Instruction

The Jump instruction always causes the arm to decelerate to a stop prior to reaching the destination point.

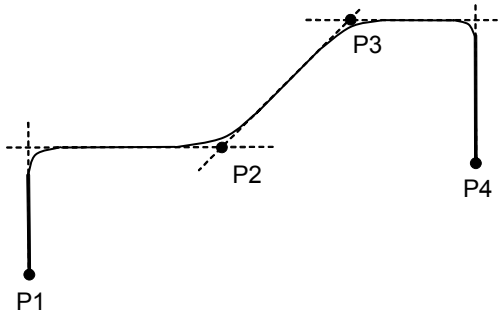
Proper Speed and Acceleration Instructions with Jump:

The Speed and Accel instructions are used to specify the speed and acceleration of the robot during **Jump** motion. Pay close attention to the fact that Speed and Accel apply to point to point type motion (Go, Jump, Etc.). while linear and circular interpolated motion instructions such as Move or Arc use the SpeedS and AccelS instructions. For the Jump instruction, it is possible to separately specify speeds and accelerations for Z joint upward motion, horizontal travel including U joint rotation, and Z joint downward motion.

Pass function of Jump (Ver.4.2 or after)

When CP parameter is specified to Jump with 0 downward motion, the Jump horizontal travel does not decelerate to a stop but goes on smoothly to the next PTP motion.
When CP parameter is specified to the PTP motion command right before Jump with 0 upward motion, the PTP motion does not decelerate to a stop but connects smoothly with the Jump horizontal travel.
This is useful when you want to replace the horizontal travel of Jump (a PTP motion) to several smooth PTP motion connection.
(Exampl)

```
Go P1
Jump P2 :Z(-50) LimZ -50 C0 CP
Go P3 :Z(0) CP
Jump P4 LimZ 0 C0
```



Potential Errors

LimZ Value Not High Enough

When the current arm position of the Z joint is higher than the value set for LimZ and a Jump instruction is attempted, an Error 146 will occur.

See Also

Accel, Arc, Arch, Go, Js, Jt, LimZ, Point Expression, Pulse, Sense, Speed, Stat, Till

Jump Statement Example

The example shown below shows a simple point to point move between points P0 and P1 and then moves back to P0 using the Jump instruction. Later in the program the arm moves using the Jump instruction until Input #4 goes high. If input #4 goes high then the arm stops moving. If input #4 never goes high then the arm completes the Jump and arrives on point P1.

```
Function jumptest
    Home
    Go P0
    Go P1
    Sense Sw(4) = 1
    Jump P0 LimZ -10
    Jump P1 LimZ-10 Sense   'Check input #4
    If Js(0) = 1 Then
        Print "Input #4 came on during the move and"
        Print "the robot stopped prior to arriving on"
        Print "point P1."
    Else
        Print "The move to P1 completed successfully."
        Print "Input #4 never came on during the move."
    EndIf
Fend

> Jump P10+X50 C0 LimZ-20 Sense !D50;On 0;D80;On 1!
```

Jump3, Jump3CP Statements

> S

3D gate motion. Jump3 is a combination of two CP motions and one PTP motion. Jump3CP is a combination of three CP motions.

Syntax

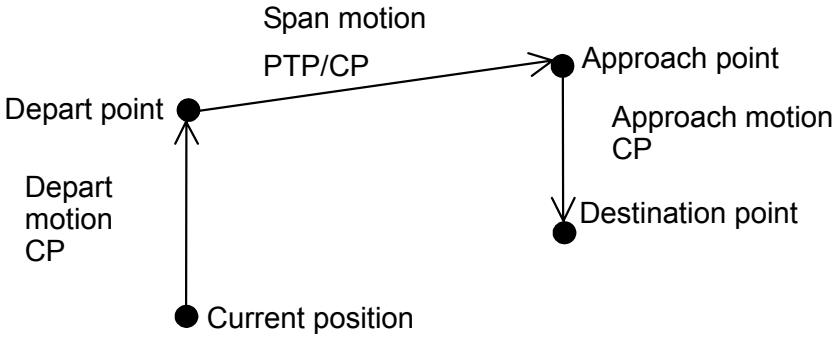
(1) Jump3 *depart, approach, destination* [**C***archNumber*] [**CP**] [*searchExpr*] [*!...!*]
(2) Jump3CP *depart, approach, destination* [**ROT**] [**C***archNumber*] [**CP**] [*searchExpr*] [*!...!*]

Parameters

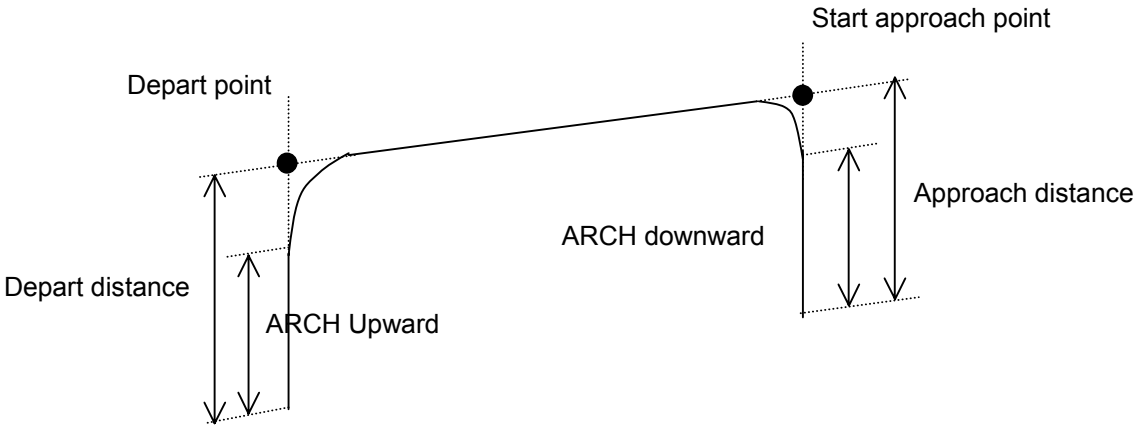
depart	The departure point above the current position using a point expression.
approach	The approach point above the destination position a point expression.
destination	The target destination of the motion using a point expression.
ROT	Optional. :Decides the speed/acceleration/deceleration in favor of tool rotation.
archNumber	Optional. The arch number (*archNumber*) specifies which Arch Table entry to use for the Arch type motion caused by the Jump instruction. *archNumber* must always be proceeded by the letter C. (Valid entries are C0-C7.)
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Sense, Till or Find expression. **Sense** | **Till** | **Find** **Sense Sw**(*expr*) = {**On** | **Off**} **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to the Jump instruction to cause I/O and other commands to execute during motion.

Description

Moves the arm from the current position to the destination point with 3D gate motion. 3D gate motion consists of depart motion, span motion, and approach motion. The depart motion form the current position to the depart point is always CP motion. The span motion from the depart point to the start approach point is PTP motion in **Jump3**, and the CP motion in **Jump3CP**.
The approach motion from the starting approach point to the target point is always CP motion.



Arch motion is achieved by specifying the arch number. The arch motion for Jump3, Jump3CP is as shown in the figure below. For arch motion to occur, the Depart distance must be greater than the arch upward distance and the Approach distance must be greater than the arch downward distance.



Jump3CP uses the SpeedS speed value and AccelS acceleration and deceleration values. Refer to *Using Jump3CP with CP* below on the relation between the speed/acceleration and the acceleration/deceleration. If, however, the ROT modifier parameter is used, **Jump3CP** uses the SpeedR speed value and AccelR acceleration and deceleration values. In this case SpeedS speed value and AccelS acceleration and deceleration value have no effect.

Usually, when the move distance is 0 and only the tool orientation is changed, an error will occur. However, by using the ROT parameter and giving priority to the acceleration and the deceleration of the tool rotation, it is possible to move without an error. When there is not an orientational change with the ROT modifier parameter and movement distance is not 0, an error will occur.

Also, when the tool rotation is large as compared to move distance, and when the rotation speed exceeds the specified speed of the manipulator, an error will occur. In this case, please reduce the speed or append the ROT modifier parameter to give priority to the rotational speed/acceleration/deceleration.

Notes

Jump3 and Jump3CP not supported for PG robots

Jump3 and **Jump3CP** cannot be used for robots that use the PG Motion System.

Jump3 span motion is PTP (point to point)

It is difficult to predict Jump3 span motion trajectory. Therefore, be careful that the robot doesn't collide with peripheral equipment and that robot arms don’t collide with the robot.

Jump3 Motion trajectory changes depending on motion and speed

Jump3 motion trajectory is comprised of depart, span, and approach motions. It is not a continuous path trajectory. The actual Jump3 trajectory of arch motion is not determined by **Arch** parameters alone. It also depends on motion and speed.

Always use care when optimizing Jump3 trajectory in your applications. Execute Jump3 with the desired motion and speed to verify the actual trajectory.

When speed is lower, the trajectory will be lower. If Jump3 is executed with high speed to verify an arch motion trajectory, the end effector may crash into an obstacle with lower speed.

In a Jump3 trajectory, the depart distance increases and the approach distance decreases when the motion speed is set high. When the approach distance of the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the approach distance to be larger.

Even if Jump commands with the same distance and speed are executed, the trajectory is affected by motion of the robot arms. As a general example, for a SCARA robot the depart distance increases and

the approach distance decreases when the movement of the first arm is large. When the approach distance decreases and the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the approach distance to be larger.

LimZ does not affect Jump3 and Jump3CP

LimZ has no affect on Jump3 or Jump3CP since the span motion is not necessarily perpendicular to the Z axis of the coordinate system.

Potential acceleration errors

An acceleration error may occur during an arch motion execution by the Jump3 andJump3CP commands. This error is issued frequently when the majority of the motion during depart or approach uses the same joint as the span motion. To avoid this error, reduce the acceleration/deceleration speed of the span motion using Accel command for Jump3 or using AccelS command for Jump3CP. Depending on the motion and orientation of the robot, it may also help to reduce the acceleration and deceleration of the depart motion (approach motion) using the AccelS command.

Using Jump3, Jump3CP with CP

The CP parameter causes the arm to move to *destination* without decelerating or stopping at the point defined by *destination*. This is done to allow the user to string a series of motion instructions together to cause the arm to move along a continuous path while maintaining a specified speed throughout all the motion. The **Jump3** and **Jump3CP** instructions without CP always cause the arm to decelerate to a stop prior to reaching the point *desination*.

Pass function of Jump3 (Ver.4.2 or after)

When CP parameter is specified to Jump3 with 0 approach motion, the Jump3 span motion does not decelerate to a stop but goes on smoothly to the next PTP motion.
When CP parameter is specified to the PTP motion command right before Jump3 with 0 depart motion, the PTP motion does not decelerate to a stop but connects smoothly with the Jump3 span motion.
This is useful when you want to replace the span motion of Jump3 (a PTP motion) to several smooth PTP motion connection.

Pass function of Jump3CP (Ver.4.2 or after)

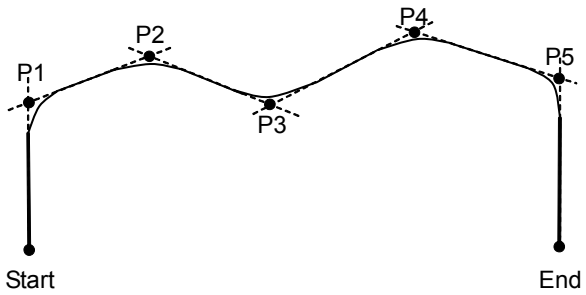
When CP parameter is specified to Jump3CP with 0 approach motion, the Jump3CP span motion does not decelerate to a stop but goes on smoothly to the next CP motion.
When CP parameter is specified to the CP motion command right before Jump3CP with 0 depart motion, the CP motion does not decelerate to a stop but connects smoothly with the Jump3CP span motion.
This is useful when you want to replace the span motion of Jump3CP (a CP motion) to several smooth CP motion connection.

(Example 1)

```
Jump3 P1,P2,P2 CP
Go P3,P4 CP
Jump3 P4,P5,P5+tlz(50)
```

(Example 2)

```
Jump3CP P1,P2,P2 CP
Move P3,P4 CP
Jump3CP P4,P5,P5+tlz(50)
```



See Also

Accel, Arc, Arch, Go, Js, Jt, Point Expression, Pulse, Sense, Speed, Stat, Till

Jump3 Statement Example

```
' 6 axis robot motion which works like Jump of SCARA robot
Jump3  P* :Z(100), P3 :Z(100), P3

' Depart and approach use Z tool coordinates
Jump3  P* -TLZ(100), P3 -TLZ(100), P3

' Depart uses base Z and approach uses tool Z
Jump3  P* +Z(100), P3 -TLZ(100), P3

' Example for the depart motion from P1 in Tool 1 and the approach
motion to P3 in Tool 2

Arch 0,20,20
Tool 1
Go P1

P2 = P1 -TLZ(100)
Tool 2
Jump3 P2, P3-TLZ(100), P3 C0
```

Kill Statement

>

Deletes files on a PC disk drive.

Syntax

Kill *pathName*

Parameters

pathName String expression containing filename to delete. *pathName* may include the drive and directory.

See Also

Del, Dir, SavePoints

Kill Statement Example

```
Function DeleteFile(path$ As String)
  Integer answer
  String msg$

  msg$ = "OK to delete file: '" + path$ + "'?"
  MsgBox msg$, MB_YESNO + MB_ICONQUESTION, "MyApp", answer
  If answer = IDYES Then
    Kill path$
  EndIf
Fend
```

LCase$ Function

F

Returns a string that has been converted to lowercase.

Syntax

LCase$(*string*)

Parameters

string A valid string expression.

Return Values

The converted lower case string.

See Also

LTrim$, Trim$, RTrim$, UCase$

LCase$ Function Example

```
str$ = "Data"
str$ = LCase$(str$)   ' str$ = "data"
```

Left$ Function

F

Returns a substring from the left side of a string expression.

Syntax

Left$*(string, count)*

Parameters

string	String expression from which the leftmost characters are copied.
count	The number of characters to copy from *string* starting with the leftmost character.

Return Values

Returns a string of the leftmost *number* characters from the character string specified by the user.

Description

Left$ returns the leftmost *number* characters of a string specified by the user. **Left$** can return up to as many characters as are in the character string.

See Also

Asc, Chr$, InStr, Len, Mid$, Right$, Space$, Str$, Val

Left$ Function Example

The example shown below shows a program which takes a part data string as its input and parses out the part number, part name, and part count.

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(datain$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(datain$, Len(DataIn$) - pos))

Fend
```

Some other example results from the Left$ instruction from the Monitor window.

```
> Print Left$("ABCDEFG", 2)
 AB

> Print Left$("ABC", 3)
 ABC
```

Len Function

F

Returns the number of characters in a character string.

Syntax

Len*(string)*

Parameters

string	String expression.

Return Values

Returns an integer number representing the number of characters in the string *string* which was given as an argument to the **Len** instruction.

Description

Len returns an integer number representing the number of characters in a string specified by the user. **Len** will return values between 0-256 (since a string can contain between 0-256 characters).

See Also

Asc, Chr$, InStr, Left$, Mid$, Right$, Space$, Str$, Val

Len Function Example

The example shown below shows a program which takes a part data string as its input and parses out the part number, part name, and part count.

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(datain$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(datain$, Len(DataIn$) - pos))

Fend
```

Some other example results from the **Len** instruction from the monitor window.

```
> ? len("ABCDEFG")
7

> ? len("ABC")
3

> ? len("")
0
>
```

LimZ Statement

> S

Determines the default value of the Z joint height for Jump commands.

Syntax

(1) LimZ *zLimit*
(2) LimZ

Parameters

zLimit	A coordinate value within the movable range of the Z joint.

Return Values

Displays the current LimZ value when parameter is omitted.

Description

LimZ determines the maximum Z joint height which the arm move to when using the Jump instruction, wherein the robot arm raises on the Z joint, moves in the X-Y plane, then lowers on the Z joint. **LimZ** is simply a default Z joint value used to define the Z joint ceiling position for use during motion caused by the Jump instruction. When a specific **LimZ** value is not specified in the Jump instruction, the last **LimZ** setting is used for the Jump instruction.

Note

Resetting LimZ to 0

Restarting the software, or executing the SFree, SLock, Motor On, and Verinit instructions all initialize LimZ to 0.

LimZ Value is Not Valid for Arm, Tool, or Local Coordinates:

LimZ Z joint height limit specification is the Z joint value for the robot coordinate system. It is not the Z joint value for Arm, Tool, or Local coordinates. Therefore take the necessary precautions when using tools or end effectors with different operating heights.

LimZ does not affect Jump3 and Jump3CP

LimZ has no affect on Jump3 or Jump3CP since the span motion is not necessarily perpendicular to the Z axis of the coordinate system.

See Also

Jump

LimZ Statement Example

The example below shows the use of LimZ in Jump operations.

```
Function main
    LimZ -10            'Set the default LimZ value
    Jump P1             'Move up to Z=-10 position for Jump
    Jump P2 LimZ -20    'Move up to Z=-20 position for Jump
    Jump P3          'Move up to Z=-10 position for Jump
Fend
```

LimZ Function

F

Returns the current LimZ setting.

Syntax

LimZ

Return Values

Real number containing the current LimZ setting.

See Also

LimZ Statement

LimZ Function Example

```
Real savLimz

savLimz = LimZ
LimZ -25
Go pick
LimZ savLimZ
```

Line Input Statement

> S

Reads input data of one line and assigns the data to a string variable.

Syntax

Line Input *stringVar$*

Parameters

stringVar$	A string variable name. (Remember that the string variable must end with the $ character.)

Description

Line Input reads input data of one line from the controlling device and assigns the data to the string variable used in the Line Input instruction. When the Line Input instruction is ready to receive data from the user, it causes a "?" prompt to be displayed on the controlling device. The input data line after the prompt is then received as the value for the string variable. After inputting the line of data press the [ENTER] key.

See Also

Input, Input #, Line Input#, ParseStr

Line Input Example

The example below shows the use of **Line Input**.

```
Function Main
    String A$
    Line Input A$  'Read one line input data into A$
    Print A$
Fend

Run the program above using the F5 key or Run menu from EPSON RC+ main
screen. A resulting run session may be as follows:

?A, B, C
A, B, C
```

Line Input # Statement

> S

Reads data of one line from the specified file or communication port.

Syntax

Line Input #*portNumber, stringVar$*

Parameters

portNumber	Integer expression representing a file number or communication port number.
stringVar$	A string variable. (Remember that string variables must end with a $ character.)

Description

Line Input # reads string data of one line from the file or communication port specified with the *portNumber* parameter, and assigns the data to the string variable *stringVar$.*

See Also

Input, Input #, Line Input

Line Input # Example

This example receives the string data from the communication port number 1, and assigns the data to the string variable A$.

```
Function lintest
    String a$
    Print #1, "Please input string to be sent to robot"
    Line Input #1, a$
    Print "Value entered = ", a$
Fend
```

LoadPoints Statement

> S

Loads a point file into the point memory area for the current robot.

Syntax

LoadPoints *fileName* **[, Merge]**

Parameters

fileName	String expression containing the specific file to load into the current robot's point memory area. The extension must be .PNT. The file must exist in the current project for the current robot. No path can be specified.
Merge	Optional. If supplied, then the current points are not cleared before loading the new points. Points in the file are added the to the current points. If a point exists in the file, it will overwrite the point in memory.

Description

LoadPoints loads point files from disk into the main memory area of the controller.

Use **Merge** to combine point files. For example, you could have one main point file that includes common points for locals, parking, etc in the range 0 - 100. Then use **Merge** to load other point files for each part being run without clearing the common points. The range could be 101 - 999.

Potential Errors

A Path Cannot be Specified

If *fileName* contains a path, an error will occur. Only a file name in the current project can be specified.

File Does Not Exist

If *fileName* does not exist, an error will be issued.

Point file not for the current robot.

If *fileName* is not a point file for the current robot, the following error will be issued: Point file not found for current robot. To correct this, add the Point file to the robot in the Project editor, or execute SavePoints or ImportPoints.

See Also

Dir, ImportPoints, Robot, SavePoints

LoadPoints Statement Example

```
Function main

  ' Load common points for the current robot
  LoadPoints "R1Common.pnt"

  ' Merge points for part model 1
  LoadPoints "R1Model1.pnt", Merge

  Robot 2
  ' Load point file for the robot 2
  LoadPoints "R2Model1.pnt"

Fend
```

Local Statement

> S

Defines and displays local coordinate systems.

Syntax

(1) Local *localNumber,* (*pLocal1 : pBase1)*, *(pLocal2 : pBase2),* [{ **L** | **R** }], [**BaseU**]
(2) Local *localNumber, pOrigin*
(3) Local *localNumber, pOrigin,* [*pXaxis*], [*pYaxis*], [{ **X** | **Y** }]

Parameters

localNumber	The local coordinate system number. A total of 15 local coordinate systems (of the integer value from 1 to 15) may be defined.
pLocal1, pLocal2	Point variables with point data in the local coordinate system.
pBase1, pBase2	Point variables with point data in the base coordinate system.
L | **R**	Optional. Align local origin to left (first) or right (second) base points.
BaseU	Optional. When supplied, U axis coordinates are in the base coordinate system. When omitted, U axis coordinates are in the local coordinate system.
pOrigin	Origin of 3D local.
pXaxis	Optional. A point on the X axis of 3D local if X alignment is specified.
pYaxis	Optional. A point along the Y axis of 3D local if Y alignment is specified.
X | **Y**	Optional. If X alignment is specified, then *pXaxis* is on the X axis of the local and only the Z coordinate of *pYaxis* is used. If Y alignment is specified, then *pYaxis* is on the Y axis of the local and only the Z coordinate of *pXaxis* is used. If ommitted, X alignment is assumed.

Description

(1) Local defines a local coordinate system by specifying 2 points, *pLocal1* and *pLocal2*, contained in it that coincide with two points, *pBase1* and *pBase2*, contained in the base coordinate system.

Example:

```
Local 1, (P1:P11), (P2:P12)
```

P1 and P2 are local coordinate system points. P11 and P12 are base coordinate system points.

If the distance between the two specified points in the local coordinate system is not equal to that between the two specified points in the base coordinate system, the XY plane of the local coordinate system is defined in the position where the midpoint between the two specified points in the local coordinate system coincides with that between the two specified points in the base coordinate system.

Similarly, the Z axis of the local coordinate system is defined in the position where the midpoints coincide with each other.

(2) Defines a local coordinate system by specifying the origin and axis rotation angles with respect to the base coordinate system.

Example:

```
Local 1, XY(x, y, z, u)
Local 1, XY(x, y, z, u, v, w)
Local 1, P1
```

(3) Defines a 3D local coordinate system by specifying the origin point, x axis point, and y axis point. Only the X, Y, and Z coordinates of each point are used. The U, V, and W coordinates are ignored. When the X alignment parameter is used, then *pXaxis* is on the X axis of the local and only the Z coordinate of *pYaxis* is used. When the Y alignment parameter is used, then *pYaxis* is on the Y axis of the local and only the Z coordinate of *pXaxis* is used.

Example:

```
Local 1, P1, P2, P3
Local 1, P1, P2, P3, X
Local 1, P1, P2, P3, Y
```

Using L and R parameters

While Local basically uses midpoints for positioning the axes of your local coordinate system as described above, you can optionally specify left or right local by using the **L** and **R** parameters.

Left Local

Left local defines a local coordinate system by specifying point *pLocal1* corresponding to point *pBase1* in the base coordinate system (Z axis direction is included.)

Right Local

Right local defines a local coordinate system by specifying point *pLocal2* corresponding to point *pBase2* in the base coordinate system. (Z axis direction is included.)

Using the BaseU parameter

If the **BaseU** parameter is omitted, then the U axis of the local coordinate system is automatically corrected in accordance with the X and Y coordinate values of the specified 4 points. Therefore, the 2 points in the base coordinate system may initially have any U coordinate values.

It may be desired to correct the U axis of the local coordinate system based on the U coordinate values of the two points in the base coordinate system, rather than having it automatically corrected (e.g. correct the rotation axis through teaching). To do so, supply the **BaseU** parameter.

Local Definitions are Lost Under the Following Conditions

Definitions of local coordinate systems 1 through 15 will be lost when Verinit is executed.

See Also

ArmSet, Base, ECPSet, LocalClr, TLSet, Where

Local Examples

Here are some examples from the monitor window:

Left aligned local:

```
> p1 = 0, 0, 0, 0/1
> p2 = 100, 0, 0, 0/1
> p11 = 150, 150, 0, 0
> p12 = 300, 150, 0, 0
> local 1, (P1:P11), (P2:P12), L
> p21 = 50, 0, 0, 0/1
> go p21
```

Local defined with only the origin point:

```
> local 1, 100, 200, -20
```

Local defined with only the origin point rotated 45 degrees about the X axis:

```
> local 2, 50, 200, 0, 0, 45
```

3D Local with p2 aligned with the X axis of the local:

```
> local 3, p1, p2, p3, x
```

3D Local with p3 aligned with the Y axis of the local:

```
> local 4, p1, p2, p3, y
```

Local Function

F

Returns the local number of a point.

Syntax

Local(*localNumber*)

Parameters

localNumber	local coordinate system number (integer from 0 - 15) using an expression or numeric value.

Return Values

Specified local coordinate system data as point data.

See Also

Local Statement

Local Function Example

```
P1 = Local(1)
```

LocalClr Statement

> S

Clears (undefines) a local coordinate system.

Syntax

LocalClr *localNumber*

Parameters

localNumber	Integer expression representing which of 15 locals to clear (undefine). (Local 0 is the default local and cannot be cleared.)

See Also

Arm, ArmSet, ECPSet, Tool, TLClr, TLSet, Verinit

LocalClr Example

```
LocalClr 1
```

Lof Function

F

Checks whether the specified RS-232 or TCP/IP port has any lines of data.

Syntax

Lof(*portNumber*)

Parameters

portNumber The communication port number.

Return Values

The number of lines of data in the buffer. If there is no data in the buffer, **Lof** returns 0.

Description

Lof checks whether or not the specified port has received data lines. The data received is stored in the buffer irrespective of the Input# instruction.

See Also

ChkCom, ChkNet, Input#

Lof Function Example

This Monitor window example prints out the number of lines of data received through the communication port number 1.

```
>print lof(1)
 5
>
```

LogIn Statement

S

Log into EPSON RC+ as another user.

Syntax

LogIn *logID, password*

Parameters

logID	String expression that contains user login id.
password	String expression that contains user password.

Description

You can utilize EPSON RC+ security in your application. For example, you can display a menu that allows different users to log into the system. Each type of user can have its own security rights. For more details on security, see the EPSON RC+ User's Guide.

When you are running programs in the development environment, the user before programs are started will be restored after programs stop running.

When running the Operator Window using the /OPR or /OPRAS command line options, the application is logged in as a guest user, unless Auto LogIn is enabled, in which case the application is logged in as the current Windows user if such user has been configured in the EPSON RC+ system.

Note

This command will only work if the Security option is installed.

See Also

GetCurrentUser$ Function

LogIn Statement Example

```
LogIn "operator", "oprpass"
```

Long Statement

S

Declares variables of type long integer. (4 byte whole number).

Syntax

Long *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]**...**]

Parameters

varName	Variable name which the user wants to declare as type **Long**.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

Long is used to declare variables as type **Long**. Variables of type **Long** can contain whole numbers with values between -2,147,483,648 to 2,147,483,647. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

See Also

Boolean, Byte, Double, Global, Integer, Real, String

Long Statement Example

The following example shows a simple program which declares some variables as Longs using **Long**.

```
Function longtest
    Long A(10)          'Single dimension array of long
    Long B(10, 10)      'Two dimension array of long
    Long C(10, 10, 10) 'Three dimension array of long
    Long var1, arrayVar(10)
    Long i
    Print "Please enter a Long Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For I = 1 To 5
        Print "Please enter a Long Number"
        Input arrayVar(i)
        Print "Value Entered was ", arrayVar(i)
    Next I
Fend
```

LPrint Statement

> S

Sends ASCII text to the PC printer.

Syntax

(1) LPrint *expression* [, *expression...*]
(2) LPrint

Parameters

expression Optional. A numeric or string expression.

Description

Use **LPrint** to print hardcopy of data on the default PC printer. After executing one or more **LPrint** statements, you must execute EPrint to send the data to the printer.

An end of line CRLF (carriage return and line feed) is automatically appended to each output.

See Also

EPrint, Print

LPrint Statement Example

```
Function dataPrint
    Integer i

    For i = 1 to 10
        LPrint g_testDat$(i)
    Next i
    EPrint  ' Send the data to the PC printer
Fend
```

LSet$ Function

F

Returns the specified string with trailing spaces appended up to the specified length..

Syntax

LSet$ (*string, length*)

Parameters

string	String expression.
length	Integer expression for the total length of the string returned.

Return Values

Specified string with trailing spaces appended.

See Also

RSet$, Space$

LSet$ Function Example

```
str$ = "123"
str$ = LSet$(str$, 10)   ' str$ = "123       "
```

LShift Function

F

Shifts numeric data to the left by a user specified number of bits.

Syntax

LShift*(number, shiftBits)*

Parameters

number	Integer expression to be shifted.
shiftBits	The number of bits to shift *number* to the left.

Return Values

Returns a numeric result which is equal to the value of *number* after shifting left *shiftBits* number of bits.

Description

LShift shifts the specified numeric data (*number*) to the left (toward a higher order digit) by the specified number of bits (*shiftBits*). The low order bits shifted are replaced by 0.

The simplest explanation for LShift is that it simply returns the result of *number* * $2^{shiftBits}$.

Note

Numeric Data Type:

The numeric data *number* may be any valid numeric data type. **LShift** works with data types: Byte, Integer, Long, and Real.

See Also

And, Not, Or, RShift, Xor

LShift Function Example

```
Function lshiftst
    Integer i
    Integer num, snum
    num = 1
    For i = 1 to 10
         Print "i =", i
         snum = LShift(num, i)
         Print "The shifted num is ", snum
    Next i
Fend
```

Some other example results from the LShift instruction from the monitor window.

```
> Print LShift(2,2)
8
> Print LShift(5,1)
10
> Print LShift(3,2)
12
>
```

LTrim$ Function

F

Returns a string equal to specified string without leading spaces.

Syntax

LTrim$ (*string*)

Parameters

string String expression.

Return Values

Specified string with leading spaces removed.

See Also

RTrim$, Trim$

LTrim$ Function Example

```
str$ = "  data  "
str$ = LTrim$(str$)  ' str$ = "data  "
```

Mask Operator

S

Bitwise mask for Wait statement condition expression.

Syntax

Wait *expr1* **Mask** *exrp2*

Parameters

expr1 Any valid expression input condition for Wait.

expr2 Any valid expression which returns a numeric result.

Description

The **Mask** operator is a bitwise And for Wait statement input condition expressions.

See Also

Wait

Mask Operator Example

```
' Wait for the lower 3 bits of input port 0 to equal 1
Wait In(0) Mask 7 = 1
```

MCal Statement

INC > S

Executes machine calibration for robots with incremental encoders.

Syntax

MCal

Description

It is necessary to calibrate robots which have incremental encoders. This calibration must be executed after turning on the main power. If you attempt motion command execution, or any command which requires the current position data without first executing machine calibration, an error will occur.

Machine calibration is executed according to the moving joint order which is specified with the MCordr command. The default value of MCordr at the time of shipment differs from model to model, so please refer to the proper robot manual for details.

Normally, MCal is executed when the robot is in the center of the motion envelope. On SCARA robots, which have multiple calibration points, the 1st and 2nd joint combined movement is limited during Mcal to just +/-15-16 degrees in the clockwise/counter clockwise directions. This helps reduce the time required for calibrating the robot. If Mcal is executed near the limit of the motion envelope, the arm may try to move out of range. In this case an error may be issued or the arm will hit the mechanical stop. To Mcal after this occurs simply turn the MOTORs Off and manually move the arm away from the motion envelope limit.

The actual moving distance at machine calibration differs from model to model. Please refer to the corresponding robot manual for details.

Attempt to Execution a Motion Instruction without Executing Mcal First

If you attempt motion command execution, or any command which requires the current position data (e.g. Plist* instruction) without first executing machine calibration, an error will occur.

Absolute encoder robots

If you attempt to execute MCAL for absolute encoder robots, error 409 will occur, which means that the command is not supported for this type of robot. Absolute encoder robots do not need MCAL.

Robot Installation Note

Z Joint Space Required for Homing

When the Z joint homes it first moves up and then moves down and settles into the home position. This means it is very important to properly install the robot so that enough space is provided for the arm to home the Z joint. It is recommended that a space of 6 mm be provided above the upper limit. (Do not install tooling or fixtures within a 6 mm space above the robot so enough room is left for proper Z joint homing.)

See Also

Hofs, HTest, Home, Hordr, Mcorg, MCOrdr, MCofs

Mcal Example

The following example is done from the monitor window:

```
> Motor On
> Mcal
>
```

MCalComplete Function

F

Returns status of MCal.

Syntax

MCalComplete

Return Values

True if MCal has been completed, otherwise False.

See Also

MCal

MCalComplete Example

```
If Not MCalComplete Then
    MCal
EndIf
```

MCofs Statement

INC >

Specifies and displays the calibration offset parameters for machine calibration. Required only for incremental encoder robots.

Syntax

MCofs *sensorLogic, j1Pulses, j2Pulses, j3Pulses, j4Pulses, j1WDiff, j2WDiff*

Parameters

sensorLogic	Integer expression representing the sensor logic.
j1Pulses	The pulse values between where the sensor comes on to where the Z phase is detected at the specified position of 1st joint.
j2Pulses	The pulse values between where the sensor comes on to where the Z phase is detected at the specified position of 2nd joint.
j3Pulses	The pulse values between where the sensor comes on to where the Z phase is detected at the specified position of 3rd joint.
j4Pulses	The pulse values between where the sensor comes on to where the Z phase is detected at the specified position of 4th joint.
j1WDiff	The difference between the logical value and sensor edge width at the specified position of the 1st joint.
j2WDiff	The difference between the logical value and sensor edge width at the specified position of the 2nd joint.

Return Values

Displays current MCofs values when parameters are omitted.

Description

Specifies the parameters for machine calibration. This parameter is necessary data for executing machine calibration with Mcal. If you specify all specification values, they are set as parameters for machine calibration.

Parameters for calibration are automatically calculated by moving the robot arm with Mcorg. It is recommended that you use Mcorg to create Mcofs values. However, if those values are known in advance, you can input the values using this MCofs command.

Notes

IMPORTANT: MCofs Should only be Used for Maintenance Purposes

When machine calibration is executed with Mcal, the current position is recognized based on this parameter, therefore, these data must be specified. If wrong data is specified, the robot coordinates are incorrect, and the wrong data causes improper motion.

Result of Turning Main Power Off or Verinit

MCofs values are maintained when power is turned off and even after Verinit is executed.

Absolute encoder robots

If you attempt to execute MCOFS for absolute encoder robots, error 409 will occur, which means that the command is not supported for this type of robot. Absolute encoder robots do not need MCOFS.

See Also

Mcal, Mcorg, Ver

MCofs Statement Example

The following examples are done from the monitor window:

```
> MCofs
04
1364  50
4506  6304
-479  -469

> MCofs &H04, 1364, 50, 4506, 6304, -479, -469
```

MCofs Function

F

Returns an MCofs parameter value.

Syntax

MCofs(*paramNumber*)

Parameters

paramNumber	Specifies MCOFS setting numbers (integer from 0 to 7) using an expression or numeric value.

Return Values

Returns the specified parameter for machine calibration.

See Also

Mcal, Mcorg, Ver

MCofs Function Example

```
Integer a

a = MCofs(1)
```

MCordr Statement

INC > S

Specifies and displays the moving joint order for machine calibration Mcal. Required only for robots with incremental encoders.

Syntax

(1) MCordr *Step1, Step2, Step3, Step4*
(2) MCordr

Parameters

Step1	Bit pattern that tells which axes should be calibrated during the 1st step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 1st step. (see below for bit pattern definitions)
Step2	Bit pattern that tells which axes should be calibrated during the 2nd step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 2nd step. (see below for bit pattern definitions)
Step3	Bit pattern that tells which axes should be calibrated during the 3rd step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 3rd step. (see below for bit pattern definitions)
Step4	Bit pattern that tells which axes should be calibrated during the 4th step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 4th step. (see below for bit pattern definitions)

Return Values

Displays current Machine Calibration Order when parameters are omitted.

Description

After the system is powered on, or after Verinit instruction, Mcal instruction must be issued prior to any robot arm operation. When the Mcal instruction is issued each of the 4 axes of the robot will move to their respective calibration positions.

Specifies joint motion order for the Mcal command. (i.e. Defines which joint will home 1st, which joint will Mcal 2nd, 3rd, etc.)

The purpose of the **MCordr** instruction is to allow the user to change the homing order. The homing order is broken into 4 separate steps. The user then uses **MCordr** to define the specific axes which will move to the calibration position (done with the Mcal command) during each step. It is important to realize that more than 1 joint can be defined to move to the calibration position during a single step. This means that all four axes can potentially be calibrated at the same time. However, it is recommended that the Z joint normally be defined to move to the calibration position first (in Step 1) and then allow the other Axes to follow in subsequent steps. (See notes below)

The **MCordr** instruction expects that a bit pattern be defined for each of the 4 steps. Since there are 4 axes, each joint is assigned a specific bit. When the bit is high (1) (for a specific step), then the corresponding joint will calibrate. When the bit is low (0), then the corresponding joint will not calibrate during that step. The joint bit patterns are assigned as follows:

Joint:	**1**	**2**	**3**	**4**
Bit Number:	bit 0	bit 1	bit 2	bit 3
Binary Code:	&B0001	&B0010	&B0100	&B1000

Notes

Difference Between MCordr and Hordr

While at first glance the Hordr and **MCordr** commands may appear very similar there is one major difference which is important to understand. **MCordr** is used to define the Robot Calibration joint order (used with Mcal) while Hordr is used to define the Homing joint order (used with the Home command).

Default MCal Order (Factory Setting)

The default joint calibration order from the factory is that joint 3 will home in Step 1. Then joints 1, 2, and 4 joints will all home at the same time in step 2. (Steps 3 and 4 are not used in the default configuration.) The default **MCordr** values are as follows:

MCordr &B0100, &B1011, 0, 0

Verinit Instruction (Resetting the Default Hordr)

The **MCordr** values are initialized to the above values by executing the Verinit instruction.

Z Joint should normally be calibrated first

The reason for moving the Z joint first (and by itself) is to allow the tooling to be moved above the work surface before beginning any horizontal movement. This will help prevent the tooling from hitting something in the work envelope during the homing process.

MCordr values are maintained

The **MCordr** Table values are permanently saved and are not changed until either the user changes them or a Verinit instruction is issued.

See Also

Mcal, MCofs

MCordr Statement Example

Following are some monitor window examples:

This example defines the calibration order as J3 in the first step, J1 in second step, J2 in third step, and J4 in the fourth step. The order is specified with binary values.

```
>mcordr  &B0100, &B0001, &B0010, &B1000
```

This example defines the calibration order as J3 in the first step, then J1, J2 and J4 joints simultaneously in the second step. The order is specified with decimal values.

```
>mcordr  4, 11, 0, 0
```

This example displays the current calibration order in decimal numbers.

```
>mcordr
4, 11, 0, 0
>
```

MCordr Function

F

Returns an MCordr parameter setting.

Syntax

MCordr(*paramNumber*)

Parameters

paramNumber Specifies reference setting numbers (integers from 1 to 4) by an expression or numeric value.

Return Values

Returns binary values (integers) representing the joint of the specified setting number to execute machine calibration.

Description

Returns the joint motion order to execute machine calibration by Mcal.

See Also

Mcal, MCofs

MCordr Function Example

This example uses the **MCordr** function in a program:

```
Integer a
a = MCordr(1)
```

Mcorg Statement

INC >

Calculates the proper calibration parameters for Machine Calibration (Mcal).

Syntax

Mcorg *j1,* [*j2*], [*j3*], [*j4*]

Parameters

j1-j4 Integer expression between 1-4 which represents the joint to perform the **Mcorg** on. At least one joint number must be specified for the **Mcorg** command to function properly without an error. However, multiple joints may have **Mcorg** done at the same time. (In the case where the robots have a ball screw spline, such as the SCARA robots, and the need to calculate a calibration parameter is for the 3rd or 4th joint, specify both joints together. If only one is specified an error occurs.

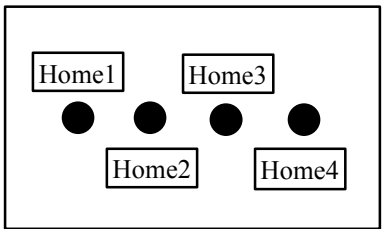
Description

Calculates the proper calibration parameters for Machine Calibration with Mcal. The steps required to properly use Mcorg with SCARA robots are shown below:

1) Make sure the Motors are off. (Execute Motor Off command)

2) Move the 3rd and 4th joints of the robot within the motion range.

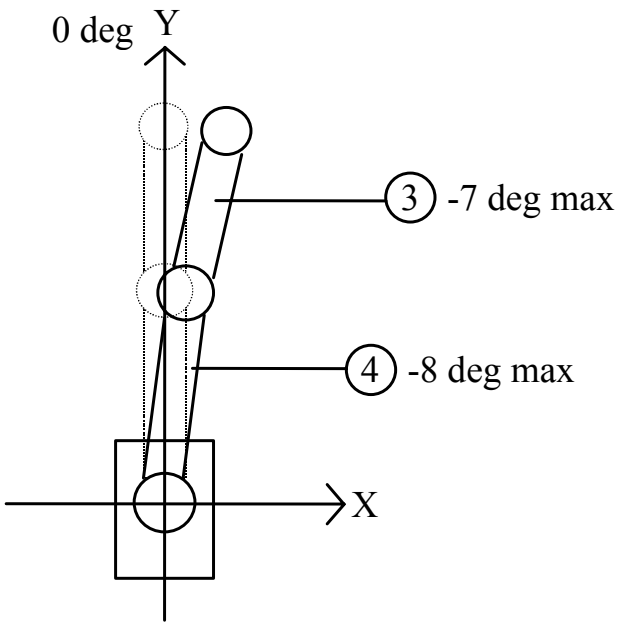
3) Move the arm into the correct **Mcorg** position by using the LED's on the sensor monitor which is located on the rear side of the robot base. LED's are shown below: The joint numbers are printed next to each LED to indicate which LED corresponds to which Calibration LED. If you move the 1st or 2nd joint, the LED corresponding to the joint turns on or off in accordance with the arm movement.



Stretch the arm as drawn with the dotted line figure below, so that the arm is parallel to the +Y axis of the robot coordinates. (i.e. pointing straight out from the base) In this position the HOME1 and HOME2 LED's turn on.

4) While watching the HOME2 LED, move the 2nd arm link in the clockwise direction by hand. While moving the 2nd arm link hold the 1st arm link so that it does not move. Stop moving the 2nd arm link at the approximate center position of the range where the HOME2 LED turns off for the first time.

5) Next while watching the HOME1 LED, move the 1st arm link in the clockwise direction by hand. Stop moving the 1st arm link at the approximate center position of the range where the HOME1 LED turns off for the first time. The approximate center position should be about 8 degrees from the Y axis centerline. The approximate center position of the 2nd arm link will be about 7 degrees from the center line position of the 1st link axis. See drawing below:



6) Turn on the motors by using the Motor On command.

7) When you execute **Mcorg** at this position the 1st and 2nd axes move in the counter clockwise direction. The maximum movement should be +15 for joint 2 and +18 for joint 1. Be sure to remove any obstacles in the range before executing **Mcorg**.

8) Execute Mcal to conduct machine calibration

9) Execute the Pulse command to move the arm to the 0 pulse position of the 1st and 2nd axes.
> Pulse 0, 0, 0, 0

10) After the arm moves to the Pulse position 0, 0, 0, 0 confirm that the 1st and 2nd arm links are straight and are on the X axis of the robot coordinate system. If they are not, it means the arm position when **Mcorg** was executed was not correct so go back to step 2 and try again.

Notes

If Mcorg is not Executed at the Correct Position:

In the case of the multiple calibration point robot, such as all TT8000 Series robots, if **Mcorg** is not executed at the right position, the parameters for calibration are not calculated correctly. If the parameters are wrong, the robot will not move properly.

If you execute **Mcorg** by mistake, do not execute any motion commands. In this situation, execute **Mcorg** from the correct position again, or find the original MCofs data as it was configured at shipment and input the data to using the MCofs command.

See Also

Mcal, MCofs

Mcorg Statement Example

The following example was done from the monitor window:

```
> mcorg 1,2   'Calculates parameters for machine calibration
              'for the 1st and 2nd axes
```

MemIn Function

F

Returns the status of the specified memory I/O port. Each port contains 8 memory bits.

Syntax

MemIn(*portNumber***)**

Parameters

portNumber Number between 0-63 representing one of the 64 memory I/O ports. Keep in mind that each port contains 8 memory I/O bits making a total of 512 memory I/O bits.

Return Values

Returns an integer value between 0-255. The return value is 8 bits, with each bit corresponding to 1 memory I/O bit.

Description

MemIn provides the ability to look at the value of 8 memory I/O bits at the same time. The **MemIn** instruction can be used to store the 8 memory I/O bit status into a variable or it can be used with the Wait instruction to Wait until a specific condition which involves more than 1 memory I/O bit is met.

Since 8 bits are retrieved at a time, the return value ranges from 0-255. Please review the chart below to see how the integer return values correspond to individual memory I/O bits.

Memory I/O Bit Result (Using Port #0)

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

Memory I/O Bit Result (Using Port #31)

Return Value	255	254	253	252	251	250	249	248
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

Notes

Difference Between MemIn and MemSw

The MemSw instruction allows the user to read the value of 1 memory I/O bit. The return value from MemSw is either a 1 or a 0 which indicates that the memory I/O bit is either On or Off. MemSw can check each of the 512 memory I/O bits individually. The **MemIn** instruction is very similar to the MemSw instruction in that it also is used to check the status of the memory I/O bits. However there is 1 distinct difference. The MemIn instruction checks 8 memory I/O bits at a time vs. the single bit checking functionality of the MemSw instruction. **MemIn** returns a value between 0-255 which tells the user which of the 8 I/O bits are On and which are Off.

See Also

In, InBCD, Off, MemOff, On, MemOn, OpBCD, Oport, Out, MemOut, Sw, MemSw, Wait

MemIn Example

The program example below gets the current value of the first 8 memory I/O bits and then makes sure that all 8 I/O are currently set to 0 before proceeding. If they are not 0 an error message is given to the operator and the task is stopped.

```
Function main
    Integer var1

    var1 = MemIn(0)  'Get 1st 8 memory I/O bit values
    If var1 = 0 Then
        Go P1
        Go P2
    Else
        Print "Error in initialization!"
        Print "First 8 memory I/O bits were not all set to 0"
    EndIf
Fend
```

Other simple examples from the Monitor window are as follows:

```
> memout 0, 1
> print MemIn(0)
1
> memon 1
> print MemIn(0)
3
> memout 31,3
> print MemIn(31)
3
> memoff 249
> print MemIn(31)
1
>
```

MemOff Statement

> S

Turns Off the specified bit of the memory I/O.

Syntax

MemOff { *bitNumber* | *memIOLabel* }

Parameters

bitNumber	Integer expression between 0-511 representing one of the 512 memory I/O bits.
memIOLabel	Memory I/O label.

Description

MemOff turns Off the specified bit of memory I/O. The 512 memory I/O bits are typically excellent choices for use as status bits for uses such as On/Off, True/False, Done/Not Done, etc. The MemOn instruction turns the memory bit On, the **MemOff** instruction turns it Off, and the MemSw instruction is used to check the current state of the specified memory bit. The Wait instruction can also be used with the memory I/O bit to cause the system to wait until a specified memory I/O status is set.

Note

Memory outputs off

All memory I/O bits are turned off when the SPEL drivers are restarted. They are not turned off by Emergency stop, safeguard open, program end, Reset command, or EPSON RC+ restart.

See Also

In, MemIn, InBCD, Off, On, MemOn, OpBCD, Oport, Out, MemOut, Sw, MemSw, Wait

MemOff Statement Example

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. MemSw is used in combination with the Wait instruction to wait until the memory I/O #1 is the proper value before it is safe to move again. MemOn and MemOff are used to turn on and turn off the memory I/O for proper synchronization.

```
Function main
    Integer I
    MemOff 1
    Xqt 2, task2
    For I = 1 to 100
        Wait Sw($1) = 0
        Go P(i)
        MemOn 1
    Next I
Fend


Function task2
    Integer I
    For I = 101 to 200
        Wait MemSw(1) = 1
        Go P(i)
        MemOff 1
    Next I
Fend
```

Other simple examples from the monitor window are as follows:

```
> MemOn 1       'Switch memory I/O bit #1 on
> Print MemSw(1)
1
> MemOff 1      'Switch memory I/O bit #1 off
> Print MemSw(1)
0
```

MemOn Statement

> S

Turns On the specified bit of the memory I/O.

Syntax

MemOn { *bitNumber* | *memIOLabel* }

Parameters

bitNumber	Integer expression between 0 - 511 representing one of the 512 memory I/O bits.
memIOLabel	Memory I/O label.

Description

MemOn turns on the specified bit of the robot memory I/O. The 512 memory I/O bits are typically used as task communication status bits. The MemOn instruction turns the memory bit On, the MemOff instruction turns it Off, and the MemSw instruction is used to check the current state of the specified memory bit. The Wait instruction can also be used with the memory bit to cause the system to wait until a specified status is set.

Note

Memory outputs off

All memory I/O bits are turned off when the SPEL drivers are restarted. They are not turned off by Emergency stop, safeguard open, program end, Reset command, or EPSON RC+ restart.

See Also

In, MemIn, InBCD, Off, MemOff, On, OpBCD, Oport, Out, MemOut, Sw, MemSw, Wait

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. MemSw is used in combination with the Wait instruction to wait until the memory I/O #1 is the proper value before it is safe to move again. MemOn and MemOff are used to turn on and turn off the memory I/O for proper synchronization.

```
Function main
    Integer I
    MemOff 1
    Xqt 2, task2
    For I = 1 to 100
        Wait MemSw(1) = Off
        Go P(i)
        MemOn 1
    Next I
Fend

Function task2
    Integer I
    For I = 101 to 200
        Wait MemSw(1) = On
        Go P(i)
        MemOff 1
    Next I
Fend
```

Other simple examples from the monitor window are as follows:

```
> memon 1
> print memsw(1)
1
> memoff 1
> print memsw(1)
0
```

MemOut Statement

> S

Simultaneously sets 8 memory I/O bits.

Syntax

MemOut *portNumber, outData*

Parameters

portNumber Integer expression between 0 - 63 representing one of the 64 output groups (each group contains 8 outputs) which make up the 512 memory I/O outputs. The *portNumber* selection corresponds to the following outputs:

Portnum	Outputs
0	0-7
1	8-15
.	.
63	496-511

outData Integer expression between 0-255 representing the output pattern for the output group selected by *portNumber*. If represented in hexadecimal form the range is from &H0 to &HFF. The lower digit represents the least significant digits (or the 1st 4 outputs) and the upper digit represents the most significant digits (or the 2nd 4 outputs).

Description

MemOut simultaneously sets 8 memory I/O bits using the combination of the *portNumber* and *outData* values specified by the user to determine which outputs will be set. The 512 memory outputs are broken into 64 groups of 8. The *portNumber* parameter specifies which group of 8 outputs to use where *portNumber* = 0 means outputs 0-7, *portNumber* = 1 means outputs 8-15, etc.

Once a *portNumber* is selected, a specific output pattern must be defined. This is done using the *outData* parameter. The *outData* parameter may have a value between 0-255 and may be represented in hexadecimal or integer format. (i.e. &H0-&HFF or 0-255)

The table below shows some of the possible I/O combinations and their associated *outData* values assuming that *portNumber* is 0, and 1 accordingly.

Output Settings When *portNumber*=0 (Output number)

OutData Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Output Settings When *portNumber*=1 (Output number)

OutData Value	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

See Also

In, MemIn, InBCD, MemOff, MemOn, MemSw, Off, On, OpBCD, Oport, Out, Sw, Wait

MemOut Example

The example below shows main task starting a background task called iotask. The iotask is a simple task to toggle memory I/O bits 0 - 3 On and Off. The MemOut instruction makes this possible using only 1 command rather than turning each memory I/O bit on and off individually.

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
Fend

Function iotask

  Do

    Wait 1
    MemOut 0, &H0
    Wait 1
  Loop
Fend
```

Other simple examples from the monitor window are as follows:

```
> MemOut 1,6      'Turns on memory I/O bits 9 & 10
> MemOut 2,1      'Turns on memory I/O bit 8
> MemOut 3,91     'Turns on memory I/O bits 24, 25, 27, 28, and 30
```

MemSw Function

F

Returns the status of the specified memory I/O bit.

Syntax

MemSw(*bitNumber***)**

Parameters

bitNumber Number between 0 - 511 representing one of the 512 memory I/O bits.

Return Values

Returns a 1 when the specified bit is On and a 0 when the specified bit is Off.

Description

MemSw returns the status of one memory I/O bit. Valid entries for MemSw range from bit 0 to bit 511. The MemOn and MemOff instructions are normally used with MemSw. MemOn turns the specified bit on and MemOff turns the specified bit Off.

See Also

In, MemIn, InBCD, MemOff, MemOn, MemOut, Off, On, OpBCD, Oport, Out, Sw, Wait

MemSw Example

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. MemSw is used in combination with the Wait instruction to wait until the memory I/O bit 1 is the proper value before it is safe to move again.

```
Function main
    Integer I
    MemOff 1
    Xqt 2, task2
    For I = 1 to 100
        Wait MemSw(1) = Off
        Go P(i)
        MemOn 1
    Next I
Fend

Function task2
    Integer I
    For I = 101 to 200
        Wait MemSw(1) = On
        Go P(i)
        MemOff 1
    Next I
Fend
```

Other simple examples from the Monitor window are as follows:

```
> memon 1
> print memsw(1)
1
> memoff 1
> print memsw(1)
0
```

Mid$ Function

F

Returns a substring of a string starting from a specified position.

Syntax

Mid$*(string, position,* [*count*])

Parameters

string	Source string expression.
position	The starting position in the character string for copying *count* characters.
count	Optional. The number of characters to copy from *string* starting with the character defined by *position*. If ommitted, then all characters from *position* to the end of the string are returned.

Return Values

Returns a substring of characters from *string*.

Description

Mid$ returns a substring of as many as *count* characters starting with the *position* character in *string*.

See Also

Asc, Chr$, InStr, Left$, Len, Right$, Space$, Str$, Val

Mid$ Function Example

The example shown below shows a program that extracts the middle 2 characters from the string "ABCDEFGHIJ" and the remainder of the string starting at position 5.

```
Function midtest
    String basestr$, m1$, m2$
    basestr$ = "ABCDEFGHIJ"
    m1$ = Mid$(basestr$, (Len(basestr$) / 2), 2)
    Print "The middle 2 characters are: ", m1$
    m2$ = Mid$(basestr$, 5)
    Print "The string starting at 5 is: ", m2$
Fend
```

MkDir Statement

> S

Creates a subdirectory on a controller disk drive.

Syntax

MkDir *dirName*

Parameters

dirName String expression that defines the path and name of the directory to create.

Description

Creates a subdirectory in the specified path.

Only one subdirectory can be created at a time.

When executed from the Monitor window, quotes may be omitted.

See Also

ChDir, ChDrive, Dir, RenDir, RmDir

MkDir Example

The following examples are done from the monitor window:

```
> mkdir \USR
> mkdir \USR\PNT
```

Mod Operator

Returns the remainder obtained by dividing a numeric expression by another numeric expression.

Syntax

number **Mod** *divisor*

Parameters

number The number being divided (the dividend).

divisor The number which *number* is divided by.

Return Values

Returns the remainder after dividing *number* by *divisor*.

Description

Mod is used to get the remainder after dividing 2 numbers. The remainder is a whole number. One clever use of the **Mod** instruction is to determine if a number is odd or even. The method in which the **Mod** instruction works is as follows: *number* is divided by *divisor*. The remainder left over after this division is then the return value for the **Mod** instruction.

See Also

Abs, Atan, Atan2, Cos, Int, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Mod Operator Example

The example shown below determines if a number (var1) is even or odd. When the number is even the result of the Mod instruction will return a 0. When the number is odd, the result of the Mod instruction will return a 1.

```
Function modtest
....Integer var1, result

....Print "Enter an integer number:"
....Input var1
....result = var1 Mod 2
....Print "Result = ", result
....If result = 0 Then
........Print "The number is EVEN"
....Else
........Print "The number is ODD"
....EndIf
Fend
```

Some other example results from the Mod instruction from the Monitor window.

```
> Print  36 Mod 6
> 0

> Print  25 Mod 10
> 5
>
```

Motor Statement

> S

Turns motor power for all axes on or off for the current robot.

Syntax

Motor ON | OFF

Parameters

ON | OFF The keyword **ON** is used to turn the Motor Power on. The keyword **OFF** is used to turn Motor Power Off.

Description

The **Motor** On command is used to turn Motor Power On and release the brakes for all axes. **Motor** Off is used to turn Motor Power Off and set the brakes.

On a multi-robot system, use the Robot command to select the current robot before executing the **Motor** command.

In order to move the robot, motor power must be turned on. Incremental encoder robots also require Mcal to be executed once after the system is started.

After an emergency stop, or after an error has occurred that requires resetting with the Reset command, execute Reset, and then execute **Motor** On.

Motor On automatically sets the following items:

Power	Low
Fine	Default values
Speed	Default values
SpeedR	Default values
SpeedS	Default values
Accel	Default values
AccelS	Default values
AccelR	Default values
PTPBoost	Default values
LimZ	0

See Also

Brake, Power, Reset, Robot, SFree, SLock

Motor Example

The following examples are done from the monitor window:

```
> Motor On

> Motor Off
```

Motor Function

F

Returns status of motor power for the current robot.

Syntax

Motor

Return Values

0 = Motors off, 1 = Motors on.

See Also

Motor Statement

Motor Function Example

```
If Motor = Off Then
    Motor On
EndIf
```

Move Statement

> S

Moves the arm from the current position to the specified point using linear interpolation (i.e. moving in a straight line) at a constant tool center point velocity).

Syntax

Move *destination* [**ROT**] [**ECP**] [**CP**] [*searchExpr*] [*!...!*]

Parameters

destination	The target destination of the motion using a point expression.
ROT	Optional. :Decides the speed/acceleration/deceleration in favor of tool rotation.
ECP	Optional. External control point motion. This parameter is valid when the ECP option is enabled.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to execute I/O and other commands during motion.

Description

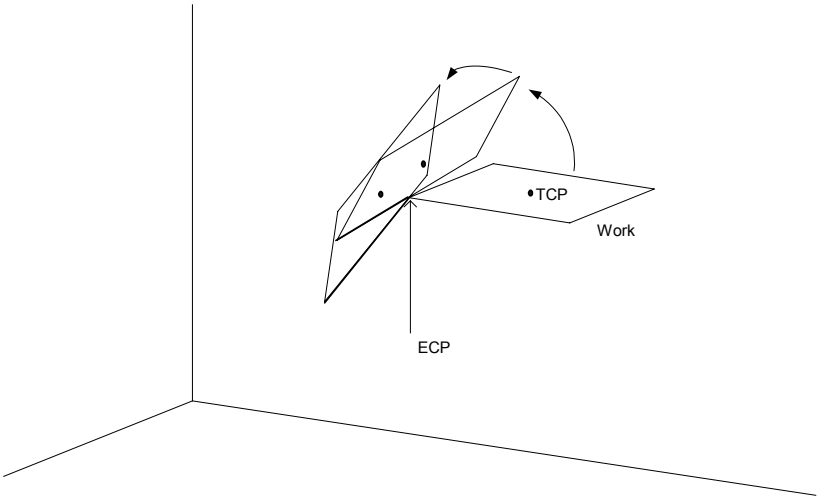
Move moves the arm from the current position to *destination* in a straight line. **Move** coordinates all axes to start and stop at the same time. The coordinates of *destination* must be taught previously before executing the **Move** instruction. Acceleration and deceleration for the **Move** is controlled by the AccelS instruction. Speed for the move is controlled by the SpeedS instruction. If the SpeedS speed value exceeds the allowable speed for any joint, power to all four joint motors will be turned off, and the robot will stop.

Move uses the SpeedS speed value and AccelS acceleration and deceleration values. Refer to *Using Move with CP* below on the relation between the speed/acceleration and the acceleration/deceleration. If, however, the ROT modifier parameter is used, **Move** uses the SpeedR speed value and AccelR acceleration and deceleration values. In this case SpeedS speed value and AccelS acceleration and deceleration value have no effect.

Usually, when the move distance is 0 and only the tool orientation is changed, an error will occur. However, by using the ROT parameter and giving priority to the acceleration and the deceleration of the tool rotation, it is possible to move without an error. When there is not an orientational change with the ROT modifier parameter and movement distance is not 0, an error will occur.

Also, when the tool rotation is large as compared to move distance, and when the rotation speed exceeds the specified speed of the manipulator, an error will occur. In this case, please reduce the speed or append the ROT modifier parameter to give priority to the rotational speed/acceleration/deceleration.

When ECP is used, the trajectory of the external control point coresponding to the ECP number specified by ECP instruction moves straight with respect to the tool coordinate system. In this case, the trajectory of tool center point does not follow a straight line.



The optional Till qualifier allows the user to specify a condition to cause the robot to decelerate to a stop prior to completing the **Move**. The condition specified is simply a check against one of the inputs. This is accomplished through using the Sw instruction. The user can check if the input is On or Off and cause the arm to stop based on the condition specified. This feature works almost like an interrupt where the **Move** is interrupted (stopped) once the Input condition is met. If the input condition is never met during the **Move** then the arm successfully arrives on the point specified by *destination.* For more information about the Till qualifier see the Till command.

Notes

Move Cannot

1) Move cannot execute motion that only changes tool orientation.

2) Move cannot execute range verification of the trajectory prior to starting the move itself. Therefore, even for target positions that are within an allowable range, it is possible for the system to find a prohibited position along the way to a target point. In this case, the arm may abruptly stop which may cause shock and a servo out condition of the arm. To prevent this, be sure to perform range verifications at low speed prior to using **Move** at high speeds. In summary, even though the target position is within the range of the arm, there are some **Move**s which will not work because the arm cannot physically make it to some of the intermediate positions required during the **Move**.

Using Move with CP

The CP parameter causes the arm to move to *destination* without decelerating or stopping at the point defined by *destination*. This is done to allow the user to string a series of motion instructions together to cause the arm to move along a continuous path while maintaining a specific speed throughout all the motion. The **Move** instruction without CP always causes the arm to decelerate to a stop prior to reaching the point destination *destination.*

Move with CP is equivalent to CMove on the SRC 3xx controllers.

Proper Speed and Acceleration Instructions with Move

The SpeedS and AccelS instructions are used to specify the speed and acceleration of the manipulator during **Move** motion. Pay close attention to the fact that SpeedS and AccelS apply to linear and circular interpolated motion while point to point motion uses the Speed and Accel instructions.

Potential Errors

Attempt to Change Only Tool Orientation

Changing only tool orientation during the move is impossible. If this is attempted, an error will occur.

Joint Overspeed Errors

When the motion requested results in the speed of one of the axes to exceed its maximum allowable speed an overspeed error occurs. In the case of a motor overspeed error, the robot arm is brought to a stop and servo power is turned off.

No Motion Instruction Following Move CP Instruction

If, following **Move** CP, there are not any motion instructions to smoothly move the arm (since the arm has not yet decelerated) damage may occur to the arm due to excessive shock. In this case an error will occur, the arm will stop and servo power will be removed. (A Pause input during continuous path motion may cause an error also due to the fact that it tries to instantaneously stop the robot.)

See Also

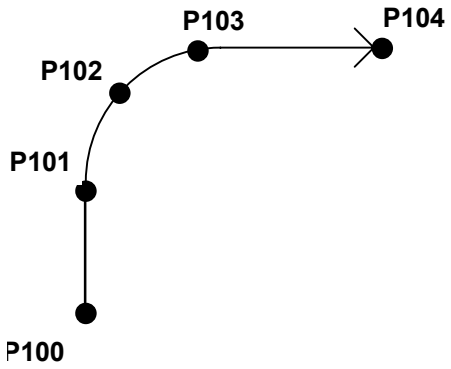
AccelS, Arc, Go, Jump, Jump3, Jump3CP, SpeedS, Sw, Till

Move Statement Example

The example shown below shows a simple point to point move between points P0 and P1 and then moves back to P0 in a straight line. Later in the program the arm moves in a straight line toward point P2 until input #2 turns on. If input #2 turns On during the Move, then the arm decelerates to a stop prior to arriving on point P2 and the next program instruction is executed.

```
Function movetest
  Home
  Go P0
  Go P1
  Move P2 Till Sw(2) = On
  If Sw(2) = On Then
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
  Else
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
  EndIf
Fend
```

This example uses **Move** with CP. The diagram below shows arc motion which originated at the point P100 and then moves in a straight line through P101, at which time the arm begins to form an arc. The arc is then continued through P102 and on to P103. Next the arm moves in a straight line to P104 where it finally decelerates to a stop. Note that the arm doesn't decelerate between each point until its final destination of P104. The following function would generate such a motion.




```
Function CornerArc
   Go P100
   Move CP P101       'Do not stop at P101
   Arc CP P102, P103  'Do not stop at P103
   Move P104          'Decelerate to stop at P104
Fend
```

MsgBox Statement

> S

Displays a message in a dialog box and waits for the user to choose a button.

Syntax

MsgBox *msg$,* [*type*], [*title$*], [*answer*]

Parameters

msg$ The message that will be displayed.

type Optional. A numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button. EPSON RC+ includes predefined constants that can be used for this parameter. The following table shows the values that can be used.

Symbolic constant	Value	Meaning
MB_OK	0	Display OK button only.
MB_OKCANCEL	1	Display OK and cancel buttons.
MB_ABORTRETRYIGNORE	2	Display Abort, Retry, and Ignore buttons.
MB_YESNOCANCEL	3	Display Yes, No, and Cancel buttons.
MB_YESNO	4	Display Yes and No buttons.
MB_RETRYCANCEL	5	Display Retry and Cancel buttons.
MB_ICONSTOP	16	Stop sign.
MB_ICONQUESTION	32	Question mark.
MB_ICONEXCLAMATION	64	Exclamation mark.
MB_DEFBUTTON1	0	First button is default.
MB_DEFBUTTON2	256	Second button is default.

title$ Optional. String expression that is displayed in the title bar of the dialog box.

answer Optional. An integer variable that receives a value indicating the action taken by the user. EPSON RC+ includes predefined constants that can be used for this parameter. The table below shows the values returned in *answer*.

Symbolic constant	Value	Meaning
IDOK	1	OK button selected.
IDCANCEL	2	Cancel button selected.
IDABORT	3	Abort button selected.
IDRETRY	4	Retry button selected.
IDYES	6	Yes button selected.
IDNO	7	No button selected.

Description

MsgBox automatically formats the message. If you want blank lines, use CRLF in the message. See the example.

See Also

InputBox

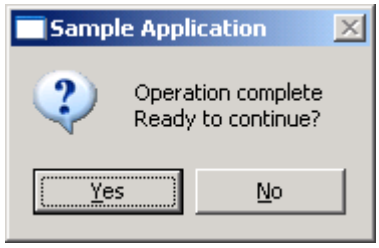
MsgBox Example

This example displays a message box that asks the operator if he/she wants to continue or not. The message box will display two buttons: Yes and No. A question mark icon will also be displayed. After MsgBox returns (after the user clicks a button), then the answer is examined. If it's no, then all tasks are stopped with the Quit command.

```
Function msgtest
   String msg$, title$
   Integer mFlags, answer

   msg$ = "Operation complete" + CRLF
   msg$ = msg$ + "Ready to continue?"
   title$ = "Sample Application"
   mFlags = MB_YESNO + MB_ICONQUESTION
   MsgBox msg$, mFlags, title$, answer
   If answer = IDNO then
      Quit All
   EndIf
Fend
```

A picture of the message box that this code will create is shown below.



MyTask Function

F

Returns the task number of the current program.

Syntax

MyTask

Return Values

The task number of the current task. Valid entries are integer numbers 1-32.

Description

MyTask returns the task number of the current program with a numeral. The **MyTask** instruction is inserted inside a specific program and when that program runs the **MyTask** function will return the task number that the program is running in.

See Also

Xqt

MyTask Function Example

The following program switches On and Off the I/O ports from 1 to 8.

```
Function main
  Xqt 2, task    'Execute task 2.
  Xqt 3, task    'Execute task 3.
  Xqt 4, task    'Execute task 4.
  Xqt 5, task    'Execute task 5.
  Xqt 6, task    'Execute task 6.
  Xqt 7, task    'Execute task 7.
  Xqt 8, task    'Execute task 8.
  Call task
Fend

Function task
  Do
    On MyTask            'Switch On I/O port which has the
                         'same number as current task number
    Off MyTask           'Switch Off I/O port which has the
                         'same number as current task number
  Loop
Fend
```

Next

> S

The For/Next instructions are used together to create a loop where instructions located between the For and Next instructions are executed multiple times as specified by the user.

Syntax

For *var1* **=** *initval* **To** *finalval* **[Step** *Increment* **]**
statements
Next *var1*

Parameters

var1	The counting variable used with the For/Next loop. This variable is normally defined as an integer but may also be defined as a Real variable.
initval	The initial value for the counter *var1*.
finalval	The final value of the counter var1. Once this value is met, the For/Next loop is complete and execution continues starting with the statement following the Next instruction.
Increment	An optional parameter which defines the counting increment for each time the Next statement is executed within the For/Next loop. This variable may be positive or negative. However, if the value is negative, the initial value of the variable must be larger than the final value of the variable. If the increment value is left out the system automatically increments by 1.
statements	Any valid SPEL$^+$ statements can be inserted inside the For/Next loop.

Return Values

None

Description

For/Next executes a set of statements within a loop a specified number of times. The beginning of the loop is the For statement. The end of the loop is the **Next** statement. A variable is used to count the number of times the statements inside the loop are executed.

The first numeric expression (*initval)* is the initial value of the counter. This value may be positive or negative as long as the *finalval* variable and Step increment correspond correctly.

The second numeric expression (*finalval*) is the final value of the counter. This is the value which once reached causes the For/**Next** loop to terminate and control of the program is passed on to the next instruction following the **Next** instruction.

Program statements after the For statement are executed until a **Next** instruction is reached. The counter variable (*var1*) is then incremented by the Step value defined by the *increment* parameter. If the Step option is not used, the counter is incremented by one.

The counter variable (*var1*) is then compared with the final value (*finalval*). If the counter is less than or equal to the final value (*finalval*), the statements following the For instruction are executed again. If the counter variable is greater than the final value (*finalval*), execution branches outside of the For/**Next** loop and continues with the instruction immediately following the **Next** instruction.

Nesting of For/**Next** statements is supported up to 10 levels deep. This means that a For/**Next** Loop can be put inside of another For/**Next** loop and so on and so on until there are 10 "nests" of For/**Next** loops.

Notes

Negative Step Values

If the value of the Step increment (*increment*) is negative, the counter variable (var1) is decremented (decreased) each time through the loop and the initial value (*initval*) must be greater than the final value (*finalval*) for the loop to work.

See Also

For

For/Next Example

```
Function fornext
    Integer ctr
    For ctr = 1 to 10
        Go Pctr
    Next ctr
    '
    For ctr = 10 to 1 Step -1
        Go Pctr
    Next ctr
Fend
```

Not Operator

F

Performs the bitwise complement on the value of the operand.

Syntax

Not *operand*

Parameters

operand	Integer expression.

Return Values

1's complement of the value of the operand.

Description

The **Not** function performs the bitwise complement on the value of the operand. Each bit of the result is the complement of the corresponding bit in the operand, effectively changing 0 bits to 1, and 1 bits to 0.

See Also

Abs, And, Atan, Atan2, Cos, Int, LShift, Mod, Or, RShift, Sgn, Sin, Sqr, Str$, Tan, Val, Xor

Not Operator Example

This is a simple Monitor window example on the usage of the **Not** instruction.

```
>print not(1)
 -2
>
```

Off Statement

> S

Turns Off the specified output and after a specified time can turn it back on.

Syntax

Off { *bitNumber* | *outputLabel* }, [*time*], [*parallel*]

Parameters

bitNumber Integer expression whose value is between 0 - 511. Tells the **Off** instruction which Output to turn Off.

outputLabel Output label.

time Optional. Specifies a time interval in seconds for the output to remain Off. After the time interval expires, the Output is turned back on. The minimum time interval is 0.01 seconds and maximum time interval is 10 seconds.

parallel Optional. When a timer is set, the parallel parameter may be used to specify when the next command executes:

0 - immediately after the output is turned off

1 - after the specified time interval elapses. (default value)

Description

Off turns off (sets to 0) the specified output.

If the *time* interval parameter is specified, the output bit specified by *bitNumber* is switched off, and then switched back on after the *time* interval elapses. If prior to executing Off, the Output bit was already off, then it is switched On after the time interval elapses.

The *parallel* parameter settings are applicable when the time interval is specified as follows:
1: Switches the output off, switches it back on after specified interval elapses, then executes the next command. (This is also the default value for the parallel parameter. If this parameter is omitted, this is the same as setting the parameter to 1.)
0: Switches the output off, and simultaneously executes the next command.

Notes

Output bits Configured as Remote Control output

If an output bit which was set up as a system output is specified, an error will occur. Remote control output bits are turned on or off automatically according to system status.

Outputs and When an Emergency Stop Occurs:

EPSON RC+ has a feature which causes all outputs to go off when an E-Stop occurs. This feature is set or disabled from the SPEL Options tab on the System Configuration dialog accessible from the Setup Menu.

See Also

In, InBCD, MemOn, MemOff, MemOut, MemSw, OpBCD, Oport, Out, Wait

Off Statement Example

The example shown below shows main task start a background task called iotask. The iotask is a simple task to turn discrete output bits 1 and 2 on and then off, Wait 10 seconds and then do it again.

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
  .
Fend


Function iotask
  Do
    On 1
    On 2
    Off 1
    Off 2
    Wait 10
  Loop
Fend
```

Other simple examples from the Monitor window are as follows:

```
> on  1
> off 1, 10   'Turn Output 1 off, wait 10 secs, turn on again
> on 2
> off 2
```

Off $ Statement

> S

Note: The Off $ statement is obsolete and has been replaced by the MemOff statement from Epson RC+ 4.0.

Turns Off the specified memory I/O bit.

Syntax

Off ${ *bitNumber* | *memIOLabel* }

Parameters

bitNumber	Integer expression between 0-511 representing one of the 512 memory I/O bits. NOTE: The "$" must be put in front of the bit number or label to indicate that this is a memory bit and not a hardware output.
memIOLabel	Memory I/O label.

Description

Off $ turns Off(sets to 0) the specified bit of memory I/O. The 512 memory I/O bits are typically excellent choices for use as status bits for uses such as On/Off, True/False, Done/Not Done, etc. The On $ instruction turns the memory bit On, the **Off $** instruction turns it Off, and the Sw $ instruction is used to check the current state of the specified memory bit. The Wait instruction can also be used with the memory bit to cause the system to wait until a specified memory I/O status is set.

See Also

In, In$, InBCD, MemOff, Off, On, On$, OpBCD, Oport, Out, Out$, Sw, Sw$, Wait

Off$ Statement Example

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. Sw $ is used in combination with the Wait instruction to wait until the memory I/O #1 is the proper value before it is safe to move again. On $ and Off $ are used to turn On and turn Off the memory I/O for proper synchronization.

```
Function main
    Integer I
    Off $1
    Xqt 2, task2
    For I = 1 to 100
        Wait Sw($1) = 0
        Go P(i)
        On $1
    Next I
Fend


Function task2
    Integer I
    For I = 101 to 200
        Wait Sw($1) = 1
        Go P(i)
        Off $1
    Next I
Fend
```

Other simple examples from the monitor window are as follows:

```
> On  $1       'Switch memory I/O bit #1 on
> Print Sw($1)
1
> Off  $1     'Switch memory I/O bit #1 off
> Print Sw($1)
0
```

OLRate Statement

> S

Display overload rating for one or all joints for the current robot.

Syntax

OLRate [*jointNumber*]

Parameters

jointNumber Integer expression whose value is between 1 and 6.

Description

OLRate can be used to check whether a cycle is causing stress on the servo system. Factors such as temperature and current can cause servo errors during applications with high duty cycles. **OLRate** can help to check if the robot system is close to having a servo error.

During a cycle, run another task to monitor **OLRate**. If **OLRate** exceeds 1.0 for any joint, then a servo error will occur.

Servo errors are more likely to occur with heavy payloads. By using **OLRate** during a test cycle, you can help insure that the speed and acceleration settings will not cause a servo error during production cycling.

To get valid readings, you must execute **OLRate** while the robot is moving.

See Also

OLRate Function

OLRate Statement Example

```
>olrate
0.10000  0.20000
0.30000  0.40000
0.50000  0.60000

Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Do
    ' Display OLRate
    OLRate
    Wait 1
  Loop
Fend
```

OLRate Function

F

Returns overload rating for one joint for the current robot.

Syntax

OLRate(*jointNumber***)**

Parameters

jointNumber Integer expression whose value is between 1 and 6.

Return Values

Returns the OLRate for the specified joint. Values are between 0.0 and 2.0.

Description

OLRate can be used to check whether a cycle is causing stress on the servo system. Factors such as temperature and current can cause servo errors during applications with high duty cycles. **OLRate** can help to check if the robot system is close to having a servo error.

During a cycle, run another task to monitor **OLRate**. If **OLRate** exceeds 1.0 for any joint, then a servo error will occur.

Servo errors are more likely to occur with heavy payloads. By using **OLRate** during a test cycle, you can help insure that the speed and acceleration settings will not cause a servo error during production cycling.

To get valid readings, you must execute **OLRate** while the robot is moving.

See Also

OLRate Statement

OLRate Function Example

```
Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Integer i
  Real olRates(4)
  Do
    For i = 1 to 4
      olRates(i) = OLRate(i)
      If olRate(i) > .5 Then
        Print "Warning: OLRate(", i, ") is over .5"
      EndIf
    Next i
  Loop
Fend
```

On Statement

> S

Turns on the specified output and after a specified time can turn it back off**.**

Syntax

On { *bitNumber* | *outputLabel* }, [*time*], [*parallel*]

Parameters

bitNumber	Integer expression whose value is between 0 - 511. Tells the **On** instruction which Output to turn On.
outputLabel	Output label.
time	Optional. Specifies a time interval in seconds for the output to remain On. After the time interval expires, the Output is turned back off. (Minimum time interval is 0.01 seconds)
parallel	Optional. When a timer is set, the parallel parameter may be used to specify when the next command executes: 0 - immediately after the output is turned on 1 - after the specified time interval elapses. (default value)

Description

On turns On (sets to 1) the specified output.
If the *time* interval parameter is specified, the output bit specified by *outnum* is switched On, and then switched back Off after the *time* interval elapses.

The *parallel* parameter settings are applicable when the time interval is specified as follows:
1: Switches the output On, switches it back Off after specified interval elapses, then executes the next command. (This is also the default value for the parallel parameter. If this parameter is omitted, this is the same as setting the parameter to 1.)
0: Switches the output On, and simultaneously executes the next command.

Notes

Output bits Configured as remote

If an output bit which was set up as remote is specified, an error will occur. Remote output bits are turned On or Off automatically according to system status. For more information regarding remote, refer to the EPSON RC+ User's Guide. The individual bits for the remote connector can be set as remote or I/O from the EPSON RC+ remote configuration dialog accessible from the setup menu.

Outputs and When an Emergency Stop Occurs

EPSON RC+ has a feature which causes all outputs to go off when an E-Stop occurs. This feature is set or disabled from one of the Option Switches. To configure this go to the SPEL Options tab on the System Configuration dialog accessible from the Setup Menu.

See Also

In, InBCD, MemOff, MemOn, Off, OpBCD, Oport, Out, Wait

On Statement Example

The example shown below shows main task start a background task called iotask. The iotask is a simple task to turn discrete output bits 1 and 2 on and then off, Wait 10 seconds and then do it again.

```
Function main
    Xqt iotask
    Go P1
    .
    .
    .
Fend


Function iotask
    '
    Do
        On 1
        On 2
        Off 1
        Off 2
        Wait 10
    Loop
Fend
```

Other simple examples from the monitor window are as follows:

```
> On 1
> Off 1
> On 2
> Off 2
```

On $ Statement

> S

Note: The On $ statement is obsolete and has been replaced by the MemOn statement from Epson RC+ 4.0.

Turns On the specified memory I/O bit.

Syntax

On $*bitNumber*

Parameters

bitNumber Number between 0-511 representing one of the 512 memory I/O bits. NOTE: The "$" must be put in front of the bit number or label to indicate that this is memory I/O and not a hardware output.

Description

On $ turns On (sets to 1) the specified bit of the robot memory I/O. The 512 memory I/O bits are typically excellent choices for use as status bits for uses such as On/Off, True/False, Finished/Not Finished, etc.. The On $ instruction turns the memory bit On, the Off $ instruction turns it Off, and the Sw $ instruction is used to check the current state of the specified memory bit. The Wait instruction can also be used with the memory bit to cause the system to wait until a specified memory I/O status is set.

See Also

In, In$, InBCD, Off, Off$, On, OpBCD, Oport, Out, Out$, Sw, Sw$, Wait

On$ Statement Example

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. Sw $ is used in combination with the Wait instruction to wait until the memory I/O #1 is the proper value before it is safe to move again. On $ and Off $ are used to turn On and turn Off the memory I/O for proper synchronization.

```
Function main
    Integer I
    Off $1
    Xqt 2, task2
    For I = 1 to 100
        Wait Sw($1) = 0
        Go P(i)
        On $1
    Next I
Fend

Function task2
    Integer I
    For I = 101 to 200
        Wait Sw($1) = 1
        Go P(i)
        Off $1
    Next I
Fend
```

Other simple examples from the monitor window are as follows:

```
> on $1
> print sw($1)
1
> off $1
> print sw($1)
0
```

OnErr

S

Sets up interrupt branching to cause control to transfer to an error handing subroutine when an error occurs. Allows users to perform error handling.

Syntax

OnErr GoTo {*label* | *lineNumber* | **0**}

Parameters

lineNumber	Statement line number to execute when an error occurs.
label	Statement label to jump to when an error occurs.
0	Parameter used to clear OnErr setting.

Description

OnErr enables user error handling. When an error occurs without **OnErr** being used, the task is terminated and the error is displayed. However, when **OnErr** is used it allows the user to "catch" the error and go to an error handler to automatically recover from the error. Upon receiving an error, **OnErr** branches control to the designated line number or label specified in the **OnErr** instruction. In this way the task is not terminated and the user is given the capability to automatically handle the error. This makes work cells run much smoother since potential problems are always handled and recovered from in the same fashion.

When the **OnErr** command is specified with the 0 parameter, the current **OnErr** setting is cleared. (i.e. After executing **OnErr** 0, if an error occurs program execution will stop)

See Also

EClr, Err

OnErr Example

The following example shows a simple utility program which checks whether points P0-P399 exist. If the point does not exist, then a message is printed on the screen to let the user know this point does not exist. The program uses the CX instruction to test each point for whether or not it has been defined. When a point is not defined control is transferred to the error handler and a message is printed on the screen to tell the user which point was undefined.

```
Function errDemo
    Integer i, errNum

    OnErr GoTo errHandler

    For i = 0 To 399
        temp = CX(P(i))
    Next i
    Exit Function
    '
    '
    '*****************************************
    '* Error Handler                         *
    '*****************************************
errHandler:
    errNum = Err
    ' Check if using undefined point
    If errNum = 78 Then
       Print "Point number P", i, " is undefined!"
    Else
       Print "ERROR: Error number ", errNum, " occurred while"
       Print "       trying to process point P", i, " !"
    EndIf
    EResume Next
Fend
```

OP_Cls Statement

S

Clears the OP (Operator Pendant) user page.

Syntax

Op_Cls

Description

Use **OP_Cls** to clear all text on the OP user page.

See Also

Op_Print
Op_Page

OP_Cls Statement Example

```
Function mainMenu

  Integer btn

  OP_Cls
  OP_Print 7, "Run Cycle"
  OP_Print 8, "Setup"
  OP_Print 9, "Exit"
  OP_Page 1
  Do
    btn = OP_Inkey
    Select btn
      Case 7
        Call RunCycle
      Case 8
        Call Setup
      Case 9
        OP_Cls
        OP_Page 0
        Exit Function
    Send
  Loop
Fend
```

OP_Inkey Function

F

Waits for OP (Operator Pendant) button to be pressed and returns button number.

Syntax

OP_Inkey

Return Values

KeyNumber	Key
1	PB1
2	PB2
3	PB3
4	PB4
5	PB5
6	PB6
7	F1
8	F2
9	F3
10	F4
11	TSW1-1
12	TSW1-2
13	TSW1-3
14	TSW1-4
15	TSW2-1
16	TSW2-2
17	TSW2-3
18	TSW2-4
19	TSW3-1
20	TSW3-2
21	TSW3-3
22	TSW3-4
23	TSW4-1
24	TSW4-2
25	TSW4-3
26	TSW4-4

Description

Use **OP_Inkey** to wait for one OP button to be pressed. If **OP_Inkey** is used again before the operator releases the button, it will first wait for the button to be released, then wait for the next button to be pressed.

The OP User page must be active before keys can be read. You can activate the User page from your program using OP_Page. For more information, refer to the OP500RC manual.

See Also

OP_Input, OP_Key, OP_Print

OP_Inkey Function Example

```
Function mainMenu

  Integer btn

  OP_Cls
  OP_Print 7, "Run Cycle"
  OP_Print 8, "Setup"
  OP_Print 9, "Exit"
  OP_Page 1
  Do
    btn = OP_Inkey
    Select btn
      Case 7
        Call RunCycle
      Case 8
        Call Setup
      Case 9
        OP_Cls
        OP_Page 0
        Exit Function
    Send
  Loop
Fend
```

OP_Input Statement

S

Gets input from operator using OP (Operator Pendant) keypad.

Syntax

OP_Input *prompt, default, var* [*, var...*]

Parameters

prompt	String expression containing the prompt for data entry.
default	String expression containing the default entry value.
var	One or more variables. Multiple variables can be used with the **OP_Input** command as long as they are separated by commas.

Description

OP_Input receives numeric data from the controlling device and assigns the data to the variable(s) used with the **OP_Input** instruction.

When executing the **OP_Input** instruction, the OP keypad appears. The *prompt* is displayed in the first 3 lines of the input area. Each line can be up to 24 characters. To display multiple lines, use CRLF to separate each line. If a line is more than 24 characters, it will be truncated.

Each inputted character is displayed on line 4 of the input area. After inputting data, the operator presses OK key on the OP and **OP_Input** returns.

When *default* is not a blank string, it is displayed in the input line when the input screen is displayed.

Notes

Rules for Numeric Input

When inputting numeric values and non-numeric data is found in the input other than the delimiter (comma), the **OP_Input** instruction discards the non-numeric data and all data following that non-numeric data.

Rules for String Input

When inputting strings, only numeric characters are permitted as data.

Other Rules for the OP_Input Instruction

- When more than one variable is specified in the instruction, the numeric data input intended for each variable has to be separated by a comma (",") character.
- Numeric variable names and string variable names are allowed. However, the input data type must match the variable type.

Potential Errors

Number of variables and input data differ

For multiple variables, the number of input data must match the number of Input variable names. When the number of the variables specified in the instruction is different from the number of numeric data received from the keyboard, an Error 30 will occur.

See Also

OP_Print

OP_Input Statement Example

```
Function GetCycleCount As Long

    Long cycles

    OP_Input "Enter cycles:" + CRLF + "(Max = 25)", "", cycles

    Print "New cycle count = ", cycles
    GetCycleCount = cycles
Fend
```

OP_Key Function

F

Returns status of the OP (Operator Pendant) user buttons and touch screen switches.

Syntax

OP_Key(*keyNumber***)**

Parameters

keyNumber Integer expression whose value is between 1 and 26. The table below shows the corresponding pendant key for each key number.

KeyNumber	Key
1	PB1
2	PB2
3	PB3
4	PB4
5	PB5
6	PB6
7	F1
8	F2
9	F3
10	F4
11	TSW1-1
12	TSW1-2
13	TSW1-3
14	TSW1-4
15	TSW2-1
16	TSW2-2
17	TSW2-3
18	TSW2-4
19	TSW3-1
20	TSW3-2
21	TSW3-3
22	TSW3-4
23	TSW4-1
24	TSW4-2
25	TSW4-3
26	TSW4-4

Return Values

0 = Off, 1 = On

Description

Use **OP_Key** to read user keys from the operator pendant. The OP User page must be active before keys can be read. You can activate the User page from your program using OP_Page. For more information, refer to the OP500RC manual.

See Also

OP_Page
OP_Print

OP_Key Function Example

```
If OP_Key(1) = On Then
  Call Initialize
EndIf
```

OP_Page Statement

S

Sets the current OP (Operator Pendant) page.

Syntax

OP_Page *pageNumber*

Parameters

pageNumber Integer expression whose value is 0 or 1.

PageNumber	Description
0	The last page before switching to the User page.
1	Activate the OP User page and display it.

Description

OP_Page is used to activate the pendant user page. If the user page is not activated and the user presses the User tab on the pendant, an error will be displayed saying that the user page is currently disabled.

See Also

OP_Key
OP_Print

OP_Page Example

```
OP_Print 7, "Func1"
OP_Print 8, "Func2"
OP_Print 9, "Func3"
OP_Page 1
```

OP_Print Statement

S

Prints text on the OP (Operator Pendant) Run and User pages.

Syntax

OP_Print *region, expression* [*, expression...*]

Parameters

region An integer expression that specifies the region to print. Values are from 0 to 10.

expression A number or string expression.

Region	Description
0	Run page text area. 24 normal-width characters. (24 characters, 1 line).
1 - 6	Captions for User page push buttons. 8 normal-width characters (4 characters, 2 lines).
7 - 10	Screen area. 27 normal-width characters (27 characters, 1 line)

Description

Use **OP_Print** to print text on the OP Run page or User page. On the Run page, you can print text. On the User page, you can print labels for the push buttons, or print text in the screen area (next to the function keys).

See Also

OP_Key
OP_Page

OP_Print Example

```
OP_Cls 1
OP_Print 7, "Func 1"
OP_Print 8, "Func 2"
OP_Print 9, "Func 3"
OP_Page 1
```

OpBCD Statement

> S

Simultaneously sets 8 output lines using BCD format. (Binary Coded Decimal)

Syntax

OpBCD portNumber, outData

Parameters

portNumber Integer expression whose value is between 0-63 representing one of the 64 output groups (each group contains 8 outputs) which make up the standard and expansion outputs for the system. Where the *portNumber* selection corresponds to the following outputs:

PortNumber	Outputs
0	0-7
1	8-15
2	16-23
3	24-31
...	...
63	504-511

outData Integer expression between 0-99 representing the output pattern for the output group selected by *portNumber*. The 2nd digit (called the 1's digit) represents the lower 4 outputs in the selected group and the 1st digit (called the 10's digit) represents the upper 4 outputs in the selected group.

Description

OpBCD simultaneously sets 8 output lines using the BCD format. The standard and expansion user outputs are broken into groups of 8. The *portNumber* parameter for the OpBCD instruction defines which group of 8 outputs to use where *portNumber* = 0 means outputs 0-7, *portNumber* = 1 means outputs 8-15, etc..

Once a port number is selected (i.e. a group of 8 outputs has be selected), a specific output pattern must be defined. This is done in Binary Coded Decimal format using the *outdata* parameter. The outdata parameter may have 1 or 2 digits. (Valid entries range from 0 to 99.) The 1st digit (or 10's digit) corresponds to the upper 4 outputs of the group of 8 outputs selected by *portNumber*. The 2nd digit (or 1's digit) corresponds to the lower 4 outputs of the group of 8 outputs selected by *portNumber*.

Since valid entries in BCD format range from 0-9 for each digit, every I/O combination cannot be met. The table below shows some of the possible I/O combinations and their associated *outnum* values assuming that *portNumber* is 0.

Output Settings (Output number)

Outnum Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

Notice that the Binary Coded Decimal format only allows decimal values to be specified. This means that through using Binary Coded Decimal format it is impossible to turn on all outputs with the **OpBCD** instruction. Please note that the maximum value for either digit for outnum is 9. This means that the

largest value possible to use with OpBCD is 99. In the table above it is easy to see that 99 does not turn all Outputs on. Instead it turns outputs 0, 3, 4, and 7 On and all the others off.

Notes

Difference between OpBCD and Out

The **OpBCD** and Out instructions are very similar in the SPEL+ language. However, there is one major difference between the two. This difference is shown below:

- The **OpBCD** instruction uses the Binary Coded Decimal format for specifying an 8 bit value to use for turning the outputs on or off. Since Binary Coded Decimal format precludes the values of &HA, &HB, &HC, &HD, &HE or &HF from being used, all combinations for setting the 8 output group cannot be satisfied.
- The Out instruction works very similarly to the OpBCD instruction except that Out allows the range for the 8 bit value to use for turning outputs on or off to be between 0-255 (vs. 0-99 for OpBCD). This allows all possible combinations for the 8 bit output groups to be initiated according to the users specifications.

Output bits Configured as Remote:

If an output bit which was set up as remote is specified to be turned on by **OpBCD**, an error will occur. Remote output bits are turned On or Off automatically according to system status. For more information regarding remote, refer to the EPSON RC+ User's Guide. The individual bits for the remote connector can be set as remote or I/O from the EPSON RC+ remote configuration dialog accessible from the setup menu.

Outputs and When an Emergency Stop Occurs:

EPSON RC+ has a feature which causes all outputs to go off when an E-Stop occurs. This feature is set or disabled from one of the Option Switches. To configure this go to the SPEL Options tab on the System Configuration dialog accessible from the Setup Menu.

See Also

In, InBCD, MemOff, MemOn, MemSw, Off, On, Oport, Out, Sw, Wait

OpBCD Function Example

The example shown below shows main task start a background task called iotask. The iotask is a simple task to flip flop between turning outputs 1 & 2 on and then outputs 0 and 3 on. When 1 & 2 are turned on, then 0 & 3 are also turned off and vice versa.

```
Function main
    Xqt 2, iotask
    Go P1
    .
    .
    .
Fend

Function iotask
    Do
        OpBCD 0, 6
        OpBCD 0, 9
        Wait 10
    Loop
Fend
```

Other simple examples from the monitor window are as follows:

```
> OpBCD 1,6  'Turns on Outputs 1 and 2
> OpBCD 2,1  'Turns on Output 8
> OpBCD 3, 91   'Turns on Output 24, 28, and 31
```

OpenCom Statement

S

Open an RS-232 communication port.

Syntax

OpenCom #*portNumber*

Parameters

portNumber Integer expression for port number to open. Values are from 1 to 16.

Description

An RS-232 port must be enabled before you can open it. See Comm Ports tab: System Configuration from the Setup Menu.

Note

Port number 3 and 4

When port number 3 or 4 is specified, the corresponding port is not open since COM3 and COM4 are reserved. An error occurs.

See Also

ChkCom, CloseCom, SetCom

OpenCom Statement Example

```
OpenCom #1
```

OpenNet Statement

S

Open a TCP/IP network port.

Syntax

OpenNet #*portNumber* **As** { **Client** | **Server** }

Parameters

portNumber Integer expression for port number to open. Range is 128 - 147.

Description

OpenNet opens a TCP/IP port for communication with another computer on the network.

One system should open as Server and the other as Client. It does not matter which one executes first.

See Also

ChkNet, CloseNet, SetNet

OpenNet Statement Example

For this example, two PCs have their TCP/IP settings configured as follows:

PC #1:
Port: #128
Host Name: PC2
TCP/IP Port: 1000

```
Function tcpip1
  OpenNet #128 As Server
  WaitNet #128
  Print #128, "Data from host 1"
Fend
```

PC #2:
Port: #128
Host Name: PC1
TCP/IP Port: 1000

```
Function tcpip2
  String data$
  OpenNet #128 As Client
  WaitNet #128
  Input #128, data$
  Print "received '", data$, "' from host 1"
Fend
```

Oport Function

F

Returns the state of the specified output.

Syntax

Oport(*outnum***)**

Parameters

outnum Number between 0-511 representing one of the standard or expansion discrete outputs.

Return Values

Returns the specified output bit status as either a 0 or 1.

0: Off status

1: On status

Description

Oport provides a status check for the outputs. It functions much in the same way as the Sw instruction does for inputs. **Oport** is most commonly used to check the status of one of the outputs which could be connected to a feeder, conveyor, gripper solenoid, or a host of other devices which works via discrete I/O. Obviously the output checked with the **Oport** instruction has 2 states (1 or 0). These indicate whether the specified output is On or Off.

Notes

Difference between Oport and Sw

It is very important for the user to understand the difference between the **Oport** and Sw instructions. Both instructions are used to get the status of I/O. However, the type of I/O is different between the two. The Sw instruction works inputs. The **Oport** instruction works with the standard and expansion hardware outputs. These hardware ports are discrete outputs which interact with devices external to the controller.

See Also

In, InBCD, MemIn, MemOn, MemOff, MemOut, MemSw, Off, On, OpBCD, Out, Sw, Wait

OPort Function Example

The example shown below turns on output 5, then checks to make sure it is on before continuing.

```
Function main
    TMOut 10
    OnErr errchk
    Integer errnum
    On 5    'Turn on output 5
    Wait Oport(5)
    Call mkpart1
    Exit Function

errchk:
    errnum = Err(0)
    If errnum = 94 Then
       Print "TIME Out Error Occurred during period"
       Print "waiting for Oport to come on. Check"
       Print "Output #5 for proper operation. Then"
       Print "restart this program."
    Else
       Print "ERROR number ", errnum, "Occurred"
       Print "Program stopped due to errors!"
    EndIf
    Exit Function
Fend
```

Other simple examples are as follows from the monitor window:

```
> On 1
> Print Oport(1)
1
> Off 1
> Print Oport(1)
0
>
```

Or Operator

Performs a bitwise or logical OR operation on two operands.

Syntax

expr1 **Or** *expr2*

Parameters

expr1, exrp2 Integer or Boolean expressions.

Return Values

Bitwise OR value of the operands if the expressions are integers. Logical OR if the expressions are Boolean.

Description

For integer expressions, the **Or** operator performs the bitwise OR operation on the values of the operands. Each bit of the result is 1 if one or both of the corresponding bits of the two operands is 1. For Boolean expressions, the result is True if either of the expressions evaluates to True.

See Also

And, LShift, Mod, Not, RShift, Xor

Or Operator Example

Here is an example of a bitwise OR.

```
>print 1 or 2
 3
```

Here is an example of a logical OR.

```
If a = 1 Or b = 2 Then
  c = 3
EndIf
```

Out Statement

> S

Simultaneously sets 8 output bits.

Syntax

Out *portNumber, outData*

Parameters

portNumber Integer expression between 0 and 63 representing one of the 64 output groups (each group contains 8 outputs) which make up the standard and expansion outputs. The portnum selection corresponds to the following outputs:

Portnum	Outputs
0	0-7
1	8-15
2	16-23
3	24-31
...	...
63	504-511

outData Integer number between 0-255 representing the output pattern for the output group selected by *portNumber*. If represented in hexadecimal form the range is from &H0 to &HFF. The lower digit represents the least significant digits (or the 1st 4 outputs) and the upper digit represents the most significant digits (or the 2nd 4 outputs).

Description

Out simultaneously sets 8 output lines using the combination of the *portNumber* and *outdata* values specified by the user to determine which outputs will be set. The outputs are broken into 64 groups of 8. The *portNumber* parameter for the OpBCD instruction defines which group of 8 outputs to use where *portNumber* = 0 means outputs 0-7, *portNumber* = 1 means outputs 8-15, etc..

Once a portnum is selected (i.e. a group of 8 outputs has be selected), a specific output pattern must be defined. This is done using the *outData* parameter. The *outData* parameter may have a value between 0-255 and may be represented in Hexadecimal or Integer format. (i.e. &H0-&HFF or 0-255)

The table below shows some of the possible I/O combinations and their associated *outData* values assuming that *portNumber* is 0, and 1 accordingly.

Output Settings When *portNumber*=0 (Output number)

OutData Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Output Settings When *portNumber*=1 (Output number)

OutData Value	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Notes

Difference between OpBCD and Out

The **Out** and OpBCD instructions are very similar in the SPEL$^+$ language. However, there is one major difference between the two. This difference is shown below:

- The OpBCD instruction uses the Binary Coded Decimal format for specifying 8 bit value to use for turning the outputs on or off. Since Binary Coded Decimal format precludes the values of &HA, &HB, &HC, &HD, &HE or &HF from being used, all combinations for setting the 8 output group cannot be satisfied.
- The **Out** instruction works very similarly to the OpBCD instruction except that **Out** allows the range for the 8 bit value to use for turning outputs on or off to be between 0-255 (vs. 0-99 for OpBCD). This allows all possible combinations for the 8 bit output groups to be initiated according to the users specifications.

See Also

In, InBCD, MemOff, MemOn, MemOut, MemSw, Off, On, Oport, Sw, Wait

Out Example

The example shown below shows main task start a background task called iotask. The iotask is a simple task to flip flop between turning output bits 0-3 On and then Off. The Out instruction makes this possible using only 1 command rather than turning each output On and Off individually.

```
Function main

  Xqt iotask
  Do
    Go P1
    Go P2
  Loop
Fend

Function iotask

  Do
    Out 0, &H0F
    Out 0, &H00
    Wait 10
  Loop
Fend

Other simple examples from the monitor window are as follows:

> Out 1,6  'Turns on Outputs 9 & 10
> Out 2,1  'Turns on Output 8
> Out 3,91 'Turns on Outputs 24, 25, 27, 28, and 30
```

Out Function

F

Returns the status of one byte of outputs.

Syntax

Out(*portNumber***)**

Parameters

portNumber	Integer expression between 0 and 63 representing one of the 64 output groups (each group contains 8 outputs) which includes both the standard and expansion hardware outputs. Where the *portNumber* selection corresponds to the following outputs:

Return Values

The output status 8 bit value for the specified port.

See Also

Out Statement

Out Function Example

```
Print Out(0)
```

Out $ Statement

> S

Note: The Out $ statement is obsolete and has been replaced by the MemOut statement from Epson RC+ 4.0.

Simultaneously sets 8 memory I/O bits.

Syntax

Out $ *portNumber, outData*

Parameters

portNumber Integer expression between 0 - 63 representing one of the 64 output groups (each group contains 8 outputs) which make up the 512 memory I/O outputs. The *portNumber* selection corresponds to the following outputs:

Portnum	Outputs
0	0-7
1	8-15
.	.
63	496-511

outData Integer expression between 0-255 representing the output pattern for the output group selected by *portNumber*. If represented in hexadecimal form the range is from &H0 to &HFF. The lower digit represents the least significant digits (or the 1st 4 outputs) and the upper digit represents the most significant digits (or the 2nd 4 outputs).

Description

Out $ simultaneously sets 8 memory I/O outputs using the combination of the *portNumber* and *outData* values specified by the user to determine which outputs will be set. The 512 memory outputs are broken into 64 groups of 8. The *portNumber* parameter for the OpBCD instruction defines which group of 8 outputs to use where *portNumber* = 0 means outputs 0-7, *portNumber* = 1 means outputs 8-15, etc.

Once a *portNumber* is selected (i.e. a group of 8 outputs has be selected), a specific output pattern must be defined. This is done using the *outData* parameter. The *outData* parameter may have a value between 0-255 and may be represented in Hexadecimal or Integer format. (i.e. &H0-&HFF or 0-255)

The table below shows some of the possible I/O combinations and their associated *outData* values assuming that *portNumber* is 0, and 1 accordingly.

Output Settings When *portNumber*=0 (Output number)

OutData Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Output Settings When *portNumber*=1 (Output number)

OutData Value	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Notes

Difference between Out and Out $

It is very important for the user to understand the difference between the Out and **Out $** instructions. This difference is shown below:

- The **Out $** instruction works with internal memory I/O and does not affect hardware I/O in any way.
- The Out instruction works with the hardware output ports located on the rear of the controller. These hardware ports are discrete Outputs which interact with devices external to the robot.

See Also

In, In$, InBCD, Off, Off$, On, On$, OpBCD, Oport, Out, Sw, Sw$, Wait

Out $ Example

The example shown below shows main task start background task called iotask. The iotask is a simple task to flip flop between turning memory I/O #'s 0-3 On and then Off. The Out $ instruction makes this possible though using only 1 command rather than turning each memory I/O On $ and Off$ individually.

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
Fend

Function iotask


  Out $ 0,&H0
  Wait 0
  GoTo 0
Fend
```

Other simple examples from the monitor window are as follows:

```
> Out 1,6    'Turns on Outputs 9 & 10
> Out 2,1    'Turns on Output 8
> Out 3,91   'Turns on Outputs 24, 25, 27, 28, and 30
```

OutW Statement

Simultaneously sets 16 output bits.

Syntax

OutW *wordPortNum, outputData*

Parameters

wordPortNum	Specifies word port numbers (integer between 0 and 31, within the mounted I/O board range) using an expression or numeric value.
outputData	Specifies output data (integers from 0 to 65535) using an expression or numeric value.

Description

Changes the current status of user I/O output port group specified by the word port number to the specified output data.

See Also

In, InW, Out

OutW Example

```
OutW 0, 25
```

OutW Function

F

Returns the status of one word (2 bytes) of outputs.

Syntax

OutW(*wordPortNum***)**

Parameters

wordPortNum	Specifies word port numbers (integer between 0 and 31, within the mounted I/O board range) using an expression or numeric value.

Return Values

The output status 16 bit value for the specified port.

See Also

OutW Statement

OutW Function Example

```
OutW 0, &H1010
```

PAgl Function

F

Return a joint value from a specified point.

Syntax

PAgl (*point*, *jointNumber)*

Parameters

point	Point expression.
jointNumber	Specifies the joint number (integer from 1 to 6) using an expression or numeric value.

Return Values

Returns the calculated joint position (real value, deg for rotary joint, mm for prismatic joint).

See Also

Agl, CX, CY, CZ, CU, CV, CW, PPls

PAgl Function Example

```
Real joint1

joint1 = PAgl(P10, 1)
```

Pallet Statement

> S

Defines and displays pallets.

Syntax

Pallet [**Outside**,] [*palletNumber, Pi, Pj, Pk* [,*Pm*], *columns, rows*]

Parameters

Outside	Optional. Allow row and column indexes outside of the range of the specified rows and columns. Range is from -32768 to 32767.
palletNumber	Pallet number represented by an integer number from 0 to 15.
Pi, Pj, Pk	Point variables which define standard 3 point pallet position.
Pm	Optional. Point variable which is used with Pi, Pj and Pk to define 4 point pallet.
columns	Integer expression representing the number of points on the Pi-to-Pj side of the pallet. Range is from 1-32767.
rows	Integer expression representing the number of points on the Pi-to-Pk side of the pallet. Range is from 1-32767.

Return Values

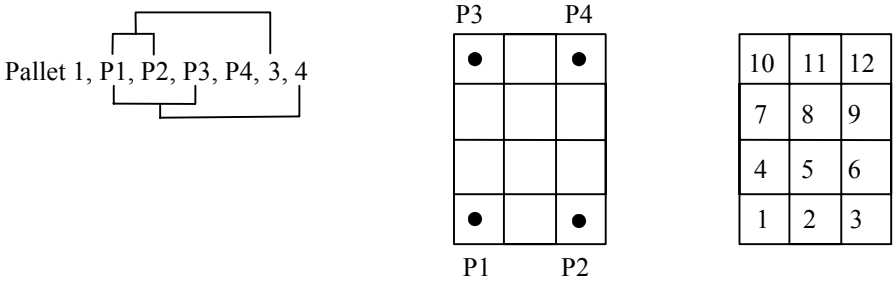
Displays all defined pallets when parameters are omitted.

Description

Defines a pallet by teaching the robot, as a minimum, points Pi, Pj and Pk and by specifying the number of points from Pi to Pj and from Pi to Pk.

If the pallet is a well ordered rectangular shape, only 3 of the 4 corner points need to be specified. However, in most situations it is better to use 4 corner points for defining a pallet.

To define a pallet, first teach the robot either 3 or 4 corner points, then define the pallet as follows:
A pallet defined with 4 points: P1, P2, P3 and P4 is shown below. There are 3 positions from P1-P2 and 4 positions from P1-P3. This makes a pallet which has 12 positions total. To define this pallet the syntax is as follows:



Points that represent divisions of a pallet are automatically assigned division numbers, which, in this example, begin at P1. These division numbers are also required by the Pallet Function.

When Outside is specified, row and column indexes outside of the range of rows and columns can be specified. The Outside should be specified by two-dimensional division. For example:

```
Pallet Outside 1, P1, P2, P3, 4, 5
Jump Pallet(1, -2, 10)
```

-2,10						
			1,5	2,5	3,5	4,5
			1,4	2,4	3,4	4,4
			1,3	2,3	3,3	4,3
			1,2	2,2	3,2	4,2
			1,1	2,1	3,1	4,1

Sample

		1,6			4,6		
-1,4		1,4	2,4	3,4	4,4		6,4
		1,3	2,3	3,3	4,3		
		1,2	2,2	3,2	4,2		
-1,1		1,1	2,1	3,1	4,1		6,1
		1,-1			4,-1		

Notes

The Maximum Pallet Size

The total number of points defined by a specific pallet must be less than 32,767.

Incorrect Pallet Shape Definitions

Be aware that incorrect order of points or incorrect number of divisions between points will result in an incorrect pallet shape definition.

Pallet Plane Definition

The pallet plane is defined by the Z axis coordinate values of the 3 corner points of the pallet. Therefore, a vertical pallet could also be defined.

Pallet Definition for a Single Row Pallet

A single row pallet can be defined with a 3 point Pallet statement or command. Simply teach a point at each end and define as follows: Specify 1 as the number of divisions between the same point.
> Pallet 2, P20, P21, P20, 5, 1 'Defines a 5x1 pallet

See Also

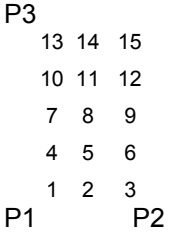
Pallet Function

Pallet Statement Example

The following instruction from the monitor window sets the pallet defined by P1, P2 and P3 points, and divides the pallet plane into 15 equally distributed pallet point positions, with the pallet point number 1, the pallet point number 2 and the pallet point number 3 sitting along the P1-to-P2 side.

```
> pallet 1, P1, P2, P3, 3, 5
> jump pallet(1, 2)          'Jump to position on pallet
```

The resulting Pallet is shown below:



Pallet Function

F

Specifies a position in a previously defined pallet.

Syntax

(1) Pallet (*palletNumber*, *palletPosition* **)**
(2) Pallet (*palletNumber*, *column, row* **)**

Parameters

palletNumber	Pallet number represented by integer expression from 0 to 15.
PalletPosition	The pallet position represented by an integer from 1 to 32767.
column	The pallet column represented by an integer expression from 1 to 32767.
row	The pallet row represented by an integer expression from 1 to 32767.

Description

Pallet returns a position in a pallet which was previously defined by the Pallet statement. Use this function with the Go or the Jump instruction to cause the arm to move the specified pallet position.

The pallet position number can be defined arithmetically or simply by using an integer.

See Also

Pallet Statement

Pallet Function Example

The following program transfers parts from pallet 1 to pallet 2.

```
Function main
  Integer index
  Pallet 1, P1, P2, P3, 3, 5      'Define pallet 1
  Pallet 2, P12, P13, P11, 5, 3   'Define pallet 2
  For index = 1 To 15
    Jump Pallet(1, index)     'Move to point index on pallet 1
    On 1     'Hold the work piece
    Wait 0.5
    Jump Pallet(2, index)     'Move to point index on pallet 2
    Off 1    'Release the work piece
    Wait 0.5
  Next I
Fend
```

ParseStr Statement / Function

F S

Parse a string and return array of tokens.

Syntax

ParseStr *inputString$, tokens$(), delimiters$*
numTokens = **ParseStr(***inputString$, tokens$(), delimiters$***)**

Parameters

inputString$	String expression to be parsed.
tokens$()	Output array of strings containing the tokens.
delimiters$	String expression containing one or more token delimiters.

Return Values

When used as a function, the number of tokens parsed is returned.

See Also

Redim, String

ParseStr Statement Example

```
String toks$(0)
Integer i

ParseStr "1 2 3 4", toks$(), " "

For i = 0 To UBound(toks)
   Print "token ", i, " = ", toks$(i)
Next i
```

Pass Statement

> S

Executes simultaneous four joint Point to Point motion, passing near but not through the specified points.

Syntax

Pass *point* [, {**On | Off**} *bitNumber* [*, point ...*]]

Parameters

point	**P***number* or **P**(*expr*) or point label.
bitNumber	The output bit to turn on or off. Integer number between 0 - 511 or output label.

Description

Pass moves the robot arm near but not through the specified point series.

To specify a point series, use points (P0,P1, ...) with commas between points.

To turn output bits on or off while executing motion, insert an On or Off command delimited with commas between points. The On or Off is executed before the robot reaches the point immediately preceding the On or Off.

If **Pass** is immediately followed by another **Pass**, control passes to the following **Pass** without the robot stopping at the preceding **Pass** final specified point.

If **Pass** is immediately followed by a motion command other than another Pass, the robot stops at the preceding **Pass** final specified point, but Fine positioning will not be executed.

If **Pass** is immediately followed by a command, statement, or function other than a motion command, the immediately following command, statement or function will be executed prior to the robot reaching the final point of the preceding Pass.

If Fine positioning at the target position is desired, follow the Pass with a Go, specifying the target position as shown in the following example:

```
Pass P5; Go P5; On 1; Move P10
```

The larger the acceleration / deceleration values, the nearer the arm moves toward the specified point. The **Pass** instruction can be used such that the robot arm avoids obstacles.

Notes

SPEL+ does not support continuous point specification

The SPEL language used on the SRC 3xx controllers allowed you to specify a continuous set of points. For example:

```
Pass P1-P10
```

However, with SPEL+, the statement above means to move to one point (P1 minus P10).
In order to specify continuous, you must specify all of the points.

See Also

Accel, Go, Jump, Speed

Pass Example

The example shows the robot arm manipulation by Pass instruction:

```
Function main
  Jump P1
  Pass P2   'Move the arm toward P2, and perform
            'the next instruction before reaching P2.
  On 2
  Pass P3
  Pass P4
  Off 0
  Pass P5
Fend
```

Pause Statement

S

Temporarily stops program execution all tasks for which pause is enabled.

Syntax

Pause

Description

When the **Pause** instruction is executed, program execution for all tasks with pause enabled is suspended. Also, if any task is executing a motion statement, it will be paused even if pause is not enabled for that task.

Notes

QP and its Affect on Pause

The QP instruction is used to cause the arm to stop immediately upon Pause or to complete the current move and then Pause the program. See the QP instruction help for more information.

See Also

Cont, Resume

Pause Statement Example

The example below shows the use of the **Pause** instruction to temporarily stop execution. The task executes program statements until the line containing the Pause command. At that point the task is paused. The user can then click the Run Window Continue Button to resume execution.

```
Function main

  Xqt monitor, NoPause  ' This task will not be paused
  Go P1
  On 1
  Jump P2
  Off 1
  Pause    'Suspend program execution
  Go P40
  Jump P50
Fend
```

PauseOn Function

F

Returns the current Pause status.

Syntax

PauseOn

Return Values

True if the controller is in the Pause state, otherwise False.

Description

PauseOn returns True when the controller is in a Pause state.

PauseOn is equivalent to `((Stat(0) And &H20000)<>0)`.

See Also

EStopOn, SafetyOn, Stat

PauseOn Function Example

```
If PauseOn Then
    Print "Pause condition is On"
EndIf
```

PDef Function

F

Returns the definition status of a specified point.

Syntax

PDef (*point*)

Parameters

point Integer expression or **P***number* or **P**(*expr*) or point label.

Return Values

True if the point is defined, otherwise False.

See Also

Here Statement, PDel

PDef Function Example

```
If Not PDef(1) Then
    Here P1
Endif
```

PDel

>

Deletes specified position data.

Syntax

PDel *firstPointNum ,* [*lastPointNum*]

Parameters

firstPointNum	The first point number in a sequence of points to delete. *firstPointNum* must be an integer.
lastPointNum	The last point number in a sequence of points to delete. *lastPointNum* must be an integer.

Description

Deletes specified position data from the controller's point memory for the current robot. Deletes all position data from *firstPointNum* up to and including *lastPointNum*. To prevent Error 2 from occurring, *firstPointNum* must be less than *lastPointNum*.

PDel Example

```
> p1=10,300,-10,0/L
> p2=0,300,-40,0
> p10=-50,350,0,0
> pdel 1,2       'Delete points 1 and 2
> plist
P10  =   -50.000,  350.000,    0.000,    0.000 /R /0
> pdel 50        'Delete point 50
> pdel 100,200   'Delete from point 100 to point 200
>
```

PLabel Statement

> S

Defines a label for a specified point.

Syntax

PLabel *pointNumber, newLabel*

Parameters

pointNumber An integer expression representing a point number.

newLabel A string expression representing the label to use for the specified point.

See Also

PDef Function, PLabel Function, PNumber Function

PLabel Statement Example

```
PLabel 1, "pick"
```

PLabel$ Function

F

Returns the point label associated with a point number.

Syntax

PLabel$(*point***)**

Parameters

point An integer expression or **P***number* or **P**(*expr*) or point label..

See Also

PDef Function, PLabel Statement, PNumber Function

PLabel$ Function Example

```
Print PLabel$(1)
Print PLabel$(P(i))
```

PList

>

Displays point data in memory for the current robot.

Syntax

(1) PList
(2) PList *
(3) PList *pointNumber*
(4) PList *startPoint*,
(5) PList *startPoint*, e*ndPoint*

Parameters

pointNumber	The number range is 0 to 999.
startPoint	The start point number. The number range is 0 to 999.
endPoint	The end point index. The number range is 0 to 999.

Return Values

Point data.

Description

Plist displays point data in memory for the current robot.

When there is no point data within the specified range of points, or a start point number is specified larger than the end point number, then no data will be displayed.

(1) PList

Displays the coordinate data for all points.

(2) PList *

Displays the coordinate data for the current position.

(3) PList pointIndex

Displays the coordinate data for the specified point.

(4) PList *startPoint*,

Displays the coordinate data for all points starting with *startPoint*.

(5) PList startPoint, endPoint

Displays the coordinate data for all points starting with *startPoint* and ending with *endPoint*.

PList Example

This example displays the point data of the current arm position for the current robot.

```
> plist *
WORLD:  X:  360.000 mm  Y:   10.000 mm  Z:    0.000 mm  U:    0.000 deg
JOINT:  1:    0.000 deg 2:    0.000 deg 3:    0.000 mm  4:    0.000 deg
PULSE:  1:        0 pls 2:        0 pls 3:        0 pls 4:        0 pls
```

This example displays the point data for one point.

```
> plist 1
P1   =   290.000,    0.000,  -20.000,    0.000 /R /0
>
```

This example displays the point data within the range of 10 and 20. In this example, only three points are found in this range.

```
> plist 10, 20
P10  =   290.000,    0.000,  -20.000,    0.000 /R /0
P12  =   300.000,    0.000,    0.000,    0.000 /R /0
P20  =   285.000,   10.000,  -30.000,   45.000 /R /0
>
```

This example displays the point data starting with point number 10.

```
> plist 10,
P10  =   290.000,    0.000,  -20.000,    0.000 /R /0
P12  =   300.000,    0.000,    0.000,    0.000 /R /0
P20  =   285.000,   10.000,  -30.000,   45.000 /R /0
P30  =   310.000,   20.000,  -50.000,   90.000 /R /0
>
```

PLocal Statement

> S

Sets the local attribute for a point.

Syntax

PLocal(*point***) =** *localNumber*

Parameters

point	Integer expression or **P***number* or **P**(*expr*) or point label.
localNumber	An integer expression representing the new local number. Range is 0 to 15.

See Also

PLocal Function

PLocal Statement Example

```
PLocal(pick) = 1
```

PLocal Function

F

Returns the local number for a specified point.

Syntax

PLocal(*point***)**

Parameters

point Integer expression or **P***number* or **P**(*expr*) or point label.

Return Values

Local number for specified point.

See Also

PLocal Statement

PLocal Function Example

```
Integer localNum

localNum = PLocal(pick)
```

Pls Function

F

Returns the current encoder pulse count for each joint at the current position.

Syntax

Pls*(jointNumber)*

Parameters

jointNumber The specific joint for which to get the current encoder pulse count.

Return Values

Returns a number value representing the current encoder pulse count for the joint specified by *jointNumber*.

Description

Pls is used to read the current encoder position (or Pulse Count) of each joint. These values can be saved and then used later with the Pulse command.

See Also

CX, CY, CZ, CU, CV, CW, Pulse

Pls Function Example

Shown below is a simple example to get the pulse values for each joint and print them.

```
Function plstest
  Real t1, t2, z, u
  t1 = pls(1)
  t2 = pls(2)
  z = pls(3)
  u = pls(4)
  Print "T1 joint current Pulse Value: ", t1
  Print "T2 joint current Pulse Value: ", t2
  Print "Z joint current Pulse Value: ", z
  Print "U joint current Pulse Value: ", u
Fend
```

PNumber Function

Returns the point number associated with a point label.

Syntax

PNumber(*pointLabel***)**

Parameters

pointLabel A point label used in the current point file or string expression containing a point label.

See Also

PDef Function, PLabel$ Function

PNumber Function Example

```
Integer pNum
String pointName$

pNum = PNumber(pick)

pNum = PNumber("pick")

pointName$ = "place"
pNum = PNumber(pointName$)
```

Point Assignment

> S

Defines a robot point by assigning it to a point expression.

Syntax

point = pointExpr

Parameters

point	A robot point specified as follows: **P***number* **P**(*expr*) *pointLabel*
pointExpr	Point expression.

Description

Define a robot point by setting it equal to another point or point expression.

See Also

Local, Pallet, PDef, PDel, Plist

Point Assignment Example

The following examples are done from the monitor window:

Assign coordinates to P1:

```
> P1 = 300,200,-50,100
```

Specify left arm posture:

```
> P2 = -400,200,-80,100/L
```

Add 20 to X coordinate of P2 and define resulting point as P3:

```
> P3 = P2 +X(20)
> plist 3
P3=-380,200,-80,100/L
```

Subtract 50 from Y coordinate of P2, substitute -30 for Z coordinate, and define the resulting point P4 as right arm posture:

```
>P4=P2 -Y(50) :Z(-30) /R
```

Add 90 to U coord of Pallet(3, 5), and define resulting point as P6:

```
> P5 = P*
> P6 = pallet(3,5) +U(90)
```

Point Expression

> S

Specifies a robot point for assignment and motion commands.

Syntax

point [{ + | - } *point*] [*local*] [*hand*] [*elbow*] [*wrist*] [*j4flag*] [*j6flag*] [*relativeOffsets*] [*absoluteCoords*]

Parameters

point	The base point specification. This can be one of the following: **P***number* **P**(*expr*) **P*** **Here** **Pallet**(*palletNumber*, *palletIndex*) *pointLabel* **XY**(*X*, *Y*, *Z*, *U,* [*V*], [*W*]) **JA**(*J1, J2, J3, J4,* [*J5*], [*J6*]) **Pulse**(*J1, J2, J3, J4,* [*J5*], [*J6*])
local	Optional. Local number from 1 to 15 preceded by a forward slash (**/0** to **/15**) or at sign (**@0** to **@15**). The forward slash means that the coordinates will be in the local. The at sign means that the coordinates will be translated into local coordinates. When using conveyor tracking, you can also specify a conveyor local coordinate system using **/CNV**(*expr*) or **@CNV**(*expr*) where *expr* is a valid conveyor number.
hand	Optional for SCARA and 6-axis robots. Specify /**L** or /**R** for lefty or righty hand orientation.
elbow	Optional for 6-axis robots. Specify /**A** or /**B** for above or below orientation.
wrist	Optional for 6-axis robots. Specify /**F** or /**NF** for flip or no flip orientation.
j4flag	Optional for 6-axis robots. Specify /**J4F0** or /**J4F1**.
j6flag	Optional for 6-axis robots. Specify /**J6F0** - /**J6F127**.
relativeOffsets	Optional. One or more relative coordinate adjustments. {**+** | **-**} {**X** | **Y** | **Z** | **U** | **V** | **W**} **(***expr***)** The TL offsets are relative offsets in the current tool coordinate system. {**+** | **-**} **{TLX** | **TLY** | **TLZ** | **TLU** | **TLV** | **TLW**} **(***expr***)**
absoluteCoords	Optional. One or more absolute coordinates. **:** {**X** | **Y** | **Z** | **U** | **V** | **W**} **(***expr***)**

Description

Point expressions are used in point assignment statements and motion commands.
You can add or subtract points if direct coordinates are not used. For example:

```
Go P1 + P2
P1 = P2 + XY(100, 100, 0, 0)
```

Using relative offsets

You can offset one or more coordinates relative to the base point. For example, the following statement moves the robot 20 mm in the positive X axis from the current position:

```
Go P* +X(20)
```

If you execute the same statement again, the robot will move an additional 20 mm along the X axis, because this is a relative move.

You can also use relative tool offsets:

```
Go P* +TLX(20) -TLY(5.5)
```

Using absolute coordinates

You can change one or more coordinates of the base point by using absolute coordinates. For example, the following statement moves the robot to the 20 mm position on the X axis:

```
Go P* :X(20)
```

If you execute the same statement again, the robot will not move because it is already in the absolute position for X from the previous move.

Relative offsets and absolute coordinates make is easy to temporarily modify a point. For example, this code moves quickly above the pick point by 10 mm using a relative offset for Z or 10 mm, then moves slowly to the pick point.

```
Speed fast
Jump pick +Z(10)
Speed slow
Go pick
```

This code moves straight up from the current position by specifying an absolute value of 0 for the Z joint:

```
LimZ 0
Jump P* :Z(0)
```

Using Locals

You can specify a local number using a forward slash or at sign. Each has a separate function.

Use the forward slash to mark the coordinates in a local. For example, adding a /1 in the following statement says that P1 will be at location 0,0,0,0 in local 1.

```
P1 = XY(0, 0, 0, 0) /1
```

Use the at sign to translate the coordinates into local coordinates. For example, here is how to teach a point in a local:

```
P1 = P* @1
```

P1 is set to the current position translated to its position in local 1.

To create a point in a conveyor local coordinate system:

```
P1 = P* /CNV(1)
```

To teach a point in a conveyor local:

```
P1 = P* @CNV(1)
```

See Also

Go, Local, Pallet, Pdel, Plist, Hand, Elbow, Wrist, J4Flag, J6Flag

Point Expression Example

Here are some examples of using point expressions in assignments statements and motion commands:

```
P1 = XY(300,200,-50,100)
P2 = P1 /R
P3 = pick /1
P4 = P5 + P6
P(i) = XY(100, 200, CZ(P100), 0)
Go P1 -X(20) :Z(-20) /R
Go Pallet(1, 1) -Y(25.5)
Move pick /R
Jump Here :Z(0)
Go P* :Z(-25.5)
Go JA(25, 0, -20, 180)
pick = XY(100, 100, -50, 0)

P1 = XY(300,200,-50,100, -90, 0)
P2 = P1 /F /B
P2 = P1 +TLV(25)
```

POrient Statement

> S

Note: The POrient statement is obsolete and has been replaced by the Hand statement from Epson RC+ 4.0.

Sets the orientation for a point.

Syntax

POrient(*point***) =** *orientation*

Parameters

point	Integer expression or **P***number* or **P**(*expr*) or point label.
orientation	An integer expression representing the new orientation value. 1 Righty 2 Lefty

See Also

POrient Function

POrient Statement Example

```
POrient(pick) = Righty
POrient(1) = Lefty
```

POrient Function

F

Note: The POrient function is obsolete and has been replaced by the Hand function from Epson RC+ 4.0.

Returns the orientation of a point.

Syntax

POrient(*point***)**

Parameters

point	Integer expression or **P***number* or **P**(*expr*) or point label.

Return Values

1 Righty (/R)
2 Lefty (/L)

See Also

POrient Statement

POrient Function Example

```
Print POrient(pick)
Print POrient(1)
Print POrient(P1)
```

PosFound Function

F

Returns status of Find operation.

Syntax

PosFound

Return Values

True if position was found during move, False if not.

See Also

Find

PosFound Function Example

```
Find Sw(5) = ON
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Error: Cannot find the sensor signal."
EndIf
```

Power Statement

> S

Previously Called - Lp

Switches Power Mode to high or low and displays the current status.

Power Syntax

(1) Power { **High | Low** }
(2) Power

Parameters

High | Low The setting can be High or Low. The default is Low.

Return Values

Displays the current Power status when parameter is omitted.

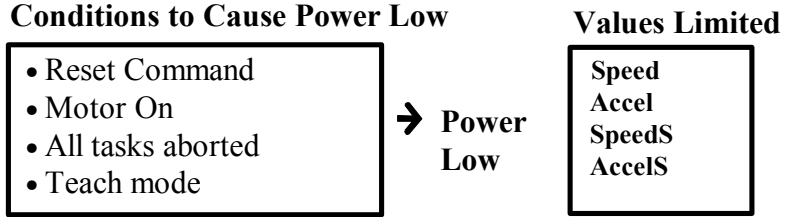
Description

Switches Power Mode to High or Low. It also displays the current mode status.

Low - When Power is set to Low, Low Power Mode is On. This means that the robot will run slow (below 250 mm/sec) and the servo stiffness is set light so as to remove servo power if the robot bumps into an object. This is the normal mode of operation for teaching points.

High - When Power is set to High, Low Power Mode is Off. This means that the robot can run at full speed with the full servo stiffness. This is the normal mode of operation for running actual applications.

The following operations will switch to low power mode. In this case, speed and acceleration settings will be limited to low values.



Notes

Low Power Mode (Power Low) and Its Effect on Max Speed:

In low power mode, motor power is limited, and effective motion speed setting is lower than the default value. If, when in Low Power mode, a higher speed is specified from the Monitor window (directly) or in a program, the speed is set to the default value. If a higher speed motion is required, set Power High.

High Power Mode (Power High) and Its Effect on Max Speed:

In high power mode, higher speeds than the default value can be set.

See Also

Accel, AccelS, Speed, SpeedS

Power Example

The following examples are executed from the monitor window:

```
> Speed 50  'Specifies high speed in Low Power mode

> Accel 100, 100  'Specifies high accel

> Jump P1   'Moves in low speed and low accel

> Speed         'Display current speed values
Low Power Mode
   50
   50     50

> Accel         'Display current accel values
Low Power Mode
   100    100
   100    100
   100    100

> Power High   'Set high power mode

> Jump P2   'Move robot at high speed
```

Power Function

F

Returns status of power.

Syntax

Power

Return Values

0 = Power Low, 1 = Power High.

See Also

Power Statement

Power Function Example

```
If Power = 0 Then
    Print "Low Power Mode"
EndIf
```

PPls Function

F

Return the pulse position of a specified joint value from a specified point.

Syntax

PPls (*point*, *jointNumber)*

Parameters

point	Point expression.
jointNumber	Specifies the joint number (integer from 1 to 6) using an expression or numeric value.

Return Values

Returns the calculated joint position (long value, in pulses).

See Also

Agl, CX, CY, CZ, CU, CV, CW, PAgl

PPls Example

```
Long pulses1

pulses1 = PPls(P10, 1)
```

Print Statement

> S

Outputs data to the current display window, including the Run window, Operator window, Monitor window, and Macro window.

Syntax

Print *expression* [*, expression...*] [,]
Print

Parameters

expression	Optional. A number or string expression.
, (comma)	Optional. If a comma is provided at the end of the statement, then a CRLF will not be added.

Return Values

Variable data or the specified character string.

Description

Print displays variable data or the character string on the controlling device.

An end of line CRLF (carriage return and line feed) is automatically appended to each output unless a comma is used at the end of the statement.

Notes

Make Sure Print is used with Wait or a motion within a loop

Tight loops (loops with no Wait or no motion) are generally not good, especially with Print.
The controller may freeze up in the worst case.
Be sure to use Print with Wait command or a motion command within a loop.

Bad example

```
Do
    Print "1234"
Loop
```

Good example

```
Do
    Print "1234"
    Wait 0.1
Loop
```

See Also

Print #, LPrint, EPrint

Print Statement Example

The following example extracts the U Axis coordinate value from a Point P100 and puts the coordinate value in the variable *uvar*. The value is then printed to the current display window.

```
Function test
  Real uvar
  uvar = CU(P100)
  Print "The U Axis Coordinate of P100 is ", uvar
Fend
```

Print # Statement

> S

Outputs data to the specified communications port.

Syntax

Print #*portNumber, expression* [*, expression...*]

Parameters

portNumber	The communication port number. One of the standard RS232 ports or TCP/IP ports. The port number should always be preceded with the # character.
expression	A numeric or string expression.
, (comma)	Optional. If a comma is provided at the end of the statement, then a CRLF will not be added.

Description

Print # outputs variable data, numerical values, or character strings to the communication port specified by *portNumber* .

See Also

Print

Print # Example

The following are some simple Print # examples:

```
Function printex
  String temp$
  Print #1, "5"     'send the character "5" to serial port 1
  temp$ = "hello"
  Print #1, temp$
  Print #2, temp$
  Print #1 " Next message for port 1"
  Print #2 " Next message for port 2"
Fend
```

PTCLR Statement

> S

Clears and intializes the peak torque for one or more joints.

Syntax

PTCLR [*j1*], [*j2*], [*j3*], [*j4*], [*j5*], [*j6*]

Parameters

j1 - j6	Optional. Integer expression representing the joint number. If no parameters are supplied, then the peak torque values are cleared for all joints.

Description

PTCLR clears the peak torque values for the specified joints.

You must execute PTCLR before executing PTRQ.

See Also

ATRQ, PTRQ

PTCLR Statement Example

```
> ptclr
> go p1
> ptrq 1
      0.227
> ptrq
      0.227    0.118
      0.249    0.083
      0.000    0.000
>
```

PTPBoost Statement

> S

Specifies or displays the acceleration, deceleration and speed algorithmic boost parameter for small distance PTP (point to point) motion.

Syntax

(1) PTPBoost *boost*, [*departBoost*], [*approBoost*]
(2) PTPBoost

Parameters

boost	Integer expression from 0 - 100.
departBoost	Optional. Jump depart boost value. Integer expression from 0 - 100.
approBoost	Optional. Jump approach boost value. Integer expression from 0 - 100.

Return Values

When parameters are omitted, the current PTPBoost settings are displayed.

Description

PTPBoost sets the acceleration, deceleration and speed for small distance PTP motion. It is effective only when the motion distance is small. The PTPBoostOK function can be used to confirm whether or not a specific motion distance to the destination is small enough to be affected by **PTPBoost** or not.

PTPBoost does not need modification under normal circumstances. Use **PTPBoost** only when you need to shorten the cycle time even if vibration becomes larger, or conversely when you need to reduce vibration even if cycle time becomes longer.

When the **PTPBoost** value is large, cycle time becomes shorter, but the positioning vibration increases. When **PTPBoost** is small, the positioning vibration becoms smaller, but cycle time becomes longer. Specifying inappropriate **PTPBoost** causes errors or can damage the manipulator. This may degrade the robot, or sometimes cause the manipulator life to shorten.

The **PTPBoost** value initializes to its default value when any one of the following is performed:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Abort All button or Ctrl + C Key

See Also

PTPBoost Function, PTPBoostOK

PTPBoost Statement Example

```
PTPBoost 50, 30, 30
```

PTPBoost Function

F

Returns the specified PTPBoost value.

Syntax

PTPBoost(*paramNumber***)**

Parameters

paramNumber Integer expression which can have the following values:

1: boost value

2: jump depart boost value

3: jump approach boost value

Return Values

Integer value from 0 - 100.

See Also

PTPBoost Statement, PTPBoostOK

PTPBoost Function Example

```
Print PTPBoost(1)
```

PTPBoostOK Function

F

Returns whether or not the PTP (Point to Point) motion from a current position to a target position is a small travel distance.

Syntax

PTPBoostOK(*targetPos***)**

Parameters

targetPos Point expression for the target position.

Return Values

True if is it possible to move to the target position from the current position in PTP motion, otherwise False.

Description

Use **PTPBoostOK** to the distance from the current position to the target position is small enough for PTPBoost to be effective.

See Also

PTPBoost

PTPBoostOK Function Example

```
If PTPBoostOK(P1) Then
  PTPBoost 50
EndIf
Go P1
```

PTPTime Function

F

Returns the estimated time for a point to point motion command without executing it.

Syntax

(1) PTPTime(*destination, destArm, destTool*)
(2) PTPTime(*start, startArm, startTool, destination, destArm, destTool*)

Parameters

start	Point expression for the starting position.
destination	Point expression for the destination position.
destArm	Integer expression for the destination arm number.
destTool	Integer expression for the destination tool number.
startArm	Integer expression for the starting point arm number.
startTool	Integer expression for the starting point tool number.

Return Values

Real value in seconds.

Description

Use PTPTime to calculate the time it would take for a point to point motion command (Go). Use syntax 1 to calculate time from the current position to the destination. Use syntax 2 to calculate time from a start point to a destination point.

The actual motion operation is not performed when this function is executed. The current position, arm, and tool settings do not change.

If the position is one that cannot be arrived at or if the arm or tool settings are incorrect, 0 is returned.

See Also

ATRQ, Go, PTRQ

PTPTime Function Example

```
Real secs

secs = PTPTime(P1, 0, 0, P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs

Go P1
secs = PTPTime(P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs
```

PTran Statement

S

Perform a relative move of one joint in pulses.

Syntax

PTran *joint, pulses*

Parameters

joint	Integer expression representing which joint to move.
pulses	Integer expression representing the number of pulses to move.

Description

Use **PTran** to move one joint a specified number of pulses from the current position.

See Also

Go, JTran, Jump, Move

PTran Statement Example

```
PTran 1, 2000
```

PTRQ Statement

> S

Displays the peak torque for the specified joint.

Syntax

PTRQ [*jointNumber*]

Parameters

jointNumber Optional. Integer expression representing the joint number.

Return Values

Displays current peak torque values for all joints.

Description

Use **PTRQ** to display the peak torque value for one or all joints since the PTCLR statement was executed.

Peak torque is a real number from 0 to 1.

See Also

ATRQ, PTCLR, PTRQ Function

PTRQ Statement Example

```
> ptclr
> go p1
> ptrq 1
      0.227
> ptrq
      0.227    0.118
      0.249    0.083
      0.000    0.000
>
```

PTRQ Function

F

Returns the peak torque for the specified joint.

Syntax

PTRQ(*jointNumber*)

Parameters

jointNumber Integer expression representing the joint number.

Return Values

Real value from 0 to 1.

See Also

ATRQ, PTCLR, PTRQ Statement

PTRQ Function Example

This example uses the **PTRQ** function in a program:

```
Function DisplayPeakTorque
   Integer i

   Print "Peak torques:"
   For i = 1 To 4
      Print "Joint ", i, " = ", PTRQ(i)
   Next i
Fend
```

Pulse Statement

> S

Moves the robot arm using point to point motion to the point specified by the pulse values for each joint.

Syntax

(1) Pulse *J1, J2, J3, J4 ,* [*J5*] *,* [*J6*]
(2) Pulse

Parameters

J1, J2, J3, J4	The pulse value for each of the first four joints. The pulse value has to be within the range defined by the Range instruction and should be an integer or long expression.
J5, J6	Optional. For use with 6-axis robots.

Return Values

When parameters are omitted, the pulse values for the current robot position are displayed.

Description

Pulse uses the joint pulse value from the zero pulse position to represent the robot arm position, rather than the orthogonal coordinate system. The Pulse instruction moves the robot arm using Point to Point motion.

The Range instruction sets the upper and lower limits used in the Pulse instruction.

Note

Make Sure Path is Obstacle Free Before Using Pulse

Unlike Jump, **Pulse** moves all axes simultaneously, including Z joint raising and lowering in traveling to the target position. Therefore, when using **Pulse**, take extreme care so that the hand can move through an obstacle free path.

Potential Errors

Pulse value exceeds limit:

If the pulse value specified in Pulse instruction exceeds the limit set by the Range instruction, an error will occur.

See Also

Go, Accel, Range, Speed, Pls, Pulse Function

Pulse Statement Example

Following are examples on the Monitor window:

This example moves the robot arm to the position which is defined by each joint pulse.

```
> pulse 16000, 10000, -100, 10
```

This example displays the pulse numbers of 1st to 4th axes of the current robot arm position.

```
> pulse
PULSE:  1:  27306 pls 2:  11378 pls 3:  -3072 pls 4:   1297 pls
>
```

Pulse Function

F

Returns a robot point whose coordinates are specified in pulses for each joint.

Syntax

Pulse (*J1, J2, J3, J4* , [*J5*] , [*J6*])

Parameters

J1, J2, J3, J4	The pulse value for joints 1 to 4. The pulse value must be within the range defined by the Range instruction and should be an integer or long expression.
J5, J6	Optional. For use with 6-axis robots.

Return Values

A robot point using the specified pulse values.

See Also

Go, JA, Jump, Move, Pulse Statement, XY

Pulse Function Example

```
Jump Pulse(1000, 2000, 0, 0)
```

QP Statement

> S

Switches Quick Pause Mode On or Off and displays the current mode status.

Syntax

(1) QP { **On** | **Off** }
(2) QP

Parameters

On | **Off**	Quick Pause can be either On or Off.

Return Values

Displays the current **QP** mode setting when parameter is omitted.

Description

If during motion command execution either the Pause switch is pressed, or a pause signal is input to the controller, quick pause mode determines whether the robot will stop immediately, or will Pause after having executed the motion command.

Immediately decelerating and stopping is referred to as a "Quick Pause".

With the On parameter specified, **QP** turns the Quick Pause mode On.
With the Off parameter specified, **QP** turns the Quick Pause mode Off.

QP displays the current setting of whether the robot arm is to respond to the Pause input by stopping immediately or after the current arm operation is completed. **QP** is simply a status instruction used to display whether Quick Pause mode is on or off.

Notes

Quick pause mode defaults to on after power is turned on:

The Quick Pause mode set by the **QP** instruction remains in effect after the Reset instruction. However, when the PC power or Drive Unit power is turned off and then back on, Quick Pause mode defaults to On.

QP and the Safe Guard Input:

Even if **QP** mode is set to Off, if the Safe Guard Input becomes open the robot will pause immediately.

See Also

Pause, Verinit

QP Statement Example

This Monitor window example displays the current setting of whether the robot arm is to stop immediately on the Pause input. (i.e. is QP mode set On or Off)

```
> qp
QP ON

> qp on  'Sets QP to Quick Pause Mode
>
```

Quit Statement

S

Terminates execution of a specified task or all tasks.

Syntax

Quit { *taskIdentifier* | **All** }

Parameters

taskIdentifier	Task name or integer expression representing the task number. A task name is the function name used in an Xqt statement or a function started from the Run window or Operator window. If an integer expression is used, the range is from 1 to 32.
All	Specifies that all tasks should be terminated.

Description

Quit stops the tasks that are currently being executed, or that have been temporarily suspended with Halt.

See Also

Exit, Halt, Resume, Xqt

Quit Example

This example shows two tasks that are terminated after 10 seconds.

```
Function main
  Xqt winc1   'Start winc1 function
  Xqt winc2   'Start winc2 function
  Wait 10
  Quit winc1  'Terminate task winc1
  Quit winc2  'Terminate task winc2
Fend

Function winc1
  Do
     On 1; Wait 0.2
     Off 1; Wait 0.2
  Loop
Fend

Function winc2
  Do
     On 2; Wait 0.5
     Off 2; Wait 0.5
  Loop
Fend
```

RadToDeg Function

> F

Converts radians to degrees.

Syntax

RadToDeg(*radians***)**

Parameters

radians Real expression representing the radians to convert to degrees.

Return Values

A double value containing the number of degrees.

See Also

ATan, ATan2, DegToRad Function

RadToDeg Function Example

```
s = Cos(RadToDeg(x))
```

Randomize Statement

S

Initializes the random-number generator.

Syntax

Randomize

See Also

Rnd

Randomize Example

```
Function main
    Real r
    Randomize
    Integer randNum

    randNum = Int(Rnd(10)) + 1
    Print "Random number is:", randNum
Fend
```

Range Statement

> S

Specifies and displays the motion limits for each of the servo joints.

Syntax

(1) Range *j1Min, j1Max, j2Min, j2Max, j3Min, j3Max, j4Min, j4Max,*[*j5Min*, *j5Max, j6Min, j6Max*]
(2) Range

Parameters

j1Min	The lower limit for joint 1 specified in pulses.
j1Max	The upper limit for joint 1 specified in pulses.
j2Min	The lower limit for joint 2 specified in pulses.
j2Max	The upper limit for joint 2 specified in pulses.
j3Min	The lower limit for joint 3 specified in pulses.
j3Max	The upper limit for joint 3 specified in pulses.
j4Min	The lower limit for joint 4 specified in pulses.
j4Max	The upper limit for joint 4 specified in pulses.
j5Min	Optional for 6-Axis robots. The lower limit for joint 5 specified in pulses.
j5Max	Optional for 6-Axis robots. The upper limit for joint 5 specified in pulses.
j6Min	Optional for 6-Axis robots. The lower limit for joint 6 specified in pulses.
j6Max	Optional for 6-Axis robots. The upper limit for joint 6 specified in pulses.

Return Values

Displays the current **Range** values when **Range** is entered without parameters

Description

Range specifies the lower and upper limits of each motor joint in pulse counts. These joint limits are specified in pulse units. This allows the user to define a maximum and minimum joint motion range for each of the individual joints. XY coordinate limits can also be set using the XYLim instruction.

The initial **Range** values are different for each robot. The values specified by this instruction remain in effect even after the power is switched off. The Verinit instruction initializes these values to the default values.

When parameters are omitted, the current **Range** values are displayed.

Notes

Range Setting is not Affected by Verinit

The Verinit command does not change the **Range** settings.

Potential Errors

Attempt to Move Out of Acceptable Range

If the robot arm attempts to move through one of the joint limits error an will occur

Axis Does Not Move

If the lower limit pulse is equal to or greater than the upper limit pulse, the joint does not move.

See Also

JRange, Ver, Verinit, XYLim

Range Example

This simple example from the monitor window displays the current range settings and then changes them.

```
> range
-18205, 182045, -82489, 82489, -36864, 0, -46695, 46695
>
> range 0, 32000, 0, 32224, -10000, 0, -40000, 40000
>
```

Read Statement

S

Reads characters from a file or communications port.

Syntax

Read #fileNumber, stringVar$, count

Parameters

fileNumber	File or communications port to read from.
stringVar$	Name of a string variable that will receive the character string.
count	Maximum number of bytes to read.

See Also

ChkCom, ChkNet, OpenCom, OpenNet, ROpen, Seek, UOpen, Write

Read Statement Example

```
Integer numOfChars
String data$

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    Read #1, data$, numOfChars
EndIf
```

ReadBin Statement

S

Reads one byte from a file or communications port.

Syntax

ReadBin #*fileNumber*, *var*

Parameters

fileNumber	File or communications port to read from.
var	Name of a byte, integer, or long variable that will receive the data byte.

See Also

BOpen, Seek, Write, WriteBin

ReadBin Statement Example

```
Integer data

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    ReadBin #1, data
EndIf

Integer f, i, data(100)

f = FreeFile
BOpen "test.dat" As #f
For i = 0 To 100
    ReadBin #f, data(i)
Next i
Close #f
```

Real Statement

S

Declares variables of type Real (4 byte real number).

Syntax

Real *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)]...**]

Parameters

varName	Variable name which the user wants to declare as type **Real**.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

Real is used to declare variables as type **Real**. Local variables should be declared at the top of a function. Global and module variables must be declared outside functions.

See Also

Boolean, Byte, Double, Global, Integer, Long, String

Real Example

The following example shows a simple program which declares some variables using **Real**.

```
Function realtest
  Real var1
  Real A(10)          'Single dimension array of real
  Real B(10, 10)      'Two dimension array of real
  Real C(10, 10, 10) 'Three dimension array of real
  Real arrayVar(10)
  Integer i
  Print "Please enter a Real Number:"
  Input var1
  Print "The Real variable var1 = ", var1
  For i = 1 To 5
    Print "Please enter a Real Number:"
    Input arrayVar(i)
    Print "Value Entered was ", arrayVar(i)
  Next i
Fend
```

Recover Function

F

Executes safeguard position recovery and returns status.

Syntax

Recover

Return Values

Boolean value. True if recover was completed, False if not.

Description

Recover can be used after the safeguard is closed to turn on the robot motors and slowly move the robot back to the position it was in when the safeguard was open. After Recover has completed successfully, you can execute the Cont method to continue the cycle. If Recover was completed successfully, it will return True. Recover will return False if a pause, abort, or safeguard open occurred during recover motion.

See Also

Cont Statement

Recover Function Example

```
Boolean sts
Integer answer

sts = Recover
If sts = True Then
  MsgBox "Ready to continue", MB_ICONQUESTION + MB_YESNO, "MyProject",
answer
  If answer = IDYES Then
    Cont
  EndIf
EndIf
```

Redim Statement

S

Redimension an array at run-time.

Syntax

Redim [**Preserve**] *arrayName* **(***subscripts***)**

Parameters

Preserve	Optional. Specifies to preserve the previous contents of the array. If omitted, the array will be cleared.
arrayName	Name of the array variable; follows standard variable naming conventions. The array must have already been declared.
subscripts	New dimensions of the array variable. You must supply the same number of dimensions as when the variable was declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer expression from 0-2147483646.

Description

Use Redim to change an array's dimensions at run time. Use Preserve to retain previous values.

See Also

Erase, UBound

Redim Statement Example

```
Integer i, numParts, a(0)

Print "Enter number of parts "
Input numParts

Redim a(numParts)

For i=0 to UBound(a)
    a(i) = i
Next

' Redimension the array with 20 more elements
Redim Preserve a(numParts + 20)

' The first element values are retained
For i = 0 to UBound(a)
    Print a(i)
Next
```

Rename Command

>

Renames a file.

Syntax

Rename *oldFileName*, *newFileName*

Parameters

oldFileName	String expression containing the path and name of the file to rename. The filename and extension may contain wildcard characters (*, ?).
newFileName	The new name to be given to the file specified by *oldFileName*.

Description

Changes name of specified file *oldFileName* to *newFileName*.

If path is omitted, **Rename** searches for *oldFileName* in the current directory.

A path cannot be specified for *newFileName*. Therefore, the renamed file exists in the same path as prior to its being renamed.

A file may not be renamed to a filename that already exists in the same path.

Rename allows the use of wildcard characters. Wildcard characters used in filename1 cause **Rename** to rename all corresponding files. For example, to change names of all files named A to B, without changing their filename extensions, use the following:
Rename A.* B.*

Wildcard characters used in *newFileName* cause **Rename** to maintain *oldFileName* characters in their respective positions in *newFileName*. For example, to rename all files corresponding to TEST.* to TEXT.* use the following:
Rename TEST.* ??X?.*

See Also

Copy

Rename Example

Shown below are examples from the monitor window:

```
> Rename A*.* B*.*
> Rename A.PRG B.PRG
```

RenDir Statement

> S

Changes a directory name.

Syntax

Rendir *oldDir*, *newDir*

Parameters

oldDir	A string expression specifying the path and name of the directory to rename.
newDir	A string expression specifying the path and new name to be given to the directory specified by *oldDir*.

Description

The same path used for *oldDir* must be included for *newDir*. A directory may not be renamed to a directory name that already exists in the same path. Wildcard characters are not allowed in either *oldDir* or *newDir*.

See Also

Dir, MkDir

RenDir Command Example

RenDir "c:\mydata", "c:\mydata1"

Reset Statement

> S

Resets the controller into an initialized state.

Syntax

Reset

Description

Reset resets the following items:

Emergency Stop Status
Error status
Output Bits (All Outputs set to Off; User can set Option Switch to turn this feature off)
Current robot Speed, SpeedR, SpeedS (Initialized to default values)
Current robot Accel, AccelR, AccelS (Initialized to default values)
Current robot LimZ parameter (Initialized to 0)
Current robot Fine (Initialized to default values)
Current robot Power Low (Low Power Mode set to On)
Current robot PTPBoost (Initialized to default values)

For servo related errors, Emergency Stop status, and any other conditions requiring a Reset, no command other than Reset will be accepted. In this case first execute Reset, then execute other processing as necessary.

For example, after an emergency stop, first verify safe operating conditions, execute Reset, and then execute Motor On.

Notes

Reset Option Switch

If the "Reset turns off outputs" SPEL Option Switch is on, then when the **Reset** instruction is issued, all outputs will be turned off. This is important to remember when wiring the system such that turning the outputs off should not cause tooling to drop or similar situations. See SPEL Options tab in System Configuration for more details.

Reset is for Current Robot

The **Reset** instruction executes for the current robot. For multi-robot systems, use the Robot instruction to select the robot you want to reset before executing Reset.

See Also

Accel, AccelS, Fine, LimZ, Motor, Off, On, PTPBoost, Robot, SFree, SLock, Speed, SpeedS, Verinit

Reset Statement Example

This shows the Reset instruction issued from the monitor window.

```
>reset
>
```

Restart Statement

> S

Restarts the current program group.

Syntax

Restart

Description

Restart stops all tasks and re-executes the last program group that was running. This is similar to the Chain command, except that the program group is not specified. In addition, **Restart** can also be used in a Trap Emergency or Trap Error routine. This allows you to restart after an emergency stop or critical error.

⚠ WARNING	■ After an EStop has occurred, you must get acknowledgement from the operator using a push-button or message box before restarting the application.

See Also

Chain, Quit, Trap, Xqt

Restart Statement Example

```
Function main

    Trap Emergency Call tEmer

    Motor On
    While TRUE
        Call PickPlac
    Wend
Fend

Function eTrap

   String msg$
   Integer ans
   msg$ = "Estop Is on.  Please clear it and Retry"

   Do While EStopOn
      MsgBox msg$, MB_RETRYCANCEL, "Clear E-stop and click Retry", ans
      If ans = IDRETRY Then
         Reset
      Else
         Exit Function
      EndIf
      Wait 1
   Loop
   ' You must get acknowledgement from the operator before restarting
   MsgBox msg$, MB_YESNO + MB_ICONQUESTION, "OK to Restart?", ans
   If ans = IDYES Then
     Restart
   EndIf
Fend
```

Resume Statement

S

Continues a task which was suspended by the Halt instruction.

Syntax

Resume { *taskIdentifier* | **All** }

Parameters

taskIdentifier	Task name or integer expression representing the task number. A task name is the function name used in an Xqt statement or a function started from the Run window or Operator window. If an integer expression is used, the range is from 1 to 32.
All	Specifies that all tasks should be resumed.

Description

Resume continues the execution of the tasks suspended by the Halt instruction.

See Also

Halt, Quit, Xqt

Resume Statement Example

This shows the use of Resume instruction after the Halt instruction.

```
Function main
   Xqt 2, flicker  'Execute flicker as task 2

   Do
      Wait 3         'Allow flicker to execute for 3 seconds
      Halt flicker   'Halt the flicker task
      Wait 3
      Resume flicker 'Resume the flicker task
   Loop
Fend

Function flicker
   Do
      On 1
      Wait 0.2
      Off 1
      Wait 0.2
   Loop
Fend
```

Return Statement

S

The **Return** statement is used with the GoSub statement. GoSub transfers program control to a subroutine. Once the subroutine is complete, **Return** causes program execution to continue at the line following the GoSub instruction which initiated the subroutine.

Syntax

Return

Description

The **Return** statement is used with the GoSub statement. The primary purpose of the **Return** statement is to return program control back to the instruction following the GoSub instruction which initiated the subroutine in the first place.

The GoSub instruction causes program control to branch to the user specified statement line number or label. The program then executes the statement on that line and continues execution through subsequent line numbers until a **Return** instruction is encountered. The **Return** instruction then causes program control to transfer back to the line which immediately follows the line which initiated the GoSub in the first place. (i.e. the GoSub instruction causes the execution of a subroutine and then execution **Return**s to the statement following the GoSub instruction.)

Potential Errors

Return Found Without GoSub

A **Return** instruction is used to "return" from a subroutine back to the original program which issued the GoSub instruction. If a **Return** instruction is encountered without a GoSub having first been issued then an Error 3 will occur. A stand alone **Return** instruction has no meaning because the system doesn't know where to **Return** to.

See Also

OnErr, GoSub**,** GoTo

Return Statement Example

The following example shows a simple function which uses a GoSub instruction to branch to a label called checkio and check the first 16 user inputs. Then the subroutine returns back to the main program.

```
Function main
     Integer var1, var2
     GoSub checkio
     On 1
     On 2
     End

checkio:  'Subroutine starts here
     var1 = In(0)
     var2 = In(1)
     If var1 <> 0 Or var2 <> 0 Then
          Print "Message to Operator here"
     EndIf
finished:
     Return  'Subroutine ends here and returns to line 40
Fend
```

Right$ Function

F

Returns a substring of the rightmost characters of a string.

Syntax

Right$*(string, count)*

Parameters

string	String variable or character string of up to 255 characters from which the rightmost characters are copied.
count	The number of characters to copy from *string* starting with the rightmost character.

Return Values

Returns a string of the rightmost *count* characters from the character string specified by the user.

Description

Right$ returns the rightmost *count* characters of a string specified by the user. Right$ can return up to as many characters as are in the character string.

See Also

Asc, Chr$, InStr, Left$, Len, Mid$, Space$, Str$, Val

Right$ Example

The example shown below shows a program which takes a part data string as its input and splits out the part number, part name, and part count.

```
Function SplitPartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    PartNum$ = Left$(DataIn$, 10)

    DataIn$ = Right$(datain$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Mid$(DataIn$, 11, 10)

    PartCount = Val(Right$(dataIn$, 5))

Fend


Some other example results from the Right$ instruction from the Monitor
window.
> Print Right$("ABCDEFG", 2)
 FG

> Print Right$("ABC", 3)
 ABC
```

RmDir Statement

> S

Removes an empty subdirectory from a controller disk drive.

Syntax

Rmdir *dirName*

Parameters

dirName String expression for the path and name of the directory to remove.

Description

Removes the specified subdirectory. Prior to executing **Rmdir** all of the subdirectory's files must be deleted.

The current directory or parent directory cannot be removed.

When executed from the Monitor window, quotes may be omitted.

Rmdir Example

Shown below is a simple example from the monitor window:

```
> RmDir \mydata
```

Rnd Function

F

Return a random number.

Syntax

Rnd(*maxValue*)

Parameters

maxValue Real expression that represents the maximum return value.

Return Values

Random real number from 0 to *range*.

Description

Use Rnd to generate random number values.

See Also

Int, Randomize

Rnd Function Example

Here's a Rnd example that generates a random number between 1 and 10.

```
Function main
    Real r
    Integer randNum

    Randomize
    randNum = Int(Rnd(9)) + 1
    Print "Random number is:", randNum
Fend
```

Robot Statement

S

Selects the current robot.

Syntax

Robot *number*

Parameters

number Number of the desired robot. The value ranges from 1 to the number of installed robots.

Description

Robot allows the user to select the default robot for subsequent motion instructions.

On a system with one robot, the Robot statement does not need to be used.

See Also

Accel, AccelS, Arm, ArmSet, Go, Hofs, Home, HOrdr, HTest, Local, Move, Pulse, Robot Function, Speed, SpeedS

Robot Example

```
Function main
    Integer I
    For I = 1 to 100
        Robot 1
        Go P(i)
        Robot 2
        Go P(i)
    Next I
Fend
```

Robot Function

F

Returns the current robot number.

Syntax

Robot

Return Values

Integer containing the current robot number.

See Also

Robot Statement

Robot Function Example

```
Print "The current robot is: ", Robot
```

RobotModel$ Function

F

Returns the robot model name.

Syntax

RobotModel$ [(*robotNumber*)]

Parameters

robotNumber	Optional. An integer expression representing a robot number. If omitted, the current robot number is used.

Return Values

A string containing the model name. This is the name that is shown on the rear panel of the robot.

See Also

RobotType

RobotModel$ Example

```
Print "The robot model is ", RobotModel$
```

RobotType Function

F

Returns the robot type.

Syntax

RobotType [(*robotNumber*)]

Parameters

robotNumber	Optional. An integer expression representing a robot number. If omitted, the current robot number is used.

Return Values

1: JOINT

2: CARTESIAN

3: SCARA

4: CYLINDRICAL

5: 6-AXIS

See Also

RobotModel$

RobotType Example

```
If RobotType = 2 Then
  Print "Robot type is Cartesian"
EndIf
```

ROpen Statement

S

Opens a file for reading.

Syntax

ROpen *fileName* **As** *#fileNumber*

Parameters

fileName	A string expression containing the file name to read from. The drive and path can also be included.
fileNumber	Integer expression from 30 - 63 which will be used as a handle for the file.

Description

Opens the specified *fileName* for reading and identifies it by the specified *fileNumber*. This statement is used to open and read data from the specified file. Close closes the file and releases the file number.

The specified filename must be the name of an existing file on the PC.

The *fileNumber* identifies the file as long as the file is open. It is used by the Input, Line Input, and Read statements for reading and for closing. Accordingly, until the current file is closed, its file number cannot be used to specify a different file.

It is recommended that you use the FreeFile function to obtain the file number so that more than one task are not using the same number.

See Also

AOpen, BOpen, Close, Input #, Line Input #, Print #, Read, UOpen, WOpen

ROpen Statement Example

Shown below is a simple example to which opens a file, writes some data to it, then later opens the same file and reads the data into an array variable.

```
Real data(200)
Integer fileNum, i

For i = 0 To 100
    data(i) = i
Next i

fileNum = FreeFile
WOpen "TEST.VAL" As #fileNum
For i = 0 To 100
    Print #fileNum , data(i)
Next i
Close #fileNum

fileNum = FreeFile
ROpen "TEST.VAL" As #fileNum
For i = 0 to 100
    Input #fileNum , data(i)
Next i
Close #fileNum
```

RSet$ Function

F

Returns the specified string with leading spaces added up to the specified length..

Syntax

RSet$ (*string, length*)

Parameters

string	String expression.
length	Integer expression for the total length of the string returned.

Return Values

Specified string with leading spaces appended.

See Also

LSet$, Space$

RSet$ Function Example

```
str$ = "123"
str$ = RSet$(str$, 10)   ' str$ = "       123"
```

RShift Function

F

Shifts numeric data to the right by a user specified number of bits.

Syntax

RShift*(number, shiftBits)*

Parameters

number	Numeric expression to be shifted.
shiftBits	The number of bits to shift *number* to the right.

Return Values

Returns a numeric result which is equal to the value of *number* after shifting right *shiftbits* number of bits.

Description

RShift shifts the specified numeric data (*number*) to the right (toward a lower order digit) by the specified number of bits (*shiftBits*). The high order bits shifted are replaced by 0.

The simplest explanation for **RShift** is that it simply returns the result of *number* / $2^{shiftBits}$. (*Number* is divided by 2 *shiftBit* times.)

Notes

Numeric Data Type:

The numeric data (*number*) may be any valid numeric data type. **RShift** works with data types: Byte, Integer, and Real.

See Also

And, LShift, Not, Or, Xor

RShift Example

The example shown below shows a program which shows all the possible **RShift** values for an Integer data type starting with the integer set to 0.

```
Function rshiftst
  Integer num, snum, i
  num = 32767
  For i = 1 to 16
     Print "i =", i
     snum = RShift(num, 1)
     Print "RShift(32767, ", i, ") = ", snum
  Next i
Fend
```

Some other example results from the **RShift** instruction from the monitor window.

```
> Print RShift(10,1)
5
> Print RShift(8,3)
1
> Print RShift(16,2)
4
```

RTrim$ Function

F

Returns a string equal to specified string without trailing spaces.

Syntax

RTrim$(*string*)

Parameters

string String expression.

Return Values

Specified string with trailing spaces removed.

See Also

LTrim$, Trim$

RTrim$ Function Example

```
str$ = "  data  "
str$ = RTrim$(str$)   ' str$ = "..data"
```

RunDialog Statement

S

Runs an EPSON RC+ dialog from a SPEL+ program.

Syntax

RunDialog *dialogID*

Parameters

dialogID Integer expression containing a valid dialog ID. These values are predefined constants as shown below.

DLG_JOG	100	Run Jog and Teach dialog
DLG_IOMON	102	Run I/O Monitor
DLG_ROBOTPANEL	104	Run Robot Control Panel
DLG_VGUIDE	110	Run Vision Guide dialog

Description

Use RunDialog to run EPSON RC+ dialogs from a SPEL+ task. The task will be suspended until the operator closes the dialog.

When running dialogs that execute robot commands, you should ensure that no other tasks will be controlling the robot while the dialog is displayed, otherwise errors could occur.

See Also

InputBox, MsgBox

RunDialog Example

```
If Motor = Off Then
    RunDialog DLG_ROBOTPANEL
    If Motor = Off Then
        Print "Motors are off, aborting program"
        Quit All
    EndIf
EndIf
```

SafetyOn Function

F

Returns the current Safety status.

Syntax

SafetyOn

Return Values

True if the Safety circuit is Open (On), otherwise False.

Description

SafetyOn returns True after the safety circuit is Open.

SafetyOn is equivalent to `((Stat(0) And &H400000)<>0)`.

See Also

EStopOn, PauseOn, Stat

SafetyOn Function Example

```
If SafetyOn Then
    Print "Safe Guard is open."
EndIf
```

SavePoints Statement

> S

Saves point data in main memory to a disk file for the current robot.

Syntax

SavePoints *fileName*

Parameters

fileName String expression containing the file into which points will be stored. The extension must be .PNT.

Description

SavePoints saves points for the current robot to the specified file in the current project directory. A .PNT extension must always be specified. The **SavePoints** command will also add the point file to the project for the current robot if it did not already exist.

Potential Errors

Out of Disk Space

If there is no space remaining an error 60 will be issued.

Point file for another robot.

If *fileName* is a point file for another robot, an error will occur

A Path Cannot be Specified

If *fileName* contains a path, an error will occur. Only a file name in the current project can be specified.

Bad File name

If a file name is entered which has spaces in the name, or other bad file name characteristics an error will be issued.

See Also

ImportPoints, LoadPoints

SavePoints Statement Example

```
Clear
For i = 1 To 10
   P(i) = i, 100, 0, 0
Next i
SavePoints "TEST.PNT"
```

Seek Statement

S

Changes position of file pointer for a specified file.

Syntax

Seek #*fileNumber*, *pointer*

Parameters

fileNumber	Integer expression for the file number to perform the seek operation.
pointer	Integer expression for the desired position to seek, starting from 0 to the length of the file.

See Also

AOpen, BOpen, Read, ROpen, UOpen, Write

Seek Statement Example

```
Integer fileNumber
String data$

fileNumber = FreeFile
UOpen "test.txt" As #fileNumber
Seek #fileNumber, 20
Read #fileNumber, data$, 2
Close #fileNumber
```

Select...Send

S

Executes one of several groups of statements, depending on the value of an expression.

Syntax

Select *selectExpr*
Case *caseExpr*
statements
[Case *caseExpr*
statements **]**
[Default
statements **]**
Send

Parameters

selectExpr	Any numeric or string expression.
caseExpr	Any numeric or string expression that evaluates to the same type as *selectExpr*.
statements	One or more valid SPEL$^+$ statements or multi-statements.

Description

If any one *caseExpr* is equivalent to *selectExpr*, then the statements after the Case statement are executed. After execution, program control transfers to the statement following the Send statement.

If no *caseExpr* is equivalent to *selectExpr*, the Default statements are executed and program control transfers to the statement following the Send statement.

If no *caseExpr* is equivalent to *selectExpr* and Default is omitted, nothing is executed and program control transfers to the statement immediately following the Send statement.

selectExpr and *caseExpr* may include constants, variables, and logical operators that use And, Or and Xor.

Up to 250 Case items are allowed with one **Select** instruction. Nesting of **Select…Send** statements is supported up to 256 levels deep including other statements (**Do…Loop**, **If…Then…Else…EndIf**, and **While…Wend**).

See Also

If...Then...Else

Select Example

Shown below is a simple example for Select...Send:

```
Function Main
  Integer I
  For I = 0 To 10
    Select I
      Case 0
           Off 1;On 2;Jump P1
      Case 3
           On 1;Off 2
           Jump P2;Move P3;On 3
      Case 7
           On 4
      Default
           On 7
    Send
  Next
Fend
```

Sense Statement

> S

Specifies and displays input condition that, if satisfied, completes the Jump in progress by stopping the robot above the target position.

Syntax

Sense [*inputCondition*]

Parameters

inputcondition This condition expression checks the status of inputs and must return a Boolean value. The following functions and operators may be used in the *inputCondition*:

Functions: In, InW, MemIn, MemSw, Sw, Force

Operators: And, Or, Xor, +, *

Other: Parenthesis for prioritizing operations, and variables

Description

Sense is used to stop approach motion during a Jump, Jump3, and Jump3CP instructions. The **Sense** condition must include at lease one input or memory I/O input function.

When variables are included in the **Sense** condition, their values are computed during **Sense** execution. Multiple **Sense** statements are permitted. The most recent **Sense** condition remains current until superseded with another **Sense** statement.

Jump, Jump3, Jump3CP with Sense Modifier

Checks if the current **Sense** condition is satisfied. If satisfied, the Jump instruction completes with the robot stopped above the target position. (i.e. When the **Sense** Condition is True, the robot arm remains just above the target position without executing approach motion. When the **Sense** condition is False, the robot arm completes the full Jump instruction motion through to the target position.

When parameters are omitted, the current Sense definition is displayed.

Notes

Sense Setting at Main Power On

At power on, the **Sense** condition is:

```
Sense Sw(0) = On      'Input bit 0 is on
```

Use of JS and Stat to Verify Sense

Use JS or Stat to verify if the Sense condition has been satisfied.

See Also

In, Js, Jump, Jump3, Jump3CP, MemIn, MemSw, Stat, Sw

Sense Statement Example

This is a simple example on the usage of the Sense instruction.

```
Function test
   .
   .
TrySense:
    Sense Sw(1) = 0   'Specifies the arm stops
                      'above the target when
                      'the input bit 1 is 0.
    Jump P1 C2 Sense
    'If the arm remains stationary at the point specified by
    'Js(0)=1, then execute ERRPRC and go to TrySense.
    If Js(0) = 1 Then
         GoSub ERRPRC
         GoTo TrySense
    EndIf
    On 1; Wait 0.2; Off 1
   .
   .
Fend
```

<Other Syntax Examples>

```
> Sense Sw(1)=1 And Sw($1)=1

> Sense Sw(0) Or Sw(1) And Sw($1)
```

SetCom Statement

S

Sets parameters for a communications port.

Syntax

SetCom #*portNumber,* [*baud*], [*dataBits*], [*stopBits*] , [*parity*] , [*terminator*] , [*HWFlow*], [*SWFlow*] , [*timeOut*]

Parameters

portNumber	Specifies which RS232 port to set parameters for. Valid values are 1-16.
baud	Optional. Specifies the baud rate. Valid values are: 110 2400 19200 115200 300 4800 38400 600 9600 56000 1200 14400 57600 (Default: 9600)
dataBits	Optional. Specifies the number of data bits per character. Valid values are **7** and **8**.
stopBits	Optional. Specifies the number of stop bits per character. Valid values are **1** and **2**.
parity	Optional. Specifies the parity. Valid values are **O** (Odd), **E** (Even), and **N** (None).
terminator	Optional. Specifies the line termination characters. Valid values are **CR**, **LF**, **CRLF**.
HWFlow	Optional. Specifies hardware control. Valid values are **RTS** and **NONE**.
SWFlow	Optional. Specifies software control. Valid values are **XON** and **NONE**.
timeOut	Optional. Specifies the maximum time for transmit or receive in seconds. If this value is 0, then there is no time out.

Notes

SetCom replaces Config

On the SRC 3xx series controllers, the Config statement was used to configure the robot controller serial ports. Config is no longer used. When a project from a previous version using Config is opened, the Config statements are converted to SetCom statements.

See Also

OpenCom, CloseCom, SetNet

SetCom Example

```
SetCom #1, 9600, 8, 1, N, CRLF, NONE, NONE, 0

SetCom #2, 4800
```

SetNet Statement

S

Sets parameters for a TCP/IP port.

Syntax

SetNet #*portNumber, hostAddress, TCP_IP_PortNum, terminator, SWFlow, timeOut*

Parameters

portNumber	Specifies which port to set parameters for. Valid values are 128 - 147.
hostAddress	Specifies the host IP address.
TCP_IP_PortNum	Specifies the TCP/IP port number for this node.
terminator	Specifies the line termination characters. Valid values are **CR**, **LF**, **CRLF**.
SWFlow	Specifies software control. Valid value is **NONE**.
timeOut	Specifies the maximum time for transmit or receive in seconds. If this value is 0, then there is no time out.

See Also

OpenCom, CloseCom, SetCom

SetNet Example

```
SetNet #128, "192.168.0.1", 1, CRLF, NONE, 0
```

SFree Statement

> S

Removes servo power from the specified servo axis.

Syntax

SFree *jointNumber* [*, jointNumber,...*]

Parameters

jointNumber An integer expression representing a servo joint number.

Description

SFree removes servo power from the specified servo joints. This instruction is used for the direct teaching or the part installation by partially de-energizing a specific joint. To re-engage a joint execute the SLock instruction or Motor On.

Notes

SFree Sets Some System Items back to Their Initial State:

SFree, for safety purposes, initializes parameters concerning the robot arm speed (Speed and SpeedS), acceleration (Accel and AccelS) and the LimZ parameter.

Notes

SFree and Its Use with the Z Joint for SCARA and Cartesian robots

The Z joint has electronic brakes so setting SFree for the Z joint does not immediately allow the Z joint to be moved. To move the Z joint by hand requires the brake to be released continuously by pressing the brake release switch on the top of the robot arm.

SFree and Its Use with 6-Axis robots

All joints of the 6-axis robots have an electromagnetic brake. The brake can be released using the Brake command with the motor off. You cannot move any joint by hand using SFree.

Executing motion commands while joints are in SFree state

Attempting to execute a motion command while in the SFree condition will cause an error in the controller's default state. However, to allow motion while 1 or more of the axes are in the SFree state, turn on the "Allow Motion with one or more axes free" option switch. (This switch can be set from the SPEL Option tab on the System Configuration dialog from the Setup Menu in EPSON RC+.)

See Also

Brake, LimZ, Motor, SFree Function, SLock

SFree Statement Example

This is a simple example on the usage of the SFree instruction. The SPEL Option Motion with SFree must be enabled for this example to work.

```
Function GoPick
   Speed pickSpeed
   'Release the excitation of J1 and J2,
   'and control the Z and U joints for part installation.
   SFree 1, 2
   Go pick
   'Restore the excitation of J1 and J2.
   SLock 1, 2
Fend
```

SFree Function

F

Returns sfree status for a specified joint.

Syntax

SFree(*jointNumber*)

Parameters

jointNumber Integer expression representing the joint number to check.

Return Values

True if the joint is free, False if not.

See Also

SFree Statement

SetFree Example

```
If SFree(1) Then
    Print "Joint 1 is free"
EndIf
```

Sgn Function

F

Determines the sign of the operand.

Syntax

Sgn(*Operand*)

Parameters

Operand A numeric expression.

Return Values

1: If the operand is a positive value.
0: If the operand is a 0
-1: If the operand is a negative value.

Description

The **Sgn** function determines the sign of the numeric value of the operand.

See Also

Abs, And, Atan, Atan2, Cos, Int, Mod, Or, Not, Sin, Sqr, Str$, Tan, Val, XOR

Sgn Function Example

This is a simple monitor window example on the usage of the Sgn function.

```
>print sgn(123)
 1
>print sgn(-123)
 -1
>
```

ShutDown Statement

S

Shuts down EPSON RC+ and optionally shuts down or restarts Windows.

Syntax

ShutDown [*mode*]

Parameters

mode Optional. An integer expression that represents the mode setting described below.

Symbolic constant	Value	Meaning
Mode ommitted		Displays a dialog allowing the user to choose the shutdown option.
SHUTDOWN_ALL	0	Shuts down EPSON RC+ and Windows.
SHUTDOWN_RESTART	1	Shuts down EPSON RC+ and restarts Windows.
SHUTDOWN_EPSONRC	2	Shuts down EPSON RC+.

Description

ShutDown allows you to exit the current EPSON RC+ session from a SPEL$^+$ program.

See Also

Restart

ShutDown Statement Example

```
' Exit EPSON RC+ and Windows
ShutDown 0
```

Signal Statement

S

Send a signal to tasks executing WaitSig.

Syntax

Signal *signalNumber*

Parameters

signalNumber Signal number to transmit. Range is 0 to 127.

Description

Signal can be used to synchronize multi-task execution.

Previous signals issued before WaitSig is executed are ignored.

See Also

WaitSig

Signal Statement Example

```
Function Main
   Xqt 2, SubTask
   Call InitSys
   Signal 1

Fend

Function SubTask
   WaitSig 1

Fend
```

Sin Function

F

Returns the sine of a numeric expression.

Syntax

Sin(*radians***)**

Parameters

radians Real expression in Radians.

Return Values

Numeric value representing the sine of the numeric expression *radians*.

Description

Sin returns the sine of the numeric expression. The numeric expression (*radians)* must be in radian units. The value returned by the **Sin** function will range from -1 to 1

To convert from radians to degrees, use the RadToDeg function.

See Also

Abs, Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sqr, Str$, Tan, Val

Sin Function Example

The following example shows a simple program which uses **Sin**.

```
Function sintest
    Real x
    Print "Please enter a value in radians:"
    Input x
    Print "Sin of ", x, " is ", Sin(x)
Fend
```

SLock Statement

> S

Restores servo power from servo free condition for the specified servo axis.

Syntax

SLock *jointNumber* [*, jointNumber,...*]

Parameters

jointNumber The servo joint number.

Description

SLock restores servo power to the specified servo joint, which was de-energized by the SFree instruction for the direct teaching or part installation.

If the joint number is omitted, all joints are engaged.

Engaging the 3rd joint (Z) causes the brake to release.

To engage all axes, Motor On may be used instead of **SLock**.

Executing **SLock** while in Motor Off state will cause an error.

Notes

SLock Sets Some System Items back to Their Initial State:

SLock, for safety purposes, initializes parameters concerning the robot arm feed speed (Speed and SpeedS), acceleration (Accel and AccelS) and the LimZ parameter.

See Also

Brake, LimZ, Reset, SFree

SLock Example

This is a simple example on the usage of the SLock instruction. The SPEL Option Motion with SFree must be enabled for this example to work.

```
Function test
   .
   .
   .
   'Release the excitation of J1 and J2,
   'and control the Z and U joints for part installation.
   SFree 1, 2
   Go P1
   'Restore the excitation of J1 and J2.
    SLock 1, 2
   .
   .
   .
Fend
```

Space$ Function

F

Returns a string of space characters.

Syntax

Space$(*count*)

Parameters

count The number of spaces to put in the return string.

Return Values

Returns a string of *count* space characters.

Description

Space$ returns a string of *count* space characters as specified by the user. **Space$** can return up to 255 characters (the maximum number of characters allowed in a string variable).

The **Space**$ instruction is normally used to insert spaces before, after, or between other strings of characters.

See Also

Asc, Chr$, InStr, Left$, Len, LSet$, Mid$, Right$, RSet$, Str$, Val

Space$ Function Example

```
> Print "XYZ" + Space$(1) + "ABC"
XYZ ABC

> Print Space$(3) + "ABC"
   ABC
>
```

Speed Statement

> S

Specifies or displays the arm speed for the point to point motion instructions Go, Jump and Pulse.

Syntax

(1) Speed *percent*, [*departSpeed*], [*approSpeed*]
(2) Speed

Parameters

percent	Integer expression between 1-100 representing the arm speed as a percentage of the maximum speed.
departSpeed	Integer expression between 1-100 representing the depart motion speed for the Jump instruction.
approSpeed	Integer expression between 1-100 representing the approach motion speed for the Jump instruction.

Return Values

Displays current **Speed** value when used without parameters.

Description

Speed specifies the arm speed for all point to point motion instructions. This includes motion caused by the Go, Jump and Pulse robot motion instructions. The speed is specified as a percentage of maximum speed with the range of acceptable values between 1-100. (1 represents 1% of the maximum speed and 100 represents 100% of maximum speed). Speed 100 represents the maximum speed possible.

Depart and approach speed values apply only to the Jump instruction. If omitted, each defaults to the *percent* value.

The speed value initializes to its default value when any one of the following is performed:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Abort All button or Ctrl+C Key

In Low Power Mode, the effective speed setting is lower than the default value. If a higher speed is specified directly (from the monitor window) or in a program, the speed is set to the default value. In High Power Mode, the motion speed setting is the value specified with **Speed**.

If higher speed motion is required, set high power mode using Power High and close the safety door. If the safety door is open, the **Speed** settings will be changed to their default value.

If **Speed** is executed when the robot is in low power mode, the following message is displayed. The following example shows that the robot will move at the default speed (5) because it is in Low Power Mode even though the speed setting value by **Speed** is 80.

```
> speed 80
> speed
Low Power Mode
    80
    80     80
>
```

See Also

Accel, Go, Jump, Power, Pass, Pulse, SpeedS

Speed Statement Example

Speed can be used from the monitor window or in a program. Shown below are simple examples of both methods.

```
Function speedtst
    Integer slow, fast, i
    slow = 10
    fast = 100
    For i = 1 To 10
        Speed slow
        Go P0
        Go P1
        Speed fast
        Go P0
        Go P1
    Next i
Fend
```

From the monitor window the user can also set Speed values.

```
> Speed 100,100,50    'Z joint downward speed set to 50
> Speed 50
> Speed
    Low Power State: Speed is limited to 5
       50
       50        50
>
```

Speed Function

F

Returns one of the three speed settings.

Syntax

Speed(*paramNumber*)

Parameters

paramNumber Integer expression which evaluates to one of the values shown below.

1: PTP motion speed

2: Jump depart speed

3: Jump approach speed

Return Values

Integer value from 1 to 100.

See Also

Speed Statement

Speed Function Example

```
        Integer savSpeed

        savSpeed = Speed(1)
        Speed 50
        Go pick
        Speed savSpeed
    Fend
```

SpeedR Statement

> S

Sets or displays the tool rotation speed for CP motion when ROT is used.

Syntax

(1) SpeedR *rotSpeed*
(2) SpeedR

Parameters

rotSpeed Real expression in degrees / second.

Return Values

When parameters are omitted, the current SpeedR setting is displayed.

Description

SpeedR is effective when the ROT modifier is used in the Move, Arc, Arc3, BMove, TMove, and Jump3CP motion commands.

The **SpeedR** value initializes to the default value (low speed) when any one of the following conditions occurs:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Abort All button or Ctrl + C Key

See Also

AccelR, Arc, Arc3, BMove, Jump3CP, Power, SpeedR Function, TMove

SpeedR Statement Example

```
SpeedR 200
```

SpeedR Function

> F

Returns tool rotation speed value.

Syntax

SpeedR

Return Values

Real value in degrees / second

See Also

AccelR Statement, SpeedR Statement

SpeedR Function Example

```
Real currSpeedR

currSpeedR = SpeedR
```

SpeedS Statement

> S

Specifies or displays the arm speed for use with the continuous path motion instructions such as Move, Arc, Arc3, Jump3, and Jump3CP.

Syntax

(1) SpeedS *speed*, [*departSpeed*], [*approSpeed*]
(2) SpeedS

Parameters

speed	Real expression representing the CP motion speed in units of mm/sec.
departSpeed	Optional. Real expression representing the Jump3 depart speed in units of mm/sec.
approSpeed	Optional. Real expression representing the Jump3 approach speed in units of mm/sec.

Return Values

Displays current **SpeedS** value when used without parameters.

Description

SpeedS specifies the tool center point speed for use with all the continuous path motion instructions. This includes motion caused by the Move and Arc instructions.

SpeedS is specified in mm/Sec which represents a Tool Center Point velocity for the robot arm. Valid entries for **SpeedS** range is 1-2000 for 6-axis robots and 1-1120 for other robots. The default value varies from robot to robot. See the robot manual for the default SpeedS values for your robot model. This is the initial **SpeedS** value set up automatically by the controller each time main power is turned on.

The **SpeedS** value initializes to its default value when any one of the following is performed:

Power On
Software Reset
Motor On
SFree, SLock
Verinit
Abort All button or Ctrl + C Key

In Low Power Mode, the effective **SpeedS** setting is lower than the default value. If a higher speed is specified directly (from the monitor window) or in a program, the speed is set to the default value. In High Power Mode, the motion **SpeedS** setting is the value of **SpeedS**.

If higher speed motion is required, set high power mode using Power High and close the safety door. If the safety door is open, the **SpeedS** settings will be changed to their default value.

If **SpeedS** is executed when the robot is in low power mode, the following message is displayed. The following example shows that the robot will move at the default speed (50) because it is in Low Power Mode even though the speed setting value by **SpeedS** is 800.

```
> SpeedS 800
Low Power State: SpeedS is limited to 50
>
> SpeedS
Low Power State: SpeedS is limited to 50
  800
>
```

See Also

AccelS, Arc, Jump3, Move, Speed

SpeedS Example

SpeedS can be used from the monitor window or in a program. Shown below are simple examples of both methods.

```
Function speedtst
    Integer slow, fast, i
    slow = 50
    fast = 500
    For i = 1 To 10
        SpeedS slow
        Go P0
        Move P1
        SpeedS fast
        Go P0
        Move P1
    Next i
Fend
```

From the monitor window the user can also set **SpeedS** values.

```
> speeds 1000
> speeds 500
> speed 30      'set point to point speed
> go p0         'point to point move
> speeds 100    'set straight line speed in mm/Sec
> move P1       'move in straight line
```

SpeedS Function

F

Returns the current SpeedS setting.

Syntax

SpeedS [(*paramNumber*)]

Parameters

paramNumber Optional. Integer expression specifying which SpeedS value to return.
1: CP speed
2: Jump3 depart speed
3: Jump3 approach speed

Return Values

Real number, in mm/sec

See Also

SpeedS Statement

SpeedS Example

```
Real savSpeeds

savSpeeds = SpeedS

Print "Jump3 depart speed = ", SpeedS(2)
```

SPELCom_Event Statement

> S

Generates a user event for a VB Guide SPELCom control used in a host program.

Syntax

SPELCom_Event *eventNumber* [*, msgArg1, msgArg2, msgArg3,...*]

Parameters

eventNumber	An integer expression whose value is from 1000 - 32767.
msgArg1, msgArg2, msgArg3...	Optional. Each message argument can be either a number, string literal, or a variable name.

Description

This instruction makes it easy to send real time information to another application using the SPELCom ActiveX control provided in the VB Guide option. For example, you can update parts count, lot number, etc. by sending an event to your host program.

Note

This command will only work if the VB Guide option is installed.

See Also

VB Guide Manual

SPELCom_Event Example

In this example, a SPEL$^+$ task sends cycle data to the host program.

```
Function RunParts
    Integer cycNum

    cycNum = 0
    ' Main loop
    While True
        ...
        ...
        cycNum = cycNum + 1
        Spelcom_Event 3000, cycNum, lot$, cycTime
    Wend
Fend
```

See Also

VB Guide Manual

SPELCom_Return Statement

> S

Returns a value from a SPEL function to the Call method of the VB Guide SPELCom control.

Syntax

SPELCom_Return *returnValue*

Parameters

returnValue	An integer expression that is returned from the SPELCom Call method.

Description

SPELCom_Return is a convenient way to return a value to the SPELCom host application. If **SPELCom_Return** is executed without being called from SPELCom, the command will be ignored and no error will occur.

Note

This command will only work if the VB Guide option is installed.

SPELCom_Return Example

In this example, a VB program calls the SPEL+ function *GetPart*, which returns an error number.

```
'VB code
Dim sts As Integer
sts = SPELCom1.Call("GetPart")


'SPEL+ code
Function GetPart
    Integer errNum

    OnErr GPErr
    errNum = 0
    Jump P1
    On vacuum
    Wait Sw(vacOn) = 1, 2
    If TW = 1 Then
        errNum = VAC_TIMEOUT
    Endif
GPExit:
    SPELCom_Return errNum
    Exit Function
GPErr:
    errNum = Err
    GoTo GPExit
Fend
```

Sqr Function

F

Computes the non-negative square root value of the operand.

Syntax

Sqr(*Operand*)

Parameters

Operand A real expression.

Return Values

Square root value.

Description

The Sqr function returns the non-negative square root value of the operand.

Potential Errors

Negative operand

If the operand is or has a negative numeric value, an error will occur.

See Also

Abs, And, Atan, Atan2, Cos, Int, Mod, Not, Or, Sgn, Sin, Str$, Tan, Val, Xor

Sqr Function Example

This is a simple Monitor window example on the usage of the Sqr function.

```
>print sqr(2)
 1.414214
>
```

The following example shows a simple program which uses **Sqr**.

```
Function sqrtest
    Real x
    Print "Please enter a numeric value:"
    Input x
    Print "The Square Root of ", x, " is ", Sqr(x)
Fend
```

Stat Function

F

Returns the execution status information of the controller.

Syntax

Stat*(address)*

Parameters

address Defines which status bits to check.

Return Values

Returns a 4 byte value that presents the status of the controller. Refer to table below.

Description

The **Stat** instruction returns information as shown in the table below:

Address	Bit		Controller Status Indicated When Bit is On
0	0 to 15	&H0 to &H8000	Task 1 is being executed (Xqt) or in Halt State to Task 16 is being executed (Xqt) or in Halt State
	16	&H10000	Task(s) is being executed
	17	&H20000	Pause condition
	18	&H40000	Error Condition
	19	&H80000	Teach mode
	20	&H100000	Emergency Stop Condition
	21	&H200000	Low Power Mode (Power Low)
	22	&H400000	Safe Guard Input is Closed
	23	&H800000	Enable Switch is Open
	24	&H1000000	Recover Required
	25	&H2000000	Recover In Cycle
	26-31		Undefined
1	0	&H0	Log of Stop above target position upon satisfaction of condition in Jump...Sense statement. (This log is erased when another Jump statement is executed).
	1	&H1	Log of stop at intermediate travel position upon satisfaction of condition in Go/Jump/Move...Till statement. (This log is erased when another Go/Jump/Move...Till statement is executed
	2	&H2	Log of execution of Mcal command/statement
	3	&H4	Log of stop at intermediate travel position upon satisfaction of condition in Trap statement
	4	&H8	Motor On mode
	5	&H20	Current position is home position
	6	&H40	Low power state
	7	&H80	Undefined
	8	&H100	4th Joint Motor is On
	9	&H200	3rd Joint Motor is On
	10	&H400	2nd Joint Motor is On
	11	&H800	1st Joint Motor is On
	12	&H1000	6th Joint Motor is On
	13	&H2000	5th Joint Motor is On
	14-31		Undefined
2	0 to 15	&H0 to &H8000	Task 17 is being executed (Xqt) or in Halt State to Task 32 is being executed (Xqt) or in Halt State

See Also

EStopOn Function, TillOn Function, PauseOn Function, SafetyOn Function

Stat Example

```
Function StatDemo

    rbt1_sts = RShift((Stat(0) And &H070000), 16)
    Select TRUE
        Case (rbt1_sts And &H01) = 1
            Print "Tasks are running"
        Case (rbt1_sts And &H02) = 2
            Print "Pause Output is ON"
        Case (rbt1_sts And &H04) = 4
            Print "Error Output is ON"
    Send
Fend
```

Str$ Function

F

Converts a numeric value to a string and returns it.

Syntax

Str$*(number)*

Parameters

number Integer or real expression.

Return Values

Returns a string representation of the numeric value.

Description

Str$ converts a number to a string. Any number up to 18 digits (positive or negative) is valid.

See Also

Abs, Asc, Chr$, InStr, Int, Left$, Len, Mid$, Mod, Right$, Sgn, Space$, Val

Str$ Function Example

The example shown below shows a program which coverts several different numbers to strings and then prints them to the screen.

```
Function strtest
    Integer intvar
    Real realvar
    '
    intvar = -32767
    Print "intvar = ", Str$(intvar)
    '
    realvar = 567.9987
    Print "realvar = ", Str$(realvar)
    '
Fend
```

Some other example results from the Str$ instruction from the monitor window.

```
> Print Str$(99999999999999)
 1.000000E+014

> Print Str$(25.999)
 25.999
```

String Statement

S

Declares variables of type String. (Character-string variables)

Syntax

String *varName$* [**(***subscripts***)**] [, *varName$* [**(***subscripts***)**]**...**]

Parameters

varName$	Variable name which the user wants to declare as type String.
subscripts	Optional. Dimensions of an array variable; up to 3 multiple dimensions may be declared. The syntax is as follows **(***dim1,* [*dim2*], [*dim3*]**)** *dim1*, *dim2*, *dim3* can be an integer number from 0-2147483646.

Description

The **String** statement is used to declare variables of type String. String variables can contain up to 255 characters. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

String Operators

The following operators can be used to manipulate string variables:

+ Merges character strings together. Can be used in the assignment statements for string variables or in the Print instruction.

```
Example: name$ = fname$ + " " + lname$
```

= Compares character strings. True is returned only when the two strings are exactly equal, including case.

```
Example: If temp1$ = "A" Then GoSub test
```

< > Compares character strings. True is returned when one or more characters in the two strings are different.

```
Example: If temp1$ <> "A" Then GoSub test
```

Notes

Variable Names Must Include "$" Character:

Variables of type String must have the character "$" as the last character in the variable name.

See Also

Boolean, Byte, Double, Global, Integer, Long, Real

String Example

```
String password$
String A$(10)         'Single dimension array of string
String B$(10, 10)     'Two dimension array of string
String C$(10, 10, 10) 'Three dimension array of string

Print "Enter password:"
Input password$
If UCase$(password$) = "EPSON" Then
    Call RunMaintenance
Else
    Print "Password invalid!"
EndIf
```

Sw Function

F

Returns or displays the selected input port status. (i.e. Discrete User I/O)

Syntax

Sw(*bitNumber***)**

Parameters

bitNumber Integer expression representing one of the standard or expansion discrete hardware inputs.

Return Values

Returns a 1 when the specified input is On and a 0 when the specified input is Off.

Description

Sw provides a status check for hardware inputs. **Sw** is most commonly used to check the status of one of the inputs which could be connected to a feeder, conveyor, gripper solenoid, or a host of other devices which works via discrete I/O. Obviously the input checked with the **Sw** instruction has 2 states (1 or 0). These indicate whether the device is On or Off.

See Also

In, InBCD, MemOn, MemOff, MemSw, Off, On, OpBCD, Oport, Out, Wait

Sw Function Example

The example shown below simply checks the discrete input #5 and branches accordingly. On is used instead of 1 for more clarity.

```
Function main
    Integer i, feed5Ready
    feed5Ready = Sw(5)
    'Check if feeder is ready
    If feed5Ready = On Then
        Call mkpart1
    Else
        Print "Feeder #5 is not ready. Please reset and"
        Print "then restart program"
    EndIf
Fend
```

Other simple examples are as follows from the monitor window:

```
> print sw(5)
1
>
```

Sw($) Function

F

Note: The Sw($) function is obsolete and has been replaced by the MemSw function from Epson RC+ 4.0.

Returns or displays the selected memory status bit.

Syntax

Sw(*$bitNumber***)**

Parameters

bitNumber Number between 0-511 representing one of the 512 memory I/O bits. NOTE: The "$" must be put in front of the bit number to indicate that this is memory I/O.

Return Values

Returns a 1 when the specified bit is On and a 0 when the specified bit is Off.

Description

Sw $ provides an internal memory I/O capability. Sw $ is most commonly used as a status variable which has 2 states (1 or 0) which could mean On/Off, True/False, Finished/Not Finished etc.. Valid entries for Sw $ range from bit 0 to bit 511. The On $ and Off $ instructions are normally used with Sw $. On $ turns the specified bit on and Off $ turns the specified bit Off.

Notes

Difference between Sw and Sw $

It is very important for the user to understand the difference between the Sw and Sw $ instructions. The Sw $ instruction works with internal memory I/O and does not affect hardware I/O of the system in any way. The Sw instruction works with the hardware input channels located on the rear of the controller. These HARDWARE channels are discrete Inputs which interact with devices external to the controller.

See Also

In, In$, InBCD, Off, Off$, On, On$, OpBCD, Oport, Out, Out$, Sw, Wait

Sw$ Example

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. Sw $ is used in combination with the Wait instruction to wait until the memory I/O #1 is the proper value before it is safe to move again.

```
Function main
    Integer I
    Off $1
    Xqt 2, task2
    For I = 1 to 100
        Wait Sw($1) = 0
        Go P(i)
        On $1
    Next I
Fend

Function task2
    Integer I
    For I = 101 to 200
        Wait Sw($1) = 1
        Go P(i)
        Off $1
    Next I
Fend
```

Other simple examples from the Monitor window are as follows:

```
> on $1
> print sw($1)
1
> off $1
> print sw($1)
0
```

SyncLock Statement

S

Synchronizes tasks using a mutual exclusion lock.

Syntax

SyncLock *syncID* [, *timeOut*]

Parameters

syncID	Integer expression representing signal number to receive. Range is from 1 to 32.
timeOut	Optional. Real expression representing the maximum time to wait for lock.

Description

Use **SyncLock** to lock use of a common resource so that only one task at a time can use it. When the task is finished with the resource, it must call SyncUnlock to release the lock so other tasks can use it.

A task can only unlock a syncID that it previously locked.

A task must execute SyncUnlock to release the lock. If the task is quit, then no other task can use the lock until all tasks are aborted.

If the *timeOut* parameter is used, then the **Tw** function must be used to check if the lock was successful.

See Also

Signal, SyncLock, Tw, Wait, WaitPos

SyncLock Example

The following example uses SyncLock and SyncUnlock to allow only one task at a time to write a message to a log file.

```
Function Main

    Xqt Func1
    Xqt Func2
Fend

Function Func1
  Long count
  Do
    Wait .5
    count = count + 1
    LogMsg "Msg from Func1, " + Str$(count)
  Loop
Fend

Function Func2
  Long count
  Do
    Wait .5
    count = count + 1
    LogMsg "Msg from Func2, " + Str$(count)
  Loop
Fend



Function LogMsg(msg$ As String)
  SyncLock 1
  AOpen "msg.txt" As #30
  Print #30, msg$
  Close #30
  SyncUnlock 1
Fend
```

The following example uses SyncLock with optional time out. Tw is used to check if the lock was successful. By using a timeout, you can execute other code periodically while waiting to lock a resource.

```
Function MySyncLock(syncID As Integer)
  Do
    SyncLock syncID, .5
    If Tw = 0 Then
      Exit Function
    EndIf
    If Sw(1) = On Then
      Off 1
    EndIf
  Loop
Fend
```

SyncUnlock Statement

S

Unlocks a sync ID that was previously locked with SyncLock.

Syntax

SyncUnlock *syncID*

Parameters

syncID	Integer expression representing signal number to receive. Range is from 1 to 32.

Description

Use **SyncUnlock** to unlock a sync ID previously locked with SyncLock.

A task can only unlock a syncID that it previously locked.

See Also

Signal, SyncLock, Wait, WaitPos

SyncUnlock Example

```
Function Main

    Xqt task
    Xqt task
    Xqt task
    Xqt task
Fend

Function task
  Do
    SyncLock 1
    Print "resource 1 is locked by task", MyTask
    Wait .5
    SyncUnlock 1
  Loop
Fend
```

Tab$ Function

F

Returns a string containing the specified number of tabs characters.

Syntax

Tab$*(number)*

Parameters

number Integer expression representing the number of tabs.

Return Values

String containing tab characters.

Description

Tab$ returns a string containing the specified number of tabs.

See Also

Left$, Mid$, Right$, Space$

Tab$ Function Example

```
Print "X", Tab$(1), "Y"
Print
For i = 1 To 10
    Print x(i), Tab$(1), y(i)
Next i
```

Tan Function

F

Returns the tangent of a numeric expression.

Syntax

Tan(*radians***)**

Parameters

radians Real expression given in radians.

Return Values

Real number containing the tangent of the parameter *radians*.

Description

Tan returns the Tangent of the numeric expression. The numeric expression (*radians)* may be any numeric value as long as it is expressed in radian units.

To convert from radians to degrees, use the RadToDeg function.

See Also

Abs, Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sin, Sqr, Str$, Val

Tan Function Example

```
Function tantest
    Real num
    Print "Enter number in radians to calculate tangent for:"
    Input num
    Print "The tangent of ", num, "is ", Tan(num)
Fend
```

The examples shown below show some typical results using the Tan instruction from the Monitor window.

```
> print tan(0)
0.00
> print tan(45)
1.6197751905439
>
```

TargetOK Function

F

Returns a status indicating whether or not the PTP (Point to Point) motion from the current position to a target position is possible.

Syntax

TargetOK(*targetPos***)**

Parameters

targetPos Point expression for the target position.

Return Values

True if is it possible to move to the target position from the current position, otherwise False.

Description

Use **TargetOK** to verify that a target position and orientation can be reached before actually moving to it. The motion trajectory to the target point is not considered.

See Also

CurPos, FindPos, InPos, WaitPos

TargetOK Function Example

```
If TargetOK(P1) Then
  Go P1
EndIf

If TargetOK(P10 /L /F) Then
  Go P10 /L /F
EndIf
```

TaskDone Function

F

Returns the completion status of a task.

Syntax

TaskDone (*taskIdentifier*)

Parameters

taskIdentifier	Task name or integer expression representing the task number. A task name is the function name used in an Xqt statement or a function started from the Run window or Operator window. If an integer expression is used, the range is from 1 to 32.

Return Values

True if the task has been completed, False if not.

Description

Use TaskDone to determine if a task has completed.

See Also

TaskState, TaskWait

TaskDone Function Example

```
Xqt 2, conveyor
Do
    .
    .
Loop Until TaskDone(conveyor)
```

TaskState Function

F

Returns the current state of a task.

Syntax

TaskState(*taskIdentifier*)

Parameters

taskIdentifier	Task name or integer expression representing the task number. A task name is the function name used in an Xqt statement or a function started from the Run window or Operator window. If an integer expression is used, the range is from 1 to 32.

Return Values

0: Task not running
1: Task is running
2: Task is waiting for an event
3: Task has been halted
4: Task has been paused in QuickPause
5: Task in error condition
6: Task is executing Wait statement

Description

Use TaskState to get status for a given task. You can specify task number or task name.

See Also

TaskDone, TaskWait

TaskState Function Example

```
If TaskState(conveyor) = 0 Then
    Xqt 2, conveyor
EndIf
```

TaskWait Statement

S

Waits to for a task to terminate.

Syntax

TaskWait (*taskIdentifier*)

Parameters

taskIdentifier	Task name or integer expression representing the task number. A task name is the function name used in an Xqt statement or a function started from the Run window or Operator window. If an integer expression is used, the range is from 1 to 32.

See Also

TaskDone, TaskState

TaskWait Statement Example

```
Xqt 2, conveyor
TaskWait conveyor
```

TCPSpeed Function

F

Returns the calculated current tool center point (TCP) speed.

Syntax

TCPSpeed

Return Values

Real value containing the calculated current tool center point speed in mm/second.

Description

Use **TCPSpeed** to get the calculated current speed of the tool center point in mm/second when executing a CP (Continuous Path) motion command. CP motion commands include Move, TMove, Arc, Arc3, CVMove, and Jump3CP. This is not the actual tool center point speed. It is the speed that the system has calculated for the tool center point at the time the function is called.

The motor compliance lag is excluded from the calculation.
If the robot is executing a PTP (Point to Point) motion command, this function returns 0.

See Also

AccelS, CurPos, InPos, SpeedS

TCPSpeed Function Example

```
Function MoveTest
  AccelS 4000, 4000
  SpeedS 200
  Xqt ShowTCPSpeed
  Do
    Move P1
    Move P2
  Loop
Fend

Function ShowTCPSpeed
  Do
    Print "Current TCP speed is: ", TCPSpeed
    Wait .1
  Loop
Fend
```

TGo Statement

> S

Executes Point to Point relative motion, in the current tool coordinate system.

Syntax

TGo *destination* [**CP**] [*searchExpr*] [*!...!*]

Parameters

destination	The target destination of the motion using a point expression.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to execute I/O and other commands during motion.

Description

Executes point to point relative motion in the current tool coordinate system.

Arm orientation attributes specified in the *destination* point expression are ignored. The manipulator keeps the current arm orientation attributes. However, for a 6-Axis manipulator, the arm orientation attributes are automatically changed in such a way that joint travel distance is as small as possible.

The Till modifier is used to complete TGo by decelerating and stopping the robot at an intermediate travel position if the current Till condition is satisfied.

The Find modifier is used to store a point in FindPos when the Find condition becomes true during motion.

When Till is used and the Till condition is satisfied, the manipulator halts immediately and the motion command is finished. If the Till condition is not satisfied, the manipulator moves to the destination point.

When Find is used and the Find condition is satisfied, the current position is stored. Please refer to Find for details.

When parallel processing is used, other processing can be executed in parallel with the motion command.

CP parameters can start the acceleration of the next motion command when the deceleration starts. In this case the track will not come to the destination coordinate and moves to the next point.

See Also

Accel, Find, !....! Parallel Processing, Point Assignment, Speed, Till, TMove, Tool

TGo Example

```
> TGo XY(100, 0, 0)  'Move 100mm in X direction
                     '(in the tool coordinate system)
Function TGoTest

  Speed 50
  Accel 50, 50
  Power High

  Tool 0
  P1 = XY(300, 300, -20, 0)
  P2 = XY(300, 300, -20, 0) /L

  Go P1
  Print P*
  TGo XY(0, 0, -30, 0)
  Print P*

  Go P2
  Print P*
  TGo XY(0, 0, -30, 0)
  Print P*

Fend
```

```
[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  300.000 Z:  -50.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /L /0
 X:  300.000 Y:  300.000 Z:  -50.000 U     0.000 V:    0.000 W     0.000 /L /0
```

Till Statement

> S

Specifies and displays input condition that, if satisfied, completes the motion command (Jump, Go, Move, etc.) in progress by decelerating and stopping the robot at an intermediate position.

Syntax

Till [*inputCondition*]

Parameters

inputCondition This condition expression checks the status of inputs and must return a Boolean value. The following functions and operators may be used in the *inputCondition*:

Functions: In, InW, MemIn, MemSw, Sw, Force

Operators: And, Or, Xor, +, *

Other: Parenthesis for prioritizing operations, and variables

Description

The **Till** statement must include at least 1 input or memory I/O input function. Include in the **Till** condition only the operators described above in the parameters section. (Using any other operator will result, not in an error, but in unpredictable motion)

The **Till** statement can be used on a separate line or as a search expression in a motion command statement.

When variables are included, their values are computed during the **Till** execution. Multiple **Till** statements are permitted. The most recent **Till** condition remains current until superseded.

When parameters are omitted, the current Till definition is displayed.

Notes

Till Setting at Main Power On

At power on, the **Till** condition is initialized to **Till** Sw(0) = On.

Use of Stat or TillOn to Verify Till

After executing a motion command which uses the **Till** qualifier there may be cases where you want to verify whether or not the **Till** condition was satisfied. This can be done through using the Stat function or the TillOn function.

See Also

Find, Go, In, InW, Jump, MemIn, MemSw, Move, Stat, Sw, TillOn

Till Example

Shown below are some sample lines from programs using the Till instruction

```
Till Sw(1) = 0   'Specifies Till condition (Input bit 1 off)
Go P1 Till       'Stop if previous line condition is satisfied
Till Sw(1) = 1 And Sw($1) = 1   'Specify new Till condition
Move P2 Till     'Stop if previous line condition satisfied
Move P5 Till Sw(10) = 1         'Stop if condition on this line
                                'is satisfied
```

TillOn Function

F

Returns the current Till status.

Syntax

TillOn

Return Values

True if the Till condition occurred in the previous motion command using Till.

Description

TillOn returns True if Till condition occurred.

TillOn is equivalent to `((Stat(1) And 2) <> 0)`.

See Also

EStopOn, SafetyOn, Sense, Stat, Till

TillOn Function Example

```
Go P0 Till Sw(1) = On
If TillOn Then
    Print "Till condition occurred during move to P0"
EndIf
```

Time Command

>

Specifies and displays the current time.

Syntax

(1) Time *hours, minutes, seconds*
(2) Time

Parameters

hours	The hour of the day to set the controller clock to. This is an integer expression between 1 and 24.
minutes	The minute of the day to set the controller clock to. This is an integer expression between 0 and 59.
seconds	The second of the day to set the controller clock to. This is an integer expression between 0 and 59.

Return Values

If parameters are omitted, displays the current time in 24 hour format.

Description

Specifies the current time.

The time specification is in 24 hour format.

See Also

Date, GetTime, GetDate

Time Example

```
> Time
The current time is 10:15:32

> Time 1,5,0
> Time
The current time is 1:05:15
```

Time Function

F

Returns the controller accumulated operating time.

Syntax

Time(*unitSelect*)

Parameters

unitSelect An integer number ranging from 0-2. This integer specifies which unit of time the controller returns:

0: hours

1: minutes

2: seconds

Description

Returns the controller accumulated operating time as an integer.

See Also

Hour

Time Function Example

Shown below are a few examples from the monitor window:

```
Function main
    Integer h, m, s

    h = Time(0)   'Store the time in hours
    m = Time(1)   'Store the time in minutes
    s = Time(2)   'Store the time in seconds
    Print "This controller has been used:"
    Print h, "hours, ",
    Print m, "minutes, ",
    Print s, "seconds"
Fend
```

Time$ Function

F

Returns the system time.

Syntax

Time$

Return Values

A string containing the current time in 24 hour format *hh:mm:ss*.

Description

Time$ is used to get the system time in a program statement. To set the system time, you must use the Time command from the monitor window.

See Also

Date, Date$, GetDate, GetTime, Time

Time$ Example

```
Function LogErr

   Integer errNum

   errNum = Err(0)
   AOpen "errlog.dat" As #30
   Print #30, "Error" , errNum, ErrMsg$(errNum), " ", Date$, " ", Time$
   Close #30
   EClr
Fend
```

TLClr Statement

> S

Clears (undefines) a tool coordinate system.

Syntax

TLClr *toolNumber*

Parameters

toolNumber	Integer expression representing which of the 3 tools to clear (undefine). (Tool 0 is the default tool and cannot be cleared.)

See Also

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLSet, Verinit

TLClr Example

```
TLClr 1
```

TLSet Statement

> S

Defines or displays a tool coordinate system.

Syntax

(1) TLSet *toolNum*, *toolDefPoint*
(2) TLSet

Parameters

toolNum	Integer number from 1-3 representing which of 3 tools to define. (Tool 0 is the default tool and cannot be modified.)
toolDefPoint	**P***number* or **P**(*expr*) or point label or point expression.

Return Values

When parameters are omitted, displays current **TLSet** Definition.

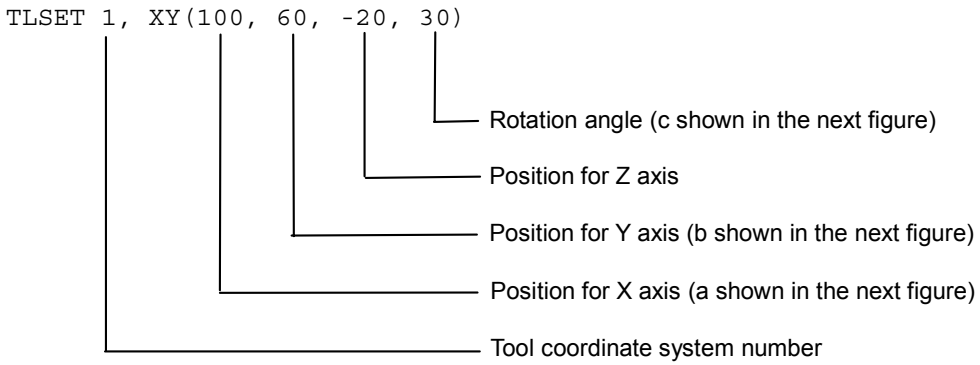
Description

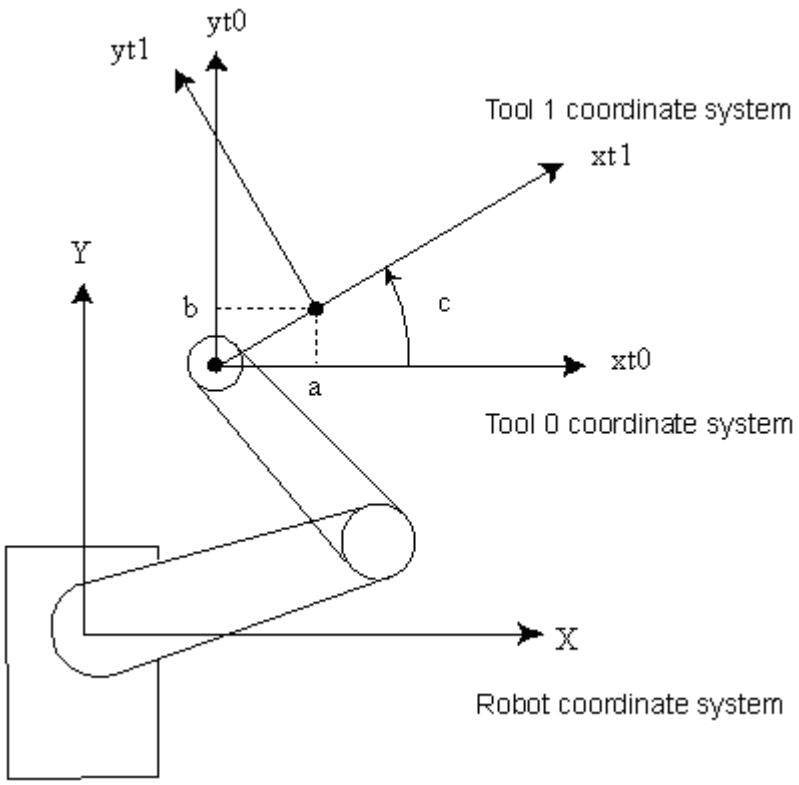
Defines the tool coordinate systems Tool 1, Tool 2 or Tool 3 by specifying tool coordinate system origin and rotation angle in relation to the Tool 0 coordinate system (Hand coordinate system).

```
TLSet 1, XY(50,100,-20,30)

TLSet 2, P10 +X(20)
```

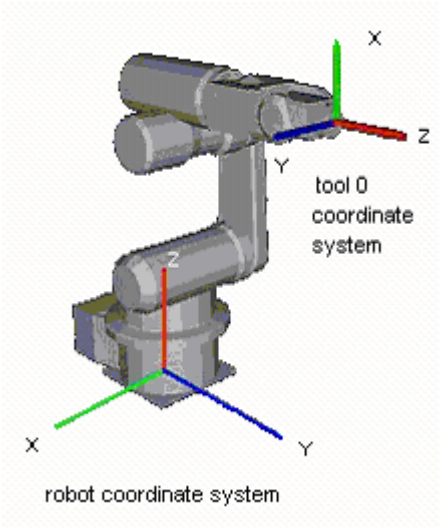
In this case, the coordinate values of P10 are referenced and 20 is added to the X value. Arm attribute and local coordinate system numbers are ignored.



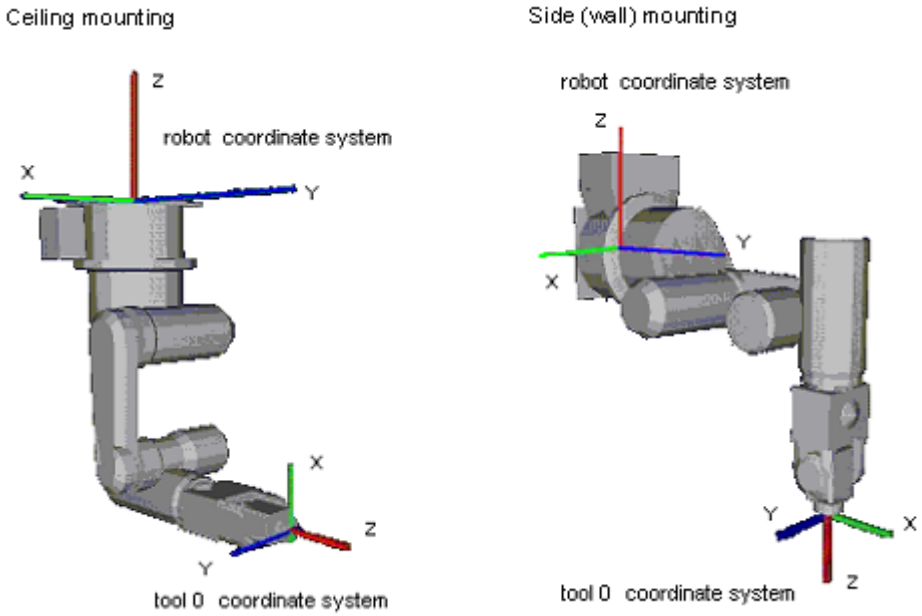


TISet for 6-Axis robots

The origin of Tool 0 is the flange side of the sixth joint. When all joints are at the 0 degree position, the Tool 0 coordinate system's X axis is aligned with the robot coordinate system's Z axis, the Y axis is aligned with the robot coordinate system's X axis, and the Z axis is perpendicular to the flange face, and is aligned with the robot coordinate system's Y axis, as shown in the figure below:



Tool 0 coordinate systems are defined for ceiling and wall mounted robots as shown in the figures below.



Notes

TLSet values are maintained

The **TLSet** values are preserved for each robot. Use TLClr to clear a tool definition. Verinit also clears all tool definitions.

See Also

Tool, Arm, ArmSet, Point Expression, TLClr, TLSet Function

TLSet Example

The example shown below shows a good test which can be done from the monitor window to help understand the difference between moving when a tool is defined and when no tool is defined.

```
> TLSet 1, XY(100, 0, 0, 0)  'Define tool coordinate system for
                             'Tool 1 (plus 100 mm in x direction
                             'from hand coordinate system)
> Tool 1  'Selects Tool 1 as defined by TLSet
> TGo P1  'Positions the Tool 1 tip position at P1
> Tool 0  'Tells robot to use no tool for future motion
> Go P1   'Positions the center of the U-Joint at P1
```

TLSet Function

F

Returns a point containing the tool definition for the specified tool.

Syntax

TLSet(*toolNumber*)

Parameters

toolNumber Integer expression representing the number of the tool to retrieve.

Return Values

A point containing the tool definition.

See Also

TLSet Statement

TLSet Function Example

```
P1 = TLSet(1)
```

TMOut Statement

> S

Specifies the number of seconds to wait for the condition specified with the Wait instruction to come true before issuing a timeout error.

Syntax

TMOut *seconds*

Parameters

seconds Integer expression representing the number of seconds until a timeout occurs. Valid range is 0-32767 seconds in 1 second intervals.

Description

TMOut sets the amount of time to wait (when using the Wait instruction) until a timeout error (Error 94) is issued. If a timeout of 0 seconds is specified, then the timeout is effectively turned off. In this case the Wait instruction waits indefinitely for the specified condition to be satisfied.

The default initial value for **TMOut** is 0.

See Also

In, MemSw, OnErr, Sw, TW, Wait

TMOut Example

```
TMOut 5
Wait MemSw(0) = On
If TW Then
    Print "Time out occurred"
EndIf
```

TMove Statement

> S

Executes linear interpolation relative motion, in the current tool coordinate system

Syntax

TMove *destination* [**ROT**] [**CP**] [*searchExpr*] [*!...!*]

Parameters

destination	The target destination of the motion using a point expression.
ROT	Optional. :Decides the speed/acceleration/deceleration in favor of tool rotation.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Till or Find expression. **Till** | **Find** **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to execute I/O and other commands during motion.

Description

Executes linear interpolated relative motion in the current tool coordinate system.

Arm orientation attributes specified in the *destination* point expression are ignored. The manipulator keeps the current arm orientation attributes. However, for a 6-Axis manipulator, the arm orientation attributes are automatically changed in such a way that joint travel distance is as small as possible.

TMove uses the SpeedS speed value and AccelS acceleration and deceleration values. Refer to *Using TMove with CP* below on the relation between the speed/acceleration and the acceleration/deceleration. If, however, the ROT modifier parameter is used, **TMove** uses the SpeedR speed value and AccelR acceleration and deceleration values. In this case SpeedS speed value and AccelS acceleration and deceleration value have no effect.

Usually, when the move distance is 0 and only the tool orientation is changed, an error will occur. However, by using the ROT parameter and giving priority to the acceleration and the deceleration of the tool rotation, it is possible to move without an error. When there is not an orientational change with the ROT modifier parameter and movement distance is not 0, an error will occur.

Also, when the tool rotation is large as compared to move distance, and when the rotation speed exceeds the specified speed of the manipulator, an error will occur. In this case, please reduce the speed or append the ROT modifier parameter to give priority to the rotational speed/acceleration/deceleration.

The Till modifier is used to complete TMove by decelerating and stopping the robot at an intermediate travel position if the current Till condition is satisfied.

The Find modifier is used to store a point in FindPos when the Find condition becomes true during motion.

When Till is used and the Till condition is satisfied, the manipulator halts immediately and the motion command is finished. If the Till condition is not satisfied, the manipulator moves to the destination point.

When Find is used and the Find condition is satisfied, the current position is stored. Please refer to Find for details.

When parallel processing is used, other processing can be executed in parallel with the motion command.

Notes

Using TMove with CP

The CP parameter causes the arm to move to *destination* without decelerating or stopping at the point defined by *destination*. This is done to allow the user to string a series of motion instructions together to cause the arm to move along a continuous path while maintaining a specified speed throughout all the motion. The **TMove** instruction without CP always causes the arm to decelerate to a stop prior to reaching the point *destination*.

See Also

AccelS, Find, !....! Parallel Processing, Point Assignment, SpeedS, TGo, Till, Tool

TMove Example

```
> TMove XY(100, 0, 0, 0)    'Move 100mm in the X
                            'direction (in the tool coordinate system)
Function TMoveTest

  Speed 50
  Accel 50, 50
  SpeedS 100
  AccelS 1000, 1000
  Power High

  Tool 0
  P1 = XY(300, 300, -20, 0)
  P2 = XY(300, 300, -20, 0) /L

  Go P1
  Print P*
  TMove XY(0, 0, -30, 0)
  Print P*

  Go P2
  Print P*
  TMove XY(0, 0, -30, 0)
  Print P*

Fend

[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  300.000 Z:  -50.000 U     0.000 V:    0.000 W     0.000 /N /0
 X:  300.000 Y:  300.000 Z:  -20.000 U     0.000 V:    0.000 W     0.000 /L /0
 X:  300.000 Y:  300.000 Z:  -50.000 U     0.000 V:    0.000 W     0.000 /L /0
```

Tmr Function

F

Timer function which returns the amount of time in seconds which has elapsed since the timer was started.

Syntax

Tmr(*timerNumber*)

Parameters

timerNumber Integer expression representing which of the 64 timers to check the time of.

Return Values

Elapsed time for the specified timer as a real number in seconds. Timer range is from 0 - approx. 1.7E+31. Timer resolution is 0.000001 seconds.

Description

Returns elapsed time in seconds since the timer specified was started. 64 timers are available numbered 0 - 63.

Timers are reset with TmReset.

SPEL+ timers use the PC CPU's high-resolution performance counter. When evaluating very small time intervals, keep in mind that there is a small overhead in retrieving the timer value from your SPEL+ program. You can calculate the overhead by calling Tmr immediately after calling TmReset. Since the overhead is very small, in the order of 10 microseconds, it is normally not required to take it into account.

```
Real overhead

TmReset 0
overHead = Tmr(0)
```

See Also

TmReset

Tmr Function Example

```
TmReset 0           'Reset Timer 0
For I=1 To 10       'Perform operation 10 times
    GoSub Cycle
Next
Print Tmr(0) / 10   'Calculate and display cycle time
```

TmReset Statement

S

Resets the timers used by the Tmr function.

Syntax

TmReset *timerNumber*

Parameters

timerNumber Integer expression from 0 - 63 specifies which of the 64 timers to reset.

Description

Resets and starts the timer specified by *timerNumber*. 64 timers are available, numbered from 0 - 63.

Use the Tmr function to retrieve the elapsed time for a specific timer.

See Also

Tmr

TmReset Example

```
TmReset 0           'Reset Timer 0
For I=1 To 10       'Perform operation 10 times
    GoSub CYL
Next
Print Tmr(0)/10     'Calculate and display cycle time
```

Tool Statement

> S

Selects or displays the current tool.

Syntax

(1) Tool *toolNumber*
(2) Tool

Parameters

toolNumber Optional. Integer expression from 0-3 representing which of 4 tool definitions to use with subsequent motion instructions.

Return Values

Displays current **Tool** when used without parameters.

Description

Tool selects the tool specified by the tool number (toolNum). When the tool number is 0, no tool is selected and all motions are done with respect to the center of the end effector joint. However, when Tool entry 1, 2, or 3 is selected motion is done with respect to the end of the tool as defined with the tool definition.

Note

Power Off and Its Effect on the Tool Selection

Turning main power off does not change the tool coordinate system selection. However, executing the Verinit instruction initializes the tool coordinate system selection to the standard tool (Tool 0).

See Also

TGo, TLSet, TMove

Tool Statement Example

The example shown below shows a good test which can be done from the monitor window to help understand the difference between moving when a tool is defined and when no tool is defined.

```
>tlset 1, 100, 0, 0, 0   'Define tool coordinate system for
                         'Tool 1 (plus 100 mm in x direction
                         'from hand coordinate system)
>tool 1     'Selects Tool 1 as defined by TLSet
>tgo p1     'Positions the Tool 1 tip position at P1
>tool 0     'Tells robot to use no tool for future motion
>go p1      'Positions the center of the U-Joint at P1
```

Tool Function

F

Returns the current tool number.

Syntax

Tool

Return Values

Integer containing the current tool number.

See Also

Tool Statement

Tool Function Example

```
Integer savTool

savTool = Tool
Tool 2
Go P1
Tool savTool
```

Trap Statement

S

Defines interrupts and what should happen when they occur.

Syntax

(1) **Trap** *trapNumber*, *condition* **GoTo** {*linenum* | *label*}
Trap *trapNumber*, *condition* **GoSub** {*linenum* | *label*}
Trap *trapNumber*, *condition* **Call** *funcname*
Trap *trapNumber*

(2) **Trap Emergency Call** *funcName*
Trap Error Call *funcName*
Trap Pause Call *funcName*
Trap SGOpen Call *funcName*
Trap SGClose Call *funcName*
Trap Abort Call *funcName*

Parameters

trapNumber	Integer number from 1-4 representing which of 4 Trap numbers to use. (SPEL$^+$ supports up to 4 active Trap interrupts at the same time.)
condition	This condition is a boolean expression. The following functions and operators may be used in the *condition* expression: **Functions:** Ctr, Cnv_QueLen, MemIn, MemSw, In, InW, Oport, Out, OutW, Sw **Operators:** And, Or, Xor, +, * **Other:** Parenthesis for prioritizing operations, and variables
lineNumber	The line number where program execution is to be transferred when the GoTo or GoSub instructions are used with Trap and the Trap condition is satisfied.
label	The label where program execution is to be transferred when the GoTo or GoSub instructions are used with Trap and the Trap condition is satisfied.
funcName	The function that is called when Call is used with the Trap instructions and the Trap condition is satisfied.

Description

There are basically two types of traps. One type uses syntax 1 and is for hardware or memory I/O conditions, and the other type uses syntax 2 and is for several system conditions.

When a trap occurs, you must re-arm it again by executing a trap statement at the end of the trap function. If you re-arm the trap before the function ends and the trap condition occurs before the first trap function had completed, an error will occur.

Syntax 1

Executes interrupt process which is specified by GoTo, GoSub, or Call when the specified condition is satisfied.
Once the interrupt process is executed, its **Trap** setting is cleared. If the same interrupt process is necessary, the **Trap** instruction must define it again.

If the trap condition is satisfied while another function by the Call instruction is in execution, the interrupt process by GoTo, GoSub in the **Trap** setting is not executed.

To cancel a **Trap** setting simply execute the **Trap** instruction with only the *trapNumber* parameter. e.g. "Trap 3" cancels Trap #3.

If GoTo is specified

The command being executed will be processed as described below, then control branches to the specified line number or label.

- Any arm motion will pause immediately
- Waiting status by the Wait or Input commands will discontinue
- All other commands will complete execution before control branches

If GoSub is specified

After executing the same process as GoTo, branches to the specified line number or label and then executes the subroutine that follows. Once the Return statement at the end of the subroutine is executed, program execution returns to the line following the GoSub.

GoSub and Call are not allowed in subroutine of trap process.

If Call is specified

Program control executes the specified function. In this case, the task which executes the Trap command will keep on operating.

Syntax 2

When Emergency Stop, Safe Guard open or close, Error, or Pause condition exists, the Trap process function by Call is executed as a privileged task.

Do not use Xqt in the trap process function.

To cancel a Trap setting execute the Trap instruction with only the keyword parameter. For example, "Trap Emergency" cancels the emergency stop trap.

If Emergency is specified

- When the Emergency Stop is activated, program control executes the Trap process function after finishing the task shutdown process. This means that the trap routine is the last code to execute after all tasks have been stopped. However, a Restart command is available that can be used in the trap emergency handler to restart the current group. You can also use Chain to start a specified group.
- If I/O status is configured to be reset when E-Stop is detected, On, Off, and Out commands for I/O in the Trap process function cannot be executed.
- If I/O status is configured to be preserved when E-Stop is detected, all I/O commands are available. This feature is set or disabled from one of the Option Switches. To configure this go to the SPEL Options tab on the System Configuration dialog accessible from the Setup Menu.

If Error is specified

- When Error (including the error which occurred inside the system) is issued, the system executes the Trap process function after finishing the task shutdown process. This means that the trap routine is the last code to execute after all tasks have been stopped. However, a Restart command is available that can be used in the trap emergency handler to restart the current group. You can also use Chain to start a specified group.

- Only errors that cause a task to shut down are caught. For example, if a task has no error handler and an error occurs, the error trap function will be called. This also includes system errors.

- There are three optional parameters for the Trap Error user function: *errorNumber*, *robotNumber*, and *jointNumber*. If you want to use these values, supply up to three byval integer parameters to the trap function.

For example:

```
Function trapError(errNum As Integer, robotNum As Integer, jointNum
As Integer)
  Print "error number = ", errNum
  Print "robot number = ", robotNum
  Print "joint number = ", jointNum
  If Ert = 0 Then
    Print "system error"
  Else
    Print "task error"
    Print "function = ", Erf$(Ert)
    Print "line number = ", Erl(Ert)
  EndIf
Fend
```

If Pause is specified

- When a Pause occurs while a program is being executed, the Trap process function is executed after the Pause status is processed. Pause can occur from a Pause statement, Safe Guard open, or Remote Pause.

If SGOpen is specified

- When safe guard is opened while a program is running, executes trap process function after pause status is processed.

If SGClose is specified

- When safe guard is closed after a pause caused by the opened safe guard while a program is running, executes trap process function.

If Abort is specified

- When an Abort All occurs from the Run Window or Stop from the Operator Window, then this trap function will be called. This function should be short and should not contain a continuous loop. If a second abort all occurs during this trap, then all tasks will be aborted and the trap will not be called.

See Also

Call, Era, Erl, Err, Ert, ErrMsg$, GoSub, GoTo, OnErr

Trap Example

<Example 1> Error process defined by User

Sw(0) Input is regarded as an error input defined by user.

```
Function Main
    Trap 1 Sw(0)= On GoTo EHandle  'Defines Trap
    .
    .
    .
EHandle:
    On 31   'Signal tower lights
    OpenCom #1
    Print #1, "Error is issued"
    CloseCom #1
Fend
```

<Example 2> Usage like multi-tasking

```
Function Main
    Trap 2 MemSw(0) = On Or MemSw(1) = On Call Feeder
    .
    .
    .
Fend
.
Function Feeder
  Select TRUE
    Case MemSw(0) = On
      MemOff 0
      On 2
    Case MemSw(1) = On
      MemOff 1
      On 3
  Send

  ' Re-arm the trap for next cycle
  Trap 2 MemSw(0) = On Or MemSw(1) = On Call Feeder
Fend
```

<Example 3> Emergency Stop Handler

If input Sw(31) is turned on, Wait status is discontinued, and execution branches to trap process subroutine.

```
Function Main
    Trap Emergency Call EstopHandler
    .
    Print "Starting main"
    Do
       Print "Main is running"
       Wait 1
    Loop
Fend

Function EstopHandler

   String msg$
   Integer ans
   msg$ = "Estop Is on.  Please clear it and Retry"

   Do While EStopOn
      MsgBox msg$, MB_RETRYCANCEL, "Clear E-stop and click Retry", ans
      If ans = IDRETRY Then
         Reset
      Else
         Exit Function
      EndIf
      Wait 1
   Loop
   ' You must get acknowledgement from the operator before restarting
   MsgBox msg$, MB_YESNO + MB_ICONQUESTION, "Ready to Restart?", ans
   If ans = IDYES Then
     Restart
   EndIf
Fend
```

<Example 4> Critical Error Handler

```
Function Main
    Trap Error Call CriticalHandler
    Print "Starting main"
    Do
       Wait 1
       ' No error handler in this function
       ' So next line will fire the
       ' error trap
       Print 1 / 0
    Loop
    Exit Function
Fend

Function CriticalHandler(errNum As Integer)
  Print "Critical error ", errNum, " occurred"
  Print "Restarting program"
  Restart
Fend
```

Trim$ Function

F

Returns a string equal to specified string without leading or trailing spaces.

Syntax

Trim$(*string*)

Parameters

string String expression.

Return Values

Specified string with leading and trailing spaces removed.

See Also

LTrim$, RTrim$

Trim$ Function Example

```
str$ = "  data  "
str$ = Trim$(str$)   ' str$ = "data"
```

TW Function

F

Returns the status of the Wait, WaitNet, and WaitSig commands.

Syntax

TW

Return Values

Returns 0 if Wait condition is satisfied within the time interval.
Returns 1 if the time interval has elapsed.

Description

The Timer Wait function **TW** returns the status of the preceding Wait condition with time interval with a 0 (Wait condition was satisfied) or a 1 (time interval has elapsed).

See Also

TMOut, Wait

TW Function Example

```
Wait Sw(0) = On, 5 'Wait up to 5 seconds for input bit 0 On
If TW = 1 Then
    Goto TIME_UP   'Goto TIME_UP after 5 seconds
EndIf
```

Type Command

>

Displays the contents of the specified file.

Syntax

Type *fileName*

Parameters

fileName	The path and name of the file to display.

Description

Type causes the specified file's contents to be displayed. Since only ASCII files can be displayed, be sure to specify only ASCII files. The purpose of Type is to display the contents of files, not to edit files.

See Also

Dir, PList

Type Example

Type the contents of a data file.

```
> type test.dat
MyData Line 1
MyData Line 2
MyData Line 3
>
```

UBound Function

F

Returns the largest available subscript for the indicated dimension of an array.

Syntax

UBound (*arrayName* [, *dimension*])

Parameters

arrayName	Name of the array variable; follows standard variable naming conventions.
dimension	Optional. Integer expression indicating which dimension's upper bound is returned. Use 1 for the first dimension, 2 for the second, and 3 for the third. If *dimension* is omitted, 1 is assumed.

See Also

Erase, Redim

UBound Function Example

```
Integer i, a(10)

For i=0 to UBound(a)
    a(i) = i
Next
```

UCase$ Function

F

Returns a string that has been converted to uppercase.

Syntax

UCase$ (*string*)

Parameters

string String expression.

Return Values

The converted uppercase string.

See Also

LCase$, LTrim$, Trim$, RTrim$

UCase$ Example

```
str$ = "Data"
str$ = UCase$(str$)   ' str$ = "DATA"
```

UOpen Statement

S

Opens a text file for read / write access.

Syntax

UOpen *fileName* **As** #*fileNumber*
.
.
Close #*fileNumber*

Parameters

fileName	String expression that specifies valid path and file name.
fileNumber	Integer expression representing values from 30 - 63.

Description

Opens the specified text file for read / write access and identifies it by the specified file number. This statement is used for updating data in the file. **Close** closes the file and releases the file number.

fileNumber identifies the file while it is open and cannot be used to refer to a different file until the current file is closed. *fileNumber* is used by other file operations such as Print#, Read, Write, Seek, and Close.

If the specified file does not exist on disk, the file will be created and the data will be written into it.
If the specified file already exists on disk, the data will be written and read starting from the beginning of the existing data.

The read/write position (pointer) of the file can be changed using the Seek command. When switching between read and write access, you must use Seek to reposition the file pointer.

It is recommended that you use the FreeFile function to obtain the file number so that more than one task are not using the same number.

See Also

AOpen, BOpen, Close, FreeFile, Print #, ROpen, Seek, WOpen

UOpen Statement Example

```
Integer fileNum, i, pos, recNum
String rec$

fileNum = FreeFile
UOpen "TEST.TXT" As #fileNum
For i = 0 To 100
    ' Make a record with fixed length of 8 characters
   rec$ = LSet$(Str$(i), 8)
    Print #fileNum, rec$
Next i
Close #fileNum

fileNum = FreeFile
UOpen "TEST.TXT" As #fileNum
recNum = 2
 ' Calculate record position (recNum * (width + len(CRLF))
pos = recNum * (8 + 2)
Seek #fileNum, pos
Input #fileNum, rec$
Print "record = ", rec$
Close #fileNum
```

Val Function

F

Converts a character string that consists of numbers into their numerical value and returns that value.

Syntax

Val*(string)*

Parameters

string String expression which contains only numeric characters. The string may also contain a prefix: &H (hexadecimal), &O (octal), or &B (binary).

Return Values

Returns an integer or floating point result depending upon the input string. If the input string has a decimal point character than the number is converted into a floating point number. Otherwise the return value is an integer.

Description

Val converts a character string of numbers into a numeric value. The result may be an integer or floating point number. If the string passed to the Val instruction contains a decimal point then the return value will be a floating point number. Otherwise it will be an integer.

See Also

Abs, Asc, Chr$, Int, Left$, Len, Mid$, Mod, Right$, Sgn, Space$, Str$

Val Example

The example shown below shows a program which coverts several different strings to numbers and then prints them to the screen.

```
Function ValDemo
    String realstr$, intstr$
    Real realsqr, realvar
    Integer intsqr, intvar

    realstr$ = "2.5"
    realvar = Val(realstr$)
    realsqr = realvar * realvar
    Print "The value of ", realstr$, " squared is: ", realsqr

    intstr$ = "25"
    intvar = Val(intstr$)
    intsqr = intvar * intvar
    Print "The value of ", intstr$, " squared is: ", intsqr
Fend
```

Here's another example from Monitor window.

```
> Print Val("25.999")
25.999
>
```

Ver Command

>

Displays system configuration parameters.

Syntax

Ver

Return Values

Displays system configuration parameters.

Description

Displays the currently defined values for the system control data. When the robot is delivered or after changing the data it is normally a good idea to save this data. This can be done with Mkver from Maintenance dialog. It is also a good idea to print this data and store it in a safe place since information such as calibration data and other robot specific information is displayed with the Ver command. The Ver command output can be printed from the Print Dialog in EPSON RC+ by checking the Ver Command Output dialog.

The following data will be displayed. (The following data is for reference only since data will vary from controller to controller.)

```
' Version:
'   EPSON RC+ 4.2.0

' Options:
'   SPEL Runtime Drivers
'   Vision Guide
'   VB Guide

' HOUR: 5.887

' Drive Unit 1:
'   MIB I/O Address: 300
'   MIB Mem Address: D8000
'   Motor 1: Enabled, Power = 200, Gain = F7F6, Offset = FE00
'   Motor 2: Enabled, Power = 200, Gain = FDFD, Offset = 200
'   Motor 3: Enabled, Power = 200, Gain = FDFC, Offset = FF04
'   Motor 4: Enabled, Power = 100, Gain = F7F8, Offset = FDFC
' Drive Unit 2:
'   MIB I/O Address: 320
'   MIB Mem Address: D0000
'   Motor 1: Enabled, Power = 200, Gain = F8F9, Offset = FDFE
'   Motor 2: Enabled, Power = 200, Gain = 601, Offset = 2
'   Motor 3: Enabled, Power = 200, Gain = C09, Offset = FEFE
'   Motor 4: Enabled, Power = 100, Gain = FA00, Offset = FFC

ROBOT 1
' Name: rob1
‘ Model: E2C251S
‘ Serial #: 1
  ARCH 0, 30, 30
  ARCH 1, 40, 40
  ARCH 2, 50, 50
  ARCH 3, 60, 60
  ARCH 4, 70, 70
  ARCH 5, 80, 80
  ARCH 6, 90, 90
  ARMSET 0, 125, 0, 0, 125, 0
' CALPLS: Undefined
```

```
   HOFS 0, 0, 0, 0
   HOMESET 0, 0, 0, 0
   HORDR 4, 11, 0, 0
 ' HTEST: 0, 0, 0, 0
   RANGE 0, 163840, -76800, 76800, -36864, 0, -46695, 46695
   WEIGHT 1, 125
   XYLIM 0, 0, 0, 0

 ROBOT 2
 ' Name: rob2
 ‘ Model: E2C251S
 ‘ Serial #: 2
   ARCH 0, 30, 30
   ARCH 1, 40, 40
   ARCH 2, 50, 50
   ARCH 3, 60, 60
   ARCH 4, 70, 70
   ARCH 5, 80, 80
   ARCH 6, 90, 90
   ARMSET 0, 125, 0, 0, 125, 0
 ' CALPLS: Undefined
   HOFS 0, 0, 0, 0
   HOMESET 0, 0, 0, 0
   HORDR 4, 11, 0, 0
 ' HTEST: 0, 0, 0, 0
   RANGE 0, 163840, -76800, 76800, -36864, 0, -46695, 46695
   WEIGHT 1, 125
   XYLIM 0, 0, 0, 0

 ' ISA Digital I/O Boards:
 '   None installed

 ' Ethernet I/O:
 '   None installed

 ' Fieldbus I/O:
 '   None installed

 ' Standard ISA Force Sensing Boards:
 '   None installed

 ' PG Boards:
 '   None installed

 ' Conveyor Encoders:
 '   None installed
>
```

See Also

Verinit

Ver Example

```
> Ver
```

Verinit Command

>

Initializes robot system parameters.

Syntax

Verinit

Description

Verinit sets default values for all robots.

Verinit Initializes the following data

Command	Initial Value
Accel	Varies according to manipulator type
AccelR	Varies according to manipulator type
AccelS	Varies according to manipulator type
Arch	0, 30, 30 1, 40, 40 2, 50, 50 3, 60, 60 4, 70, 70 5, 80, 80 6, 90, 90
Arm	0
ArmSet	Delete user definitions (Arm 1 to 3). Arm 0 remains unchanged.
Base	Equivalent to original robot coordinate system
Base 1-15	Delete
CtrlDev	PC
Fine	Default values which depend on manipulator type
HomeSet	Delete
Hordr	Default values which depend on manipulator type
Inertia	Default values which depend on manipulator type
LimZ	0
Local 1-15	Delete
MCordr	Default values which depend on manipulator type
Range	Default values which depend on manipulator type
Speed	Default values which depend on manipulator type
SpeedR	Default values which depend on manipulator type
SpeedS	Default values which depend on manipulator type
TLSet	Delete user definitions (Tool 1 to 3). Tool 0 remains unchanged.
Tool	0
Weight	Default values which depend on manipulator type
XYLim	0, 0, 0, 0

Notes

Maintenance Purposes Only

This command is intended to be used for maintenance purposes. **Verinit** initializes values as described above. Therefore, for cases in which initial command values are not desired, be sure to revise those values after executing **Verinit**.

See Also

Ver

Verinit Command Example

```
> verinit
Initializing parameters for all robots...
>
```

Wait Statement

> S

Causes the program to Wait for a specified amount of time or until the specified input condition (using MemSw or Sw) is met. (Oport may also be used in the place of Sw to check hardware outputs.)

Syntax

(1) Wait *time*
(2) Wait *inputCondition*
(3) Wait *inputCondition*, *time*

Parameters

time Real expression between 0 and 2,147,483 which represents the amount of time to wait when using the Wait instruction to wait based on time. Time is specified in seconds. The smallest increment is .01 seconds.

inputCondition This condition must return a value of True or False. The following functions and operators may be used in the inputCondition:

Functions: Ctr, Cnv_QueLen, In, InW, Lof, MemIn, MemSw, Lof, MCalComplete, Motor, OPort, Out, OutW, Sw

Operators: And, Mask, Or, Xor, +, *

Other: Parenthesis for prioritizing operations, and variables

Description

(1) Wait with Time Interval
When used as a timer, the **Wait** instruction causes the program to pause for the amount of time specified and then continues program execution.

(2) Wait for Input Conditions without Time Interval
When used as a conditional **Wait** interlock, the **Wait** instruction causes the program to wait until specified conditions are satisfied. If after TMOut time interval has elapsed and the **Wait** conditions have not yet been satisfied, an error occurs. The user can check multiple conditions with a single Wait instruction by using the And, Mask, Or, or Xor instructions. (Please review the example section for **Wait**.)

(3) Wait with Input Condition and Time Interval
Specifies **Wait** condition and time interval. After either **Wait** condition is satisfied, or the time interval has elapsed, program control transfers to the next command. Use Tw to verify if the Wait condition was satisfied or if the time interval elapsed.

Notes

Specifying a Timeout for Use with Wait

When the **Wait** instruction is used without a time interval, a timeout can be specified which sets a time limit to wait for the specified condition. This timeout is set through using the TMOut instruction. Please refer to this instruction for more information. (The default setting for TMOut is 0 which means no timeout.)

See Also

In, InBCD, MemIn, MemOff, MemOn, MemOut, MemSw, Off, On, Oport, Out, OutW, Sw, TMOut

Wait Example

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. MemSw is used in combination with the Wait instruction to wait until the memory I/O #1 is the proper value before it is safe to move again.

```
Function main
    Integer I
    MemOff 1
    Xqt !2, task2
    For I = 1 to 100
       Wait MemSw(1) = Off
       Go P(i)
       MemOn 1
    Next I
Fend

Function task2
    Integer i
    For i = 101 to 200
       Wait MemSw(1) = On
       Go P(i)
       MemOff 1
    Next i
Fend


' Wait until input 0 turns on
Wait Sw(0) = On

' Wait 60.5 secs and then continue execution
Wait 60.5

' Wait until input 0 is off and input 1 is on
Wait Sw(0) = Off And Sw(1) = On

' Wait until memory bit 0 is on or memory bit 1 is on
Wait MemSw(0) = On Or MemSw(1) = On

'Wait one second, then turn output 1 on
Wait 1; On 1

' Wait for the lower 3 bits of input port 0 to equal 1
Wait In(0) Mask 7 = 1
```

WaitNet Statement

S

Wait for network connection to be established.

Syntax

WaitNet #*portNumber* [, *timeOut*]

Parameters

portNumber	Integer expression for port number to open. Range is 128 - 131
timeOut	Optional. Maximum time to wait for connection.

See Also

Wait, WaitSig, WaitPos

WaitNet Statement Example

For this example, two PCs have their TCP/IP settings configured as follows:

PC #1:
Port: #128
Host Name: PC2
TCP/IP Port: 1000

```
Function tcpip1
  OpenNet #128
  WaitNet #128
  Print #128, "Data from host 1"
Fend
```

PC #2:
Port: #128
Host Name: PC1
TCP/IP Port: 1000

```
Function tcpip2
  String data$
  OpenNet #128
  WaitNet #128
  Input #128, data$
  Print "received '", data$, "' from host 1"
Fend
```

WaitPos Statement

S

Wait for robot to decelerate to position before executing next statement while path motion is active.

Syntax

WaitPos [*robotNumber*]

Parameters

robotNumber Optional. Specifies which robot to wait for.

Description

Normally, when the path motion is active (CP On or CP parameter specified), the motion command starts the next statement as deceleration starts.
Write WaitPos command right before the motion to complete the deceleration motion and go on to the next motion.

See Also

Wait, WaitSig, CP

WaitPos Statement Example

```
Off 1
CP On
Move P1
Move P2
WaitPos  ' wait for robot to decelerate
On 1
CP Off
```

WaitSig Statement

S

Waits for a signal from another task.

Syntax

WaitSig *signalNumber* [, *timeOut*]

Parameters

signalNumber	Integer expression representing signal number to receive. Range is from 0 to 127.
timeOut	Optional. Real expression representing the maximum time to wait.

Description

Use **WaitSig** to wait for a signal from another task. The signal will only be received after **WaitSig** has started. Previous signals are ignored.

See Also

Wait, WaitPos, Signal

WaitSig Example

```
Function Main
  Xqt SubTask
  Wait 1
  Signal 1
  .
  .
Fend

Function SubTask
  WaitSig 1
  Print "signal received"
  .
Fend
```

Weight Statement

> S

Specifies or displays the inertia of the robot arm.

Syntax

Weight *payloadWeight* [*, distance*]
Weight

Parameters

payloadWeight	The weight of the end effector to be carried in Kg unit.
distance	The distance from the rotational center of the second arm to the center of the gravity of the end effector in mm unit. Valid only for SCARA robots.

Return Values

Displays the current **Weight** settings when parameters are omitted.

Description

Specifies parameters for calculating Point to Point motion maximum acceleration. The **Weight** instruction specifies the weight of the end effector and the parts to be carried.

The Arm length (*distance*) specification is necessary only for SCARA type robots. It is the distance from the second arm rotation joint centerline to the hand/work piece combined center of gravity.

If the equivalent value work piece weight calculated from specified parameters exceeds the maximum allowable payload, an error occurs.

Potential Errors

Weight Exceeds Maximum

When the equivalent load weight calculated from the value entered exceeds the maximum load weight, an error will occur.

Potential Damage to the Manipulator Arm

Take note that specifying a **Weight** hand weight significantly less than the actual work piece weight can result in excessive acceleration and deceleration. These, in turn, may cause severe damage to the manipulator.

Note

Weight Values Are Not Changed by Turning Main Power Off

The **Weight** values are not changed by turning power off. However, the values are initialized by executing the Verinit command. Since initial values vary with each manipulator type, refer to the manipulator manual for **Weight** initial values.

See Also

Accel, Inertia, Verinit

Weight Statement Example

This Weight instruction on the Monitor window displays the current setting.

```
> weight
2.000, 200.000
>
```

Weight Function

F

Returns a Weight parameter.

Syntax

Weight(*paramNumber*)

Parameters

paramNumber	Integer expression containing one of the values below: 1: Payload weight 2: Arm length

Return Values

Real number containing the parameter value.

See Also

Inertia, Weight Statement

Weight Function Example

```
Print "The current Weight parameters are: ", Weight(1)
```

Where Statement

>

Displays current robot position data.

Syntax

Where [*localNumber*]

Parameters

localNumber Optional. Specifies the local coordinate system number. Local 0 is default.

See Also

Joint, PList, Pulse

Where Statement Example

```
>where
WORLD:  X:  350.000 mm  Y:    0.000 mm  Z:    0.000 mm  U:    0.000 deg
JOINT:  1:    0.000 deg 2:    0.000 deg 3:    0.000 mm  4:    0.000 deg
PULSE:  1:        0 pls 2:        0 pls 3:        0 pls 4:        0 pls

> local 1, 100,100,0,0

> where 1
WORLD1: X:  250.000 mm  Y: -100.000 mm  Z:    0.000 mm  U:    0.000 deg
JOINT:  1:    0.000 deg 2:    0.000 deg 3:    0.000 mm  4:    0.000 deg
PULSE:  1:        0 pls 2:        0 pls 3:        0 pls 4:        0 pls
```

While...Wend

S

Executes the specified statements while the specified condition is satisfied.

Syntax

While [*condition*]
[*statements*]
Wend

Parameters

condition	Any valid condition which may be the checking of I/O with the Sw command or the comparison of variables.
statements	Optional. One or more statements that are repeated while *condition* is True.

Description

Specifies the While condition. If satisfied, statements are executed between While and Wend, then the While condition is checked once again. Executing While...Wend statements and rechecking the While condition continues as long as the While condition is satisfied.

If the While condition is not satisfied, program control transfers to the command following Wend. For a While condition that is not satisfied at the first check, the statements inside of the While...Wend statements are never executed. Nesting of **While...Wend** statements is supported up to 256 levels deep including other statements (**Do…Loop**, **If...Then...Else…EndIf**, and **Select…Send**).

Notes

Rules for While...Wend:

- Every **While** must be followed by a **Wend**
- Each **Wend** corresponds to the preceding **While**. A **Wend** without a preceding **While** will result in an error.
- Any valid operator may be included in the **While** condition.

See Also

If/Then/Else

While Example

```
Long i

i = 1
While i < 60    'Execute statements between While/Wend if i < 60
 .
 .
   i = i + 2
Wend
```

WOpen Statement

S

Opens a file for writing.

Syntax

WOpen *fileName* **As** *#fileNumber*

Parameters

fileName	A string expression containing the file name to read from. The drive and path can also be included.
fileNumber	Integer from 30 - 63 which will be used as a handle for the file.

Description

Opens the specified filename for writing and identifies it by the specified *fileNumber*. This statement is used to open and write data to the specified file. Close closes the file and releases the file number. (To append data refer to the AOpen explanation.)

If the specified filename does not exist on the disks current directory, **WOpen** creates the file and writes to it. If the specified filename exists, WOpen erases all of the data in the file and writes to it.

The *fileNumber* identifies the file as long as the file is open. It is used by the Input and Write statements for writing, Print # statement for printing and Close for closing. Accordingly, until the current file is closed, its file number cannot be used to specify a different file.

It is recommended that you use the FreeFile function to obtain the file number so that more than one task are not using the same number.

See Also

AOpen, BOpen, Close, Input #, ROpen, UOpen

WOpen Example

Shown below is a simple example which opens a file, writes some data to it, then later opens the same file and reads the data into an array variable.

```
Real data(100)
Integer fileNumber, i

For i = 0 To 100
    data(i) = i
Next i

fileNumber = FreeFile
WOpen "TEST.VAL" As #fileNumber
For i = 0 To 100
    Print #fileNumber , data(i)
Next i
Close #fileNumber

fileNumber = FreeFile
ROpen "TEST.VAL" As #fileNumber
For i = 0 to 100
    Input #fileNumber , data(i)
Next i
Close #fileNumber
```

Wrist Statement

> S

Sets the wrist orientation of a point.

Syntax

(1) Wrist *point*, [*value*]
(2) Wrist

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Integer expression. 1 = NoFlip (/NF) 2 = Flip (/F)

Return Values

When both parameters are omitted, the wrist orientation is displayed for the current robot position.
If *value* is ommited, the wrist orientation for the specified point is displayed.

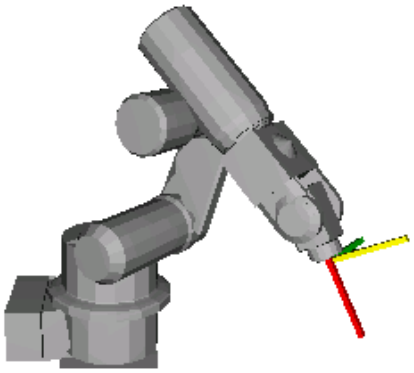
See Also

Elbow, Hand, J4Flag, J6Flag, Wrist Function

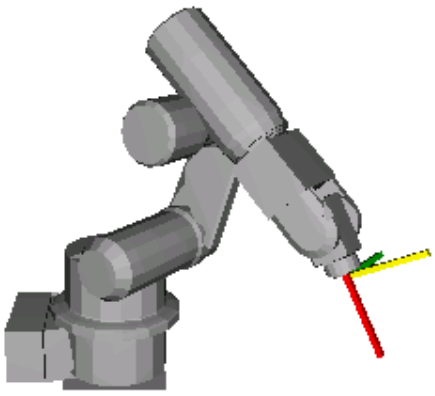
Wrist Statement Example

```
Wrist P0, Flip
Wrist P(mypoint), NoFlip

P1 = 320.000, 400.000, 350.000, 140.000, 0.000, 150.000
```



```
Wrist P1, NoFlip
Go P1
```



```
Wrist P1, Flip
Go P1
```

Wrist Function

F

Returns the wrist orientation of a point.

Syntax

Wrist [(*point*)]

Parameters

point Optional. **P*** or **P***number* or **P**(*expr*) or point label or point expression. If *point* is ommitted, then the wrist orientation of the current robot position is returned.

Return Values

1 NoFlip (/NF)
2 Flip (/F)

See Also

Elbow, Hand, J4Flag, J6Flag, Wrist Statement

Wrist Function Example

```
Print Wrist(pick)
Print Wrist(P1)
Print Wrist
Print Wrist(P1 + P2)
```

Write Statement

S

Writes characters to a file without end without end of line terminator.

Syntax

Write #*fileNumber*, *string*

Parameters

fileNumber	File or communications port to write to.
string	String expression that will be written to the file.

Description

Write is different from Print in that it does not add an end of line terminator.

The current file pointer is advanced by the length of the string expression.

See Also

Print, Read, Seek, WOpen

Write Example

```
WOpen "test.dat" As #30
For i = 1 to 10
    Write #30, data$(i)
Next i
Close #30
```

WriteBin Statement

S

Writes one byte to a file or communications port.

Syntax

WriteBin #*fileNumber*, *data*

Parameters

fileNumber	File or communications port to read from.
data	Integer expression containing the data to be written.

See Also

BOpen, ReadBin, Seek, Write

WriteBin Statement Example

```
Integer f, i, data(100)

f = FreeFile
BOpen "test.dat" As #f
For i = 0 To 100
  WriteBin #f, i
Next i
Close #f

f = FreeFile
BOpen "test.dat" As #f
For i = 0 To 100
  ReadBin #f, data(i)
Next i
Close #f
```

Xor Operator

Performs the bitwise Xor operation (exclusive OR) on two expressions.

Syntax

result = *expr1* **Xor** *expr2*

Parameters

expr1, expr2 A numeric value, or a variable name.

result An integer.

Description

The **Xor** operator performs the bitwise **Xor** operation on the values of the operands. Each bit of the result is the **Xor**ed value of the corresponding bits of the two operands.

If bit in expr1 is	And bit in expr2 is	The result is
0	0	0
0	1	1
1	0	1
1	1	0

See Also

And, LShift, Not, Or, RShift

Xor Operator Example

```
>print 2 Xor 6
 4
>
```

Xqt Statement

S

Initiates execution of a task from within another task.

Syntax

Xqt [*taskNumber,*] *funcName* [**(***argList***)**] [, **NoPause**]

Parameters

taskNumber	Optional. The task number for the task to be executed. The range of the task number is 1 to 32. **Note**: The SPEL Language on the SRC-3xx series controllers used an exclamation point before the task number. The exclamation point is no longer necessary and is ignored.
funcName	The name of the function to be executed.
argList	Optional. List of arguments that are passed to the function procedure when it is called. Multiple arguments are separated by commas.
NoPause	Optional. Specifies that the task will not pause when a Pause statement or signal occurs.

Description

Xqt starts the specified function and returns immediately.

Normally, the *taskNumber* parameter is not required. When *taskNumber* is omitted, SPEL$^+$ automatically assigns a task number to the function, so you don't have to keep track of which task numbers are in use.
Notes

The SPEL Language on the SRC-3xx series controllers supported *startLine* and *endLine* arguments. These are not supported in SPEL$^+$.

See Also

Chain, Function/Fend, Halt, Resume, Quit

Xqt Example

```
Function main
    Xqt flash, NoPause 'Start flash function as task 2
    Xqt Cycle(5)       'Start Cycle function as task 3

    Do
        Wait 3         'Execute task 2 for 3 seconds
        Halt flash     'Suspend the task

        Wait 3
        Resume flash   'Resume the task
    Loop
Fend

Function Cycle(count As Integer)
    Integer i

    For i = 1 To count
        Jump pick
        On vac
        Wait .2
        Jump place
        Off vac
        Wait .2
    Next i
Fend

Function flash
    Do
        On 1
        Wait 0.2
        Off 1
        Wait 0.2
    Loop
Fend
```

XY Function

F

Returns a point from individual coordinates that can be used in a point expression.

Syntax

XY(*x, y, z, u,* [*v*, *w*]**)**

Parameters

x Real expression representing the X coordinate.
y Real expression representing the Y coordinate.
z Real expression representing the Z coordinate.
u Real expression representing the U coordinate.
v Optional for 6-Axis robots. Real expression representing the V coordinate.
w Optional for 6-Axis robots. Real expression representing the W coordinate.

Return Values

A point constructed from the specified coordinates.

See Also

JA, Point Expression

XY Function Example

```
P10 = XY(60, 30, -50, 45) + P20
```

XYLim Statement

> S

Sets or displays the permissible XY motion range limits for the robot.

Syntax

XYLim *minX, maxX, minY, maxY,* [*minZ*], [*maxZ*]
XYLim

Parameters

minX	The minimum X coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the X Coordinate less than *minX*.)
maxX	The maximum X coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the X Coordinate greater than *maxX*.)
minY	The minimum Y coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Y Coordinate less than *minY*.)
maxY	The maximum Y coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Y Coordinate greater than *maxY*.)
minZ	Optional. The minimum Z coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Z Coordinate less than *minZ*.)
maxZ	Optional. The maximum Z coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Z Coordinate greater than *maxZ*.)

Return Values

Displays current **XYLim** values when used without parameters

Description

XYLim is used to define XY motion range limits. Many robot systems allow users to define joint limits but the SPEL$^+$ language allows both joint limits and motion limits to be defined. In effect this allows users to create a work envelope for their application. (Keep in mind that joint range limits are also definable with SPEL.)

The motion range established with **XYLim** values applies to motion command target positions only, and not to motion paths from starting position to target position. Therefore, the arm may move outside the XYLim range during motion. (i.e. The **XYLim** range does not affect Pulse.)

Notes

Turning Off Motion Range Checking

There are many applications which don't require Motion Range limit checking and for that reason there is a simple method to turn this limit checking off. To turn motion range limit checking off, define the Motion Range Limit values for *minX*, *maxX*, *minY*, and *maxY* to be 0. For example XYLim 0, 0, 0, 0.

Default Motion Range Limit Values

There are some cases which cause the system to set everything back to their original defaults. For example, the Verinit command will cause all system values back to their defaults. The default values for the XYLim instruction are "0, 0, 0, 0". (Motion Range Limit Checking is turned off.)

Tip

Point & Click Setup for XYLim

EPSON RC+ has a point and click dialog box for defining the motion range limits. The simplest method to set the XYLim values is through using the XYLim tab: Robot Parameters command (Project Menu) .

See Also

XYLim tab: Robot Parameters command (Project Menu)
Range

XYLim Statement Example

This simple example from the monitor window sets and then displays the current XYLim setting:

```
> xylim -200, 300, 0, 500

> XYLim
-200.000, 300.000, 0.000, 500.000
```

XYLim Function

F

Returns point data for either upper or lower limit of XYLim region.

Syntax

XYLim(*limit*)

Parameters

limit	Integer expression that specifies which limit to return. 1: Lower limit. 2: Upper limit.

Return Values

Point containing the specified limit coordinates.

See Also

XYLim Statement

XYLim Function Example

```
P1 = XYLim(1)
P2 = XYLim(2)
```

ZeroFlg Function

F

Returns value of memory I/O previous to it last being switched on or off.

Syntax

ZeroFlg

Return Values

Returns the value of the memory bit previous to it last being switched on or off. (Returns a 0 or a 1)

Description

ZeroFlg is for exclusive control of a single resource (such as an RS232 port) while running multiple tasks.

After turning on a memory bit, use **ZeroFlg** to determine if the current task is the task that turned on the bit (vs some other task turning is on at the same time). If the current task turned on the bit, then **ZeroFlg** will return 0, indicating that the previous value was 0.

You must user **ZeroFlg** immediately after turning on the memory bit.

For example:

```
MemOn 1
If ZeroFlg = 0 Then
  Print "This task turned on memory bit 1"
Else
  Print "Some other task turned on memory bit 1"
EndIf
```

See Also

MemOff, MemOn, SyncLock, SyncUnlock, WaitSig

ZeroFlg Function Example

This example uses two functions, AccessPort and ReleasePort, to allow only one task at a time to communicate with the device connected to RS232 port 1. When one task is using the port, the other task will wait until it can get control of the port.

```
Function main

  ' Initialize access flag
  MemOff port

  Xqt task2

  Do
    AccessPort
    Print #1, "main"
    ReleasePort
  Loop
Fend

Function task2

  Do
    AccessPort
    For i = 0 To 100
      Print #1, i
    Next
    ReleasePort
  Loop
Fend

Function AccessPort

Do
  ' Turn on the access flag
  MemOn port
  ' Check if this task turned it on
  If ZeroFlg = 0 Then
    ' This task has control so exit
    Exit Function
  EndIf
  ' Wait for the controlling task
  ' to free the resource
  Wait MemSw(port) = Off
Loop
Fend

Function ReleasePort

  MemOff port
  ' Allow time for another task
  ' to get access
  Wait .01
Fend
```

SPEL+ Error Messages

To get help for any SPEL+ error, place the cursor on the error message in the run or monitor windows and press the F1 key.

Error #	Message
2	Syntax error.
3	A corresponding GOSUB statement was not found for a RETURN statement.
4	Too many [GOSUB] statements. (Maximum 512)
7	There are more nested GOSUB statements than allowed [16].
9	Subscript out of range.
16	Statement not allowed outside of function body.
27	The Input or Output Bit No. is out of available range.
30	There was not enough data to satisfy INPUT parameters.
53	The specified file cannot be opened.
63	The specified disk drive is not available.
65	File read error.
66	File write error.
77	Point number is invalid.
78	The specified point data has not been defined.
88	The bit has been assigned for remote use.
92	Safe guard input signal failure.
94	The specified event condition did not occur for WAIT during the timeout period set by TMOUT.
121	The command could not be executed under emergency stop condition.
124	The specified value is out of allowable range.
125	The arm reached the limit of motion range.
126	The Arm reached the limit of XY motion range (specified by XYLIM).
129	The current point position is above the specified LIMZ value.
143	HOME was executed before setting HOMESET.
146	Command not allowed with one or more joints free.
150	Command not allowed with motors off.
151	Arm positioning cannot be completed as specified by FINE.
173	Torque error in the low-power state.
174	Torque error in the high-power state.
185	Encoder signal cable disconnection.
190	Absolute Encoder overheat.
191	Absolute Encoder overspeed.
193	Absolute Encoder battery alarm.

Error #	Message
194	Absolute Encoder check-sum error.
195	Absolute encoder backup alarm.
230	MCORG has not been completed.
231	MCAL has not been completed.
233	The distance from the Home Sensor to the Encoder Z Phase varies significantly from the previous value.
359	CodeReading option is not enabled.
360	Undefined vision sequence
361	Undefined vision object
362	Undefined vision property
363	Vision object not found
364	Improper value for calibration property
365	No previous sequence with acquire
366	Illegal property value
367	Illegal result for specified object
368	Invalid property or result
369	Wrong number of arguments
370	Model not trained
371	Incomplete calibration
372	Undefined calibration
373	Calibration points have not been taught
374	Printer error
375	No controller
376	Bad object type
377	Vision System error
378	Sequence already exists
379	Path not found
380	Invalid camera
381	Format string error
382	Invalid image file
383	ModelObject does not have angle enabled
384	Object already exists
385	Calibration mark not found
386	Cal target sequence object 'NumberToFind' property is less than 9.
387	Not enough steps in cal target sequence.
388	No upward camera sequence specified.
389	Upward camera sequence has no calibration.
390	Upward camera sequence calibration has not been completed.
391	Bad calibration reference.
392	No target sequence for calibration.
393	Incorrect string length for calibration or verify.
394	Invalid font.
395	Invalid CalString value.
396	Invalid character position parameter for constraints.
397	Invalid CalString length.
398	Ocr calibration failed.
399	No objects in sequence

Error #	Message
400	Missing FUNCTION for FEND
401	Missing FEND for FUNCTION
402	Duplicate line number
403	Bad line number
404	Line number too large
405	Function already declared as DLL function
406	Main function of group cannot use parameters
407	Tasks aborted by emergency stop
408	Point file not found for current robot
409	Invalid command for the current robot
410	Variable access timeout
411	Type mismatch
412	DeviceNet access timeout
413	Bad dimension specified
414	Groups are not enabled for the current project
415	Out of memory
416	Not an array
417	SPEL OLE Channel busy
418	SPEL OLE Command Timeout
419	EPSON RC+ access timeout
420	ENetIO access timeout
421	Trap is already running
422	ERCLIB1 OLE Error
423	Task aborted
424	Power High is disabled because EnablePowerHigh remote input is off.
425	Comm port is not supported
426	Cannot continue with safe guard open
427	Robot not in current project
428	Cannot continue tasks.
429	Cannot recover safeguard position.
430	Tasks aborted by safeguard open.
500	Expected 'All'
501	Expected 'As'
502	Expected '='
503	Bad arm number. Range is 0 to 3.
504	Bad point attribute
505	Bad axis. Use X, Y, Z, U, V, W.
506	Bad file number
507	Bad identifier
508	Only # statements are allowed in include files
509	Bad pallet number. Range is 0 to 15
510	Bad point
511	Bad subscript
512	Bad tool number. Range is 0 to 3.
513	Bad baud rate
514	Expected CALL

Error #	Message
515	CALL, GOTO, or GOSUB expected
516	Expected colon
517	Expected comma
518	Bad data bits
519	Data type expected
520	Illegal declaration inside function
521	Cannot use #define inside a function
522	Too many dimensions
523	Expected DLL alias
524	Expected left parentheses or AS
525	Expected DLL path and file name
526	Duplicate function name
527	Else expected
528	Exclamation expected
529	Expected Do, For, or Function
530	Expected point number
531	Expected trap number or mode
532	Expression expected
533	Expected array
534	Bad file number. Range is 30 - 63
535	Number expected
536	Must be used as a function
537	Bad hardware flow specified
538	Filename expected with .INC extension
539	INPUT expected
540	Integer expression expected
541	Cannot use keyword as identifier
542	Cannot use keyword as label
543	Expected ON or OFF
544	Expected HIGH or LOW
545	Cannot include inside a function
546	Line numbers are not allowed in include files
547	Left parentheses expected
548	Line number or label expected
549	Expected MERGE
550	Missing point attribute
551	Missing axis
552	Missing identifier
553	Expected String
554	Missing arm number
555	Operand expected
556	Missing pallet number
557	Missing tool number
558	Execute MKVER from Tools | Maintenance
559	Expected NEXT
560	No arguments were expected

Error #	Message
561	End of statement expected
562	Keyword cannot be used as a function
563	Command not allowed from monitor window
564	Expected variable only, not an array
565	Command not supported on EPSON RC+ controller.
566	Instruction is not yet supported
567	Vision system not enabled
568	Bad parallel processing delay value
569	Illegal parallel processing statement
570	Expected parallel processing statement
571	Parallel processing argument is currently not supported
572	Bad parity
573	Expected dot
574	Point expected
575	# expected
576	Quote expected
577	Expected Emergency, Output, or Robot
578	Wrong number of dimensions
579	Right parentheses expected
580	Semicolon expected
581	Execute SETVER from Tools | Maintenance
582	String expected
583	STEP expected
584	Bad stop bits
585	Bad software flow
586	Task number (1 - 32) expected
587	Bad terminator
588	TO expected
589	THEN expected
590	WHILE or UNTIL expected
591	Undefined label
592	Wrong number of arguments
593	No corresponding SEND statement
594	Invalid trap number. Range is 1 - 4.
595	Include file does not exist
596	Include file is not in the current project
597	Bad include file name
598	Variable is already declared as global
599	Variable is already declared at module level
600	Variable is already declared as function parameter
601	Type mismatch
602	A file cannot include itself
603	Expected point number or '*'
604	Expected file number or variable
605	Too many parameters
606	The include file has already been included

Error #	Message
607	Expected task type
608	Invalid task type
609	Cannot pass array by value. Use ByRef.
610	Command is only valid from monitor window.
611	Task is already running.
612	Bad parameter
613	Expected right bracket.
614	Too many ELSE instructions in statement
615	Expected point label
616	Invalid timer number
617	All tasks aborted by change in remote EnablePowerHigh input
618	Expected GOTO
619	SPEL+ Runtime option not active.
620	SPEL+ runtime drivers restart required. Check drive unit power and cables.
621	SPEL+ runtime drivers restart failed. Check drive unit power and cables.
623	Invalid point data
624	Arch not allowed for this type of motion statement
625	No monitor commands allowed in Teach Mode.
626	No matching #ifdef or #ifndef for #endif
627	Tasks were stopped by Teach mode ON
628	For variable already in use
629	Too many For...Next loops in function
630	No function found
631	Not a function
632	Macro not defined
633	Invalid SPELCom event number. Events 1 - 999 are reserved for system.
634	Identifier too long. 32 characters max.
635	Invalid keyNumber for OP_Key
636	OP user page is inactive
637	Invalid pageNumber for OP_Page
638	OP is not enabled
639	Invalid region for OP_Print
640	CtrlDev is not PC
641	Command not allowed in Teach mode
642	Jump cannot be executed. Robot has no Z joint.
643	Invalid sync lock ID.
644	SyncLock mutex could not be created.
645	SyncUnlock failed. Lock was not owned by this task.
646	Fieldbus option not installed
647	Fieldbus invalid bus number
648	Fieldbus general error
649	Fieldbus invalid board
650	Fieldbus device not configured
651	Fieldbus invalid parameter
652	6 axis robots are not supported with this version of OP500RC.
653	Point file name is used by another robot

Error #	Message
654	Calibration already exists
655	I/O label invalid or not allowed
656	Fieldbus device connection denied
657	Fieldbus invalid device configuration
658	Fieldbus line defect. Check power, baud rate, cables.
659	Fieldbus sendBytes array too large
660	ROT not allowed with this command
661	ECP not allowed with this command
662	Fieldbus invalid device.
663	Invalid object used for calibration vision sequence final result.
664	ECP Option is not active.
665	A path cannot be specified. Only a file name is required.
666	Source and destination cannot be the same.
667	RunDialog already in cycle.
668	String too long.
669	Property cannot be changed when ModelObject is not Self.
670	Ethernet IO is not supported in Wait statement.
671	Fieldbus IO is not supported in Wait statement.
672	Object cannot be taught outside of the video image.
673	Point cannot be taught. One or more joints is out of range.
700	Invalid ENetIO rack number
701	Invalid ENetIO point type
702	ENetIO communication time out
703	ENetIO general error
704	ENetIO socket interface could not be found
705	ENetIO cannot create socket
706	ENetIO cannot connect socket
707	ENetIO bad response from brain module
708	Ethernet I/O option not active
750	Force Sensing Option not active
751	Invalid force sensor
752	Invalid force sensor axis
800	Security option is not active
801	Invalid user
802	Invalid password
803	Permission denied
850	Conveyor Tracking option is not active
851	Conveyor has not been calibrated
900	Code Reading option is not active
1000	Error count exceeds 20; stopping compilation
1001	Direct numeric typed integer is too small
1002	Pointed wrong parameters
1003	Too many characters in literal
1004	Direct numeric typed integer is too large. (Beyond specified range.)
1010	Direct numeric typed real is too large.
1011	Direct numeric typed real is too small.

Error #	Message
1012	Overflow or Underflow.
1020	Line number is too big. (Overflow)
1030	Parameter out of range.
1040	Array dimension out of range.
1130	Syntax error.
1146	Unsigned integer(1-8) is not correct.
1147	String expression is not correct.
1148	Variable name and string variable name is not correct.
1149	Illegal function name.
1150	Integer and variable that is not signed "#" is not correct.
1151	Unsigned integer is not correct.
1152	Unsigned integer | (Functions) are not correct.
1161	Line number and labels are not correct.
1162	Expression is not correct.
1163	Expression error. Describe by Expression used parenthesis.
1164	Position Expression is not correct.
1170	The expression for input condition is not correct.
1174	Point data is not correct.
1175	Point is not correct.
1176	Direct designate point is not correct.
1177	Function name is not correct.
1178	Definition argument order is not correct.
1179	String expression (path list) is not correct.
1180	String expression (directory path list) is not correct.
1181	Drive name is not correct.
1182	File name is not correct.
1183	Directory name is not correct.
1185	Argument order is not correct.
1186	Parallel process is not correct.
1187	Can not call user function during parallel processing.
1188	Can not call the user function with the argument(s) in the TRAP statement.
1198	String variable is not correct.
1199	Axis destination expression is not correct.
1200	Print list is not correct.
1201	Invalid point label.
1231	Invalid use of THEN.
1244	Illegal CASE syntax.
1252	Syntax error.
1254	Local coordinates are not correct.
1255	Arch syntax is not correct.
1256	AS # unsigned integer | variable
1259	Parenthesis at the function is not correct.
1260	The identifier is not described correctly.
1261	The parenthesis is not closed.
1265	Variable name is not correct.
1266	No parameters followed "*".

Error #	Message
1267	Invalid LINE INPUT syntax.
1268	Invalid GOTO, GOSUB, CALL syntax.
1269	Syntax error.
1270	Syntax error.
1400	Illegal monitor mode command.
1450	Function not found
1500	No corresponding FOR statement.
1501	Invalid use of Else.
1502	Invalid use of Case.
1503	Invalid use of Exit.
1504	No statements allowed between Select and first Case.
1505	Block statement error.
1551	Invalid use of Fend.
1552	Statement not allowed outside of function body.
1570	Wrong number of parameters.
1571	The number of position parameters is incorrect.
1590	Macro instruction syntax error.
1591	Macro instruction include syntax error.
1597	Syntax error.
1598	Syntax error.
1600	No corresponding ENDIF statement.
1601	No corresponding END SELECT statement.
1602	No corresponding WEND statement.
1603	No corresponding LOOP WHILE, UNTIL statement.
1604	No corresponding LOOP statement.
1605	No corresponding NEXT statement.
1606	No corresponding IF statement.
1607	No corresponding SELECT statement.
1608	No corresponding WHILE statement.
1609	No corresponding DO WHILE, UNTIL statement.
1610	No corresponding DO statement.
1650	Undefined line number.
1651	Undefined line label.
1652	Undefined variable.
1700	Run [EXIT] two times in the same nest.
1701	ELSE found two times in the same IF statement.
1702	Default found two times in the same SELECT statement.
1703	The iteration has too many nesting levels. (Maximum 16 steps.)
1704	The conditional statement has too many nesting levels. (Maximum 16 steps)
1705	Too many conditions of a SELECT statement.
1706	The condition statements are too large. (Maximum 250)
1750	Too many pause declarations. (Maximum 4)
1755	Too many "String + String" expressions. (Maximum 16)
1756	Too many string functions. (Maximum 8)
1757	Too many parameters.
1758	Too many string const, string variable or string functions. (Max 17)

Error #	Message
1760	Too many parallel processing statements.
1761	Too many output commands in the CURVE statement (MAX 16)
1800	The specified command or statement cannot be used.
1900	Too many GOSUB statements (MAX 512).
1908	The same line number was defined multiple times.
1909	The same label name was defined multiple times.
1910	The same variable name was defined multiple times.
1912	Too many literal characters.
1913	Too many characters used on a line.
1914	Too many variables used on a line.
1920	The same function name was defined multiple times.
1921	The same extern function name was defined multiple times.
1930	The end of file is abnormal.
1940	Undefined variable.
1941	Not defined as an array variable.
1942	The dimension of array is wrong.
1943	Defined as an array variable.
1944	String arrays can only contain two dimensions.
1950	Undefined function internal code.
1960	Lack of working area.
1970	Tree buffer too deep.
1971	Can not determine internal code.
1972	There are too many labels in a file.
1973	There are too many lines in a file.
1974	Too many blocks(the top) in a file.
1975	Too many blocks(jumping position) in a file.
1985	Syntax error.
1990	Too many internal codes were generated in the capacity of one line.
1992	Extern function area full.
1993	Can not keep variable area.
1994	Constant table full.
1995	Variable storage area full.
1996	Variable table full.
1997	Can not keep sectors.
1998	Internal code buffer full.
1999	Parser error.
2000	The command or statement that was specified cannot be used.
2001	The specified TMOUT value is wrong.
2002	The specified TMR value is wrong.
2003	You may not use a negative value as the argument.
2004	A negative value cannot be specified for WAIT.
2005	The specified argument to the TAN( ) function is out of allowed range.
2006	The arguments of the called function do not match the definition.
2007	The Byte type entered is out of specified range [0-255].
2009	The designated line number does not exist.
2010	The number of characters in the string is over 255.

Error #	Message
2011	The error number for ERROR statement is out of range [1-32767].
2012	The Task Number specified is out of available range [1-32].
2013	The Joint Number specified is out of available range [1-6].
2015	The specified Input or Output Port No. is out of available range.
2016	The specified BCD data is out of valid range.
2017	The specified Signal No. for SIGNAL or WAITSIG is out of range [0-127].
2018	The specified Memory I/O Bit No. is out of range [0-511].
2019	The specified Memory I/O Port No. is out of range [0-63].
2020	Too many event conditions for WAIT.
2021	Too many event conditions for TRAP.
2022	The Word type specified is out of range [0-65535].
2023	The Bit type specified is neither 0 nor 1.
2024	The specified directory or file name cannot be found.
2025	The COM No. is out of available range.
2026	Division by zero.
2027	The DSW Number is out of specified range.
2028	The specified robot does not exist.
2029	You may not mask all the input conditions for WAIT.
2030	You may not mask a bit condition for WAIT.
2032	The number of elements in the array variable is beyond the dimension.
2033	The dimensions of the array variable do not match the definition.
2034	The Group No. is out of specified range [1-16].
2035	The specified file name cannot be found.
2036	The number of points defined in the point file is beyond the maximum number of points.
2037	The coordinate axis definition in the point file does not match to the point data definition in memory.
2038	The parameter to SHUTDOWN is not within the allowable range [0-4].
2039	The specified time delay for the asynchronous output is beyond maximum value [10 sec.].
2040	Invalid date or time format.
2041	Command No. error.
2042	The specified number is out of range.
2043	The Application number is out of range.
2044	The Scheme File number is out of range.
2045	The Scheme Data is not registered.
2046	The Application number is out of nominal range.
2047	The CONFIG Mode number is invalid.
2048	The CONFIG Protocol number is invalid.
2049	The CONFIG Baud Rate number is invalid.
2050	The specified value for DIMAREA is out of allowable range [1-3].
2051	The specified value for TCSPEED is out of allowable range [1-100].
2052	The specified HTASKTYPE value is out of specified range [0-3].
2053	The specified I/O Label has not been defined.
2054	The specified I/O port number for IOLABEL$ is out of specified range.
2055	The specified I/O type for IOLABEL$ is invalid [1, 8 or 16].
2056	The specified I/O kind for IOLABEL$ is out of specified range [0-2].

Error #	Message
2057	The specified Point Label has not been defined.
2058	Point number is invalid.
2059	The specified Orientation No. for PORIENT is invalid [0, 1 or 3].
2060	The specified LOCAL No. for PLOCAL is out of specified range [0-15].
2061	The specified point data for INPUT has invalid form.
2066	Overflow
2067	Negative overflow
2069	The specified information number is an invalid number of CTRLINFO().
2078	The specified event resource in Cnv_QueLen() is incorrect.
2081	The specified character code is an invalid code of CHR$().
2083	The specified character following ampersand is an invalid code of VAL().
2084	The character string of the VAL() cannot be converted.
2085	Invalid J4FLAG value.
2086	Invalid J6FLAG value.
2087	Invalid HAND value.
2088	Invalid ELBOW value.
2089	Invalid WRIST value.
2091	A critical error has occurred. No tasks can be executed.
2092	There is an error in the write filter process.
2093	The write filter cannot be used in this system.
2102	There is no entry for SPELCT in the Registry.
2104	MOTOR ON cannot be executed because a critical error has occurred.
2105	Hard disk S.M.A.R.T. failure is predicted.
2150	The version of the project file is invalid.
2151	Project file Read error.
2152	The header of the project file is invalid.
2153	The point setting of the project file is invalid.
2154	The variable setting of the project file is invalid.
2155	Project file read error.
2160	I/O label file read error.
2165	Point file create error.
2166	The header of the point file is invalid.
2167	The number of points in the point file is beyond the max.
2168	Point file Setting error.
2170	The sector of the object file is invalid.
2171	The data of the object file is invalid.
2172	The program file was not found.
2173	The include file was not found.
2174	The version of the object file is invalid.
2175	Object file Read error.
2200	The Control box control error.
2201	All conditions are FALSE in the selective wait.
2202	RCVMSG has already executed.
2203	SNDMSG has already executed.
2204	The DECLARE command has not been executed.
2300	Jump Table error.

Error #	Message
2301	The number of tasks to be executed is beyond the maximum number of executable tasks [32].
2302	The specified function can not be found.
2303	The specified function is being stopped by an execution error.
2304	The specified function is being executed.
2305	No space available in the user program.
2308	The specified task (or sub system) cannot be found.
2309	The specified function is being edited and cannot be executed.
2310	The number of breakpoints is beyond the limit [64].
2311	The memory area for the string variables is not enough.
2312	The specified task cannot be found.
2313	All the CTRs are running.
2314	There is no bit number assigned to the specified CTR.
2316	The specified port includes bits already assigned for remote use.
2317	Invalid INBCD value.
2318	No more then 4 TRAPs can be used at one time.
2319	No BFCB Free List found.
2320	No TCB Free List found.
2321	Messaging error.
2322	Messaging exceeds the Buffer limit.
2323	The specified robot is already executing a parallel process.
2324	An error occurred during parallel processing.
2325	The specified path (file) was not found.
2327	The specified task number is being used.
2328	The specified task number is outside of the range [1-32].
2329	GOSUB statement cannot be used in the TRAP process.
2330	CALL statement cannot be used in the TRAP process.
2331	Arguments cannot be used in the TRAP process function.
2332	The specified event condition did not take place for WAIT during the timeout period set by TMOUT.
2333	The specified Sub System has already begun.
2334	Parameter stack overflow.
2335	The specified DLL file was not found.
2336	The specified function has already been defined.
2337	There is not enough space left for DLLFAT.
2338	The specified Communication Port Number cannot be found.
2339	The specified group cannot be found.
2340	The specified group cannot be found.
2341	Main Function has not been declared.
2342	Main Function has not been declared.
2343	Output command cannot be executed during Emergency Stop condition.
2344	Task handling command cannot be executed under Emergency Stop condition.
2345	The specified command cannot be executed in Privilege Task.
2346	EResume statement cannot be used without OnErr GoTo.
2347	String Expression Stack overflow.
2348	XQT statement cannot be used in the TRAP process.
2349	Signal was not received for WAITSIG during the timeout period set by TMOUT.

Error #	Message
2351	CALL statement cannot be used in the command line.
2352	Too many points specified in Curve path.
2353	The specified file name is too long.
2354	Too many points specified in Curve path.
2355	Too many On / Off commands specified in Curve statement.
2356	The size of internal code is too large in CURVE.
2357	Can not keep the memory for Curve.
2358	Can not keep the memory for CVMove.
2359	The specified file is not a curve file.
2360	The version of curve file is wrong.
2361	The manipulator number of curve file is wrong.
2362	The size of curve file is wrong.
2363	The checksum of curve file is wrong.
2364	Internal code is wrong.
2365	The ARM specified with ARMCLR is currently in use.
2366	ARM 0 cannot be specified with ARMCLR.
2367	The TOOL specified with TLCLR is currently in use.
2368	TOOL 0 cannot be specified with TLCLR.
2369	LOCAL 0 cannot be specified with LocalClr
2370	The ECP specified with ECPCLR is being used.
2371	ECP 0 cannot be specified with ECPCLR.
2401	FEND system error.
2501	There are more nested function calls than allowed.
2502	There are more nested FOR loops than allowed.
2503	There are more nested IF branches than allowed.
2505	There are more nested SELECT structures than allowed.
2506	There are more nested DO loops than allowed.
2507	There are more nested WHILE loops than allowed.
2520	Too many Global variables or Global variable memory space exceeded.
2521	Too many Backup variables or Backup variable memory space exceeded.
2522	The specified global variable has not been declared.
2523	The specified backup variable has not been declared.
2524	The total size of local variables in this task exceeds the limit.
2525	Global variable has already been declared.
2526	The specified global variable has already been declared as of another type.
2527	The specified backup variable has already been declared as of another type.
2528	There is another backup variable already declared with the same name.
2601	Parameter type error.
2602	A corresponding IF statement was not found for an ELSE statement.
2603	A corresponding IF statement was not found for an ELSEIF statement.
2604	A corresponding IF statement was not found for an ENDIF statement.
2605	A corresponding FOR statement was not found for a NEXT statement.
2606	A corresponding FOR statement was not found for an EXIT FOR statement.
2607	A corresponding DO statement was not found for an EXIT DO statement.
2608	You may only enter a string for a string variable.
2609	The specified parameter was not a string variable.

Error #	Message
2610	An expected parameter was specified.
2611	The type of CASE entry does not match to the variable type specified for SELECT.
2612	A corresponding SELECT statement was not found for a CASE statement.
2614	A numeric value should follow "#" in a DECLARE statement.
2700	The specified file or port number is out of range.
2701	The specified file number has already been used.
2702	The specified file is being used in another task.
2703	File number not available.
2705	The specified file cannot be closed.
2708	File seek error.
2709	The specified file was not opened with ROPEN.
2710	The specified file was not opened with WOPEN.
2711	The specified file was not opened with AOPEN.
2712	The specified path name (file name) is too long.
2713	An invalid path name (file name) was specified.
2715	A read-only file was specified.
2716	The specified folder is not empty or does not exist.
2717	The file cannot be copied (Destination file exists or path error or disk error).
2718	The specified file has not been opened.
2719	The function does not support the file.
2720	The file has not been opened in binary mode.
2800	The specified communication port is not available.
2801	The specified communication port is being used in another task.
2802	The data cannot be written to the specified communication port.
2803	The data cannot be read from the specified communication port.
2804	Timeout error in reading data from the specified communication port.
2805	The specified communication port is being used in another task.
2806	Communication port could not be opened.
2807	The specified communication port has not been installed.
2808	The specified communication port is being used in another application.
2809	The specified setting is not supported for the communication port.
2830	The specified network port can not be used.
2831	The specified network port is being used in another task.
2832	The data cannot be written to the specified network Port.
2833	The data cannot be read from the specified network port.
2834	Timeout error in reading data from the specified network port.
2835	The specified network port is being used in another task.
2836	Network Port open error.
2837	Connection has not been established.
2838	The network settings cannot be changed while the network port is being used.
2839	Timeout error for WAITCOM at the specified port.
2840	Expected Server or Client.
2841	Network option has not been installed.
2842	The specified TCP/IP port is being used by an RS232 port.
2851	The specified communication port is invalid.
2852	The selected Baud Rate is invalid.

Error #	Message
2853	The selected Data Bit Length is invalid.
2854	The selected Stop Bit is invalid.
2855	The selected Parity is invalid.
2856	The selected Terminator is invalid.
2857	The selected H/W Flow Control is invalid.
2858	The selected S/W Flow Control is invalid.
2859	The specified Timeout value is out of nominal range.
2860	The specified Port Number is not for the Network Port.
2861	The specified Host Name is invalid.
2862	The specified Port Number is not for the Network Port.
2901	Memory allocation failure for the Local Variable area.
2902	Failed to execute the Interpreter.
2903	Failed to release the Local Variable area.
2904	Failed to allocate the working area for string operations.
2905	Failed to release the working area from string operations.
2906	The Registry cannot be opened.
2907	Failed to read data from the Registry.
2908	Failed to write data into the Registry.
2909	Robot error.
2910	A error occurred with a robot.
3000	The specified Joint No. is out of range [1-6].
3001	The specified value exceeds the parameter's upper limit.
3002	The specified value does not reach the parameter's lower limit.
3003	An invalid value [0] is specified.
3004	An invalid value (A positive value cannot be specified).
3005	An invalid value (A negative value cannot be specified).
3006	The number of parameters given is invalid.
3007	Direct designate position number is wrong.
3008	The specified value is out of valid range [1-6] for ACCEL( ).
3009	The specified value is out of valid range [1-3] for SPEED( ).
3010	The specified value is out of valid range [1-2] for ACCELS( ).
3011	The specified Joint No. is out of valid range [1-6] for FINE( ).
3012	The specified Joint No. is out of valid range [1-6] for JRANGE( ).
3013	The specified Reference Data No. is out of valid range [1-2] for JRANGE( ).
3014	Invalid Arch number (valid range is 0 to 6).
3015	Invalid Arch number (valid range is 0 to 6).
3016	The specified Reference Data No. is out of the valid range [1-2] of ARCH( ).
3017	The specified value for SPEEDS function is out of the valid range.
3018	The specified value is outside the valid range [1-2] of ACCELR( ).
3019	The specified value is outside the valid range [1-3] of PTPBOOST( ).
3022	The current position is undefined before MCAL. Execute MCAL. Check MCORDR if error occurs.
3030	Invalid robot number.
3031	PRM file is broken.
3032	PRM file version is unknown.
3033	The specified ARM No. is invalid (The valid range is 0 to 3).
3034	The specified TOOL No. is invalid.

Error #	Message
3035	ENCRESET was attempted when tasks were executing.
3036	ENCRESET was attempted when MOTOR mode is ON.
3037	The jointNumber was not specified for ENCRESET.
3038	The LOCAL number is invalid.
3039	The ARM number is invalid.
3100	No robots found.
3101	A motion command that was attempted is not supported.
3102	The Motion Control Module status is not correct.
3110	Values have not been defined.
3200	The robot is being used by another user task or internal system task.
3201	The robot is being used by another user task or internal system task.
3228	Error during MMD load.
3300	A motion command was attempted when the safe guard was open.
3301	A task was started when the safe guard was open.
3302	An output command was attempted when the safe guard was open.
3303	Recover was attepmted when motor on is prohibited.
3304	Continue operation was attempted with SafeGuard open.
3305	Continue operation was attempted when recover is required.
3306	Continue operation was attempted during recover.
3307	A motion command was attempted when recover is required.
3308	A motion command was attempted during recover.
3309	Continue or Recover attempted during safeguard open processing.
3310	Teach mode input signal failure. System restart required.
3311	Safe guard input signal failure. System restart required.
3312	Emergency Stop input signal failure. System restart required.
3320	Continue operation attempted too many times.
4003	Communication error within the Motion Control Module.
4004	Event waiting error with the Motion Control Module.
4006	The target point position is above the specified LIMZ value.
4007	The arm reached the limit of motion range.
4008	The current point position or specified LIMZ value is out of motion range.
4009	Timeout error within Motion Control Module.
4010	The specified LOCAL coordinate is not defined.
4013	Motion Control Module internal calculation error.
4016	SFREE was attempted for a prohibited joint.
4017	MCAL/MCORG is attempted when not all the joints are engaged.
4018	Communication error within the Motion Control Module.
4019	The specified command is not mechanically supported.
4021	One of the point position sets specified by LOCAL is too close.
4022	The point data for LOCAL is invalid.
4023	The specified command cannot be executed in motor off condition.
4026	Communication error within the Motion Control Module.
4027	Communication error within the Motion Control Module.
4028	Communication error within the Motion Control Module.
4029	Communication error within the Motion Control Module.
4030	Torque RMS calculation value overflowed.

Error #	Message
4033	The specified command is not supported for the joints with Pulse Generating Board.
4034	The specified command is not supported for this type of robot.
4035	The target position is specified in a CP motion command without spatial transfer of the Arm.
4036	The target position in a CP motion command is not viable (the target position is too close to the current position).
4037	The Arm attributes of the current and target point positions differs in executing a CP control command.
4038	Two point positions for an ARC command are too close.
4039	The three point positions specified by an ARC command are on a straight line.
4040	Communication error with the Pulse Generating Board.
4041	A motion command was attempted into the prohibited area.
4042	Interrupt detection error.
4043	The specified command is not supported for this type of robot or joint.
4044	The specified curve form is not supported.
4045	The specified mode is not supported.
4046	The number of coordinate is outside the range.
4047	Not all point data was specified.
4048	The parallel process was specified before the point designation.
4049	The number of parallel process is outside the range.
4050	Illegal number of points specified.
4051	Local or orientation are not the same for all points in curve.
4052	Not enough memory to process curve file.
4053	Curve file could not be built. Check points.
4054	Curve file error.
4055	Illegal distance between curve points.
4056	The point positions for an CURVE command are too close.
4057	Pulse Generating joint distance is overflowed.
4058	Prohibited command while tracking was executed.
4059	The Encoder Reset command is not allowed when Motor Mode is On.
4060	The specified command is not allowed when Motor Mode is On.
4061	The specified parameter is now used.
4062	Orientation variation is over 360 degrees.
4063	Orientation variation of adjacent point is over 90 degrees.
4064	Automatic correction of the orientation
4065	J6 is unable to be rotated with the same orientation in CP motion.
4066	A motion command has been attempted into a prohibited area.
4068	ROT modifier parameter was used in a CP motion command without rotation of the point orientation.
4069	ECP modifier parameter was used in a CP motion command with invalid current ECP number.
4070	ECP number in the curve file is different from the current ECP number.
4099	Servo error is detected.
4100	The current point and pulse data calculation failed because of communication error in the Motion Control Module.
4101	The current point and pulse data detection failed because of communication error in the Motion Control Module.
4102	The Drive Unit power is not turned on.

Error #	Message
4103	Motion Control Module initialization error
4104	Position timeout of a PG robot joint.
4105	Emergency Connector Connection Failure
4106	Drive unit type setting is not correct
4150	Emergency Stop input signal failure.
4151	Safe guard input signal failure.
4180	The specified robot number is invalid.
4181	The specified robot is being used in another task.
4182	Robot name is too long.
4183	Robot data version error.
4184	There is more than one joint assigned to a single Axis.
4185	There is more than one robot assigned to a single Axis.
4186	Necessary hardware resource was not found.
4187	Communication error with VxD (VSRCMNPK.VXD).
4188	The Joint Interference Matrix is not regular.
4189	Communication error with VxD (VSRCMC.VXD).
4190	Communication error with VxD (VSRCPG.VXD).
4191	The Physical - Logical Pulse Conversion Matrix is not regular.
4192	Communication error with servo module.
4210	The RAS circuit detected the control unit malfunction.
4211	Servo CPU internal RAM error.
4212	Drive Unit dual-port RAM error.
4213	Servo CPU Program-RAM error.
4214	Servo CPU initial command check-sum error.
4215	Servo CPU operation overrun.
4216	Servo real-time command overrun.
4217	Servo real-time command check-sum error.
4218	Servo long-time command overrun.
4219	Servo long-time command check-sum error.
4220	Drive Unit watch-dog timer error.
4221	Driver Unit check error.
4222	Servo CPU external RAM error.
4230	Servo real-time status check-sum error.
4231	Servo long-time status check-sum error.
4232	Free-running counter error with Servo.
4233	Communication error with Servo CPU.
4240	Irregular interrupt is detected.
4241	The detected speed in the low-power state is too fast.
4242	Improper acceleration reference is generated.
4243	Improper speed reference is generated in the high-power state.
4250	The arm reached the limit of motion range.
4251	The arm reached the limit of XY motion range specified by XYLIM.
4252	The arm reached the limit of motion range.
4261	The arm reached the limit of motion range in conveyor tracking.
4262	The arm reached the limit of XY motion range in conveyor tracking.
4263	The arm reached the limit of pulse motion range in conveyor tracking.

Error #	Message
4264	Conveyor speed is too fast.
4267	It is not possible to change the J4Flag attribute during CP motion.
4268	It is not possible to change the J6Flag attribute during CP motion.
4269	It is not possible to change the WRIST attribute during CP motion.
4270	It is not possible to change the ARM attribute during CP motion.
4271	It is not possible to change the ELBOW attribute during CP motion.
4272	The specified point flag is invalid.
4273	The J6Flag was changed during JUMP3 depart motion in conveyor tracking.
4301	The Pulse Generating Board detected a limit signal.
4302	The Pulse Generating Board detected an alarm signal.
4401	The specified conveyor number is illegal.
4402	The specified queue is full.
4403	Continue operation cannot be done in tracking motion.
4404	The specified queue data does not exist.
4405	The conveyor is not correctly initialized.
4406	The specified queue data is outside the set area.
4409	The parameter of the conveyor instruction is invalid.
4410	The conveyor coordinates conversion error occurs.
4411	Communication error within the conveyor modules.
4413	Conveyor tracking starting error.
5000	Malfunction of the Servo Gate-Array.
5001	Encoder signal cable disconnection.
5002	Motor driver is not installed.
5003	Initialization time-out error with incremental encoder.
5004	Initialization time-out error with absolute encoder.
5005	Encoder division setting is invalid.
5006	Absolute encoder character error.
5007	Absolute encoder rotation overflow.
5008	Pulse counter overflow.
5009	Initialization time-out error with Serial Encoder.
5010	Initialization error with Serial Encoder.
5011	Data receive error with Serial Encoder.
5012	Serial Encoder reset execution.
5016	Absolute Encoder backup alarm.
5017	Absolute Encoder check-sum error.
5018	Absolute Encoder battery alarm.
5019	Absolute Encoder absolute error.
5020	Absolute Encoder overspeed.
5021	Absolute Encoder overheat.
5032	Servo alarm A
5042	Position error overflow in the high-power state.
5043	Position error overflow in the low-power state.
5044	Speed error overflow in the high-power state.
5045	Speed error overflow in the low-power state.
5046	Overspeed in the high-power state.
5047	Overspeed in the low-power state.

Error #	Message
5048	Overvoltage.
5049	Overcurrent.
5050	Overspeed in the torque control mode.
5054	Servo overload A.
5055	Servo overload B.
5072	Servo alarm B.
5080	Servo overload C.
5112	Servo alarm C.
5120	Servo overload D.
5136	Absolute Encoder has been initialized.
5137	Absolute Encoder Battery low voltage warning.
5152	Servo alarm D.
6001	Failure to create BASIC DAEMON thread.
6002	Failure to create LED7SEG DAEMON thread.
6003	Failure to create QP DAEMON thread.
6004	Failure to create InTrap DAEMON thread.
6005	Failure to create Jogging DAEMON thread.
6006	Failure to create COMM DAEMON thread.
6007	Failure to create TCPIP DAEMON thread.
6008	Failure to create NAIS DAEMON thread.
6102	TEACH/AUTO key input signal failure.
6103	The specified robot is not found in Remote I/O.
6104	Emergency Stop input signal failure.
6500	There are no answers from the Optional Device.
6501	The Optional Device is busy.
6502	Data cannot be sent/written to the Optional Device.
6503	Settings for the Optional Device cannot be found.
6504	Invalid data was detected during continuous jog execution.
6505	COM3 is not ready.
6506	Failure to create Jogging Motion DAEMON thread.
6507	COM3 Framing error.
6508	COM3 Over Run error.
6509	COM3 Receive Buffer Full error.
6510	COM3 Parity error.
6511	COM3 Other error.
6512	Optional device port Line Buffer Full error.
6701	TCP/IP Port open error.
6901	Unexpected robot data version.
6902	Robot data file Read error.
6903	Robot data file Write error.
6904	Robot data memory allocation error.
6905	Robot data cannot be revised.
7000	SRCMng undefined error.
7103	Too many counters
7350	Failure to set up an asynchronous output.
7351	Failure to change priority of the SPELCT threads.

Error #	Message
7352	Event operation delay check failure.
7353	Failure to create a memory mapped file.
7901	Failure to create a message table.
7902	Failure to obtain area for the program allocation information data.
7903	Failure to obtain the programming area.
7904	Failure to obtain the sub system area.
7905	Failure to obtain the break-point information data area.
7906	Failure to obtain the timer information data area.
7907	Failure to obtain the control block area.
7908	Failure to obtain the robot information data area.
7909	Failure to obtain the local variable data area.
7910	Failure to obtain the DLLFAT area.
7911	The DLLFAT cannot be created.
8000	System point data file is irregular.
8001	System point data file cannot be created.
8002	System point data file cannot be closed.
8003	Robot number is invalid.
8004	Current robot is invalid or system point data error.
8006	Invalid point number.
8009	The specified point data file cannot be opened.
8010	The pallet definition area cannot be obtained.
8011	The pallet definition area cannot be released.
8012	The pallet number is out of range.
8013	The specified pallet data has not been defined.
8014	Pallet division value is out of range.
8015	Coordinate Axis error.
8016	Point flag is invalid.
8017	A point label table cannot be created.
8018	The specified point label cannot be found.
8019	The specified system point data file cannot be found.
8020	The point Data cannot be written to a Point Data file.
8021	Duplicate point label.
8022	The local coordinates of the specified point data are not coincident.
8023	The Point Data Format is invalid.
8024	Current robot is invalid or system point data error.
8100	The specified point label has not been defined.
9009	Cannot complete abort all
9010	Incorrect Value
9011	Task Executing
9012	Same Path
9013	Group Not Found
9014	Filename
9015	Function name
9016	No Main Function
9017	Safeguard Open
9018	IP

Error #	Message
9019	Unknown value
9020	Invalid IO port
9021	Error History
9022	Xqt Main
9023	Invalid Task Number
9024	Already Group
9025	File copy
9026	Overwrote
9027	Already file
9028	Delete Group
9029	Mnp Number
9030	Password
9031	Not Input
9032	Invalid Workmode
9033	Compile
9034	Opr Mode
9035	Console
9036	No Robot
9037	Project not closed
9038	Project already exists
9039	Failed to create project
9040	No Project
9041	Cannot delete current project
9042	Failed to delete project
9043	Compile Error
9044	Failed to set point number
9045	Failed to get point number
9046	IO Number duplication
9047	Not Set
9048	Failed to set RBP
9049	Failed to set point label
9050	Failed to get point label
9051	ROM data file not found
9052	Failed to create MCD file
9053	Failed to create PRM file
9054	Failed to read MCD file
9070	Abort All in progress
9071	Point Data is not defined.
9072	Failed to save file.
9073	Failed to load file.
9074	Enable switch was not closed.
9077	Illegal file name length.
9078	The number of the point data for one manipulator exceeds the limitation.
9079	The number of the point data for all manipulators exceeds the limitation.
9080	The combination of this console device and teaching device cannot be specified.
9081	The controller is in the process of shutdown.

Error #	Message
9082	The teaching device setting is not OLE.
9081	Parameter error in the SRCLoad Initialization function.
9082	Parameter error in the SRCLoad termination function.
9083	Parameter error in the SRCLoad termination function.
9086	Continue or Recover attempted during safeguard open processing.
9100	SRCLoad setting cannot be found.
9101	SRCLoad information error (SRCMng System).
9102	SRCLoad information error (SRCMC).
9103	SRCLoad information error (Input port).
9104	SRCLoad information error (Output port).
9201	The manipulator setting is done.
9202	SRCMng event resource allocation error.
9203	SRCMng event object creation error.
9204	SRCMng setting error (Interrupt Interval).
9205	SRCMng setting error (Intermediate TimeOut).
9206	SRCMng setting error (Priority Default).
9207	SRCMng setting error (Stack Size Default).
9208	SRCMng setting error (I/O Interval).
9209	SRCMng setting error (Interrupt overtime).
9210	SRCMng setting error (CallBack overtime).
9211	SRCMng setting error (Event Operation Delay Overtime).
9212	SRCMng setting error (Timer Count Error(%)).
9213	SRCMng setting error (Timer Check Interrupt Count).
9214	SRCMng setting error (Time Slice).
9215	SRCMng timer interrupt start error.
9216	SRCMng timer interrupt stop error.
9217	SRCMng termination error.
9301	Debug operation start error.
9302	Debug operation end error.
9401	The setting for SRCLoad cannot be found.
9402	SRCLoad memory lock failure.
9403	SRCLoad Main Thread priority value setting error.
9404	SRCLoad re-initialization error.
9701	Failure to obtain memory for SRCMC initialization.
9702	Failure to obtain data on the SRCMC Pulse Generating Board.
9703	Failure to obtain data on the SRCMC Drive Unit.
9704	SRCMC DLL loading error.
9705	Failure to detect the SRCMC initialization function pointer.
9706	SRCMC Module Handle error.
9707	Failure to detect the SRCMC termination function pointer.
9800	SRCMC initialization error.
9880	SRCMC termination error.
9901	Vision calibration error.
9950	Failure to create a Scheme file.