EPSON

EPSON RC+ 7.0 Ver.7.5

SPEL$^+$ Language Reference

Rev.9 ENM23YS6229F

Original instructions

EPSON RC+ 7.0 (Ver.7.5)   SPEL^{+} Language Reference Rev.9

EPSON RC+ 7.0 (Ver.7.5)

SPEL$^+$ Language Reference

Rev.9

FOREWORD

Thank you for purchasing our robot products.
This manual contains the information necessary for the correct use of the EPSON RC+ 7.0 software.
Please carefully read this manual and other related manuals before installing the robot system.
Keep this manual handy for easy access at all times.

The robot system 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. Please note that the basic performance of the product will not be exhibited if our robot system is used outside of the usage conditions and product specifications described in the manuals.

This manual describes possible dangers and consequences that we can foresee. Be sure to comply with safety precautions on this manual to use our robot system safety and correctly.

TRADEMARKS

TRADEMARK NOTATION IN THIS MANUAL

Microsoft® Windows® 8 operating system
Microsoft® Windows® 10 operating system
Microsoft® Windows® 11 operating system
Throughout this manual, Windows 8, Windows 10 and Windows 11 refer to above respective operating systems. In some cases, Windows refers generically to Windows 8, Windows 10 and Windows 11.

NOTICE

MANUFACTURER

SEIKO EPSON CORPORATION

CONTACT INFORMATION

Contact information is described in "SUPPLIERS" in the first pages of the following manual:

Robot System Safety Manual Read this manual first

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.

Summary of SPEL^{+} Commands

The following is a summary of SPEL+ commands.

System Management Commands

Reset	Resets the controller.
SysConfig	Displays controller setup.
SysErr	Returns the latest error status or warning status.
Date	Sets the system date.
Time	Sets system time.
Date$	Returns the system date as a string.
Time$	Returns system time as a string.
Hour	Displays / returns controller operation time.
Stat	Returns controller status bits.
CtrlInfo	Returns controller information.
RobotInfo	Returns robot information.
RobotInfo$	Returns robot text information.
TaskInfo	Returns task information.
TaskInfo$	Returns task text information.
DispDev	Sets the current display device.
EStopOn	Return the Emergency Stop status.
CtrlDev	Returns the current control device number.
Cls	Clears the EPSON RC+ 6.0 Run, Operator, or Command window text area. Clears the TP print panel.
Toff	Turns off execution line display on the LCD.
Ton	Specifies a task which shows an execution line on the LCD.
SafetyOn	Return the Safety Door open status.
Eval	Executes a Command window statement from a program and returns the error status.
ShutDown	Shuts down EPSON RC+ and optionally shuts down or restarts Windows.
TeachOn	Returns the Teach mode status.
WindowsStatus	Returns the Windows startup status.

Robot Control Commands

AIO_TrackingSet	Sets the distance tracking function.
AIO_TrackingStart	Starts the distance tracking function.
AIO_TrackingEnd	Ends the distance tracking function.
AIO_TrackingOn Function	Returns the status of the distance tracking function.
AtHome	Retunes if the current robot orientation is Home position or not.
Calib	Replaces the current arm posture pulse values with the current CalPls values.
CalPls	Specifies and displays the position and orientation pulse values for calibration.
Hofs	Returns the offset pulses used for software zero point correction.
JointAccuracy	Specifies and displays offset value of the axis correction accuracy
HofsJointAccuracy	Returns the offset pulses used for software zero point correction. without changing the axis correction accuracy.
MCal	Executes machine calibration for robots with incremental encoders.
MCalComplete	Returns status of MCal.
MCordr	Specifies and displays the moving joint order for machine calibration Mcal. Required only for robots with incremental encoders.
Power	Sets / returns servo power mode.
Motor	Sets / returns motor status.
MHour Function	Returns the accumulated MOTOR ON time of the robot motors.
SFree	Free joint the specified servo axis.
SLock	Release free joint state of the specified servo axis.
SyncRobots	Start the reserved robot motion.
Jump	Jumps to a point using point to point motion.
Jump3	Jumps to a point using 3D gate motion.
Jump3CP	Jumps to a point using 3D motion in continuous path.
JumpTLZ	Jumps to a point using 3D gate motion.
Arch	Sets / returns arch parameters for Jump motion.
LimZ	Sets the upper Z limit for the Jump command.
LimZMargin	Sets / returns the margin for error detection when the operation starts at the position higher than LimZ value.
Sense	Sets / returns the condition to stop the manipulator above the target coordinate when Sense is specified by Jump command.
JS	Returns status of Sense operation.
JT	Returns the status of the most recent Jump command for the current robot.
Go	Moves 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	Moves 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.
TillOn	Returns the current Till status.
!...!	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.
SpeedFactor	Sets / returns speed for point to point motion commands.
Inertia	Specifies or displays the inertia settings for the robot arm.
Weight	Specifies or displays the weight settings for the robot arm.
Arc	Moves the arm using circular interpolation.
Arc3	Moves the arm in 3D using circular interpolation.
Move	Moves the robot using linear interpolation.
Curve	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.
CVMove	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.
AccelMax	Returns maximum acceleration value limit available for Accel.
Brake	Turns brake on or off for specified joint of the current robot.
Home	Moves robot to user defined home position.
HomeClr	Clears the home position definition.
HomeDef	Returns status of home position definition.
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.
PalletClr	Clears a pallet definition.
Fine	Specifies and displays the positioning error limits. (Unit: pulse)
FineDist	Specifies and displays the positioning error limits (Unit: mm)

FineStatus Function	Returns whether Fine or FineDist is used by the integer.
QP	Sets / returns Quick Pause status.
QPDecelR	Sets the deceleration speed of quick pause for the change of tool orientation during the CP motion.
QPDecelS	Sets the deceleration speed of quick pause in the CP motion.
CP	Sets CP (Continuous Path) motion mode.
Box	Specifies and displays the approach check area.
BoxClr	Clears the definition of approach check area.
BoxDef	Returns whether Box has been defined or not.
Plane	Specifies and displays the approach check plane.
PlaneClr	Clears (undefines) a Plane definition.
PlaneDef	Returns the setting of the approach check plane.
InsideBox	Returns the check status of the approach check area.
InsidePlane	Returns the check status of the approach check plane.
GetRobotInsideBox	Returns a robot which is in the approach check area.
GetRobotInsidePlane	Returns a robot which is in the approach check plane.
Find	Specifies or displays the condition to store coordinates during motion.
FindPos	Returns a robot point stored by Find during a motion command.
PosFound	Returns status of Find operation.
WaitPos	Waits for robot to decelerate and stop at position before executing the next statement while path motion is active.
Robot	Selects the current robot. (Returns the robot number by Robot Function)
RobotModel$	Returns the robot model name.
RobotName$	Returns the robot name.
RobotSerial$	Returns the robot serial number.
RobotType	Returns the robot type.
TargetOK	Returns a status indicating whether or not the PTP (Point to Point) motion from the current position to a target position is possible.
ROTOK Function	Returns whether an ROT modifier parameter can be added when issuing a move command to a destination.
JRange	Sets / returns joint limits for one joint.
Range	Sets limits for all joints.
XYLim	Sets or displays the permissible XY motion range limits for the robot.
XYLimClr	Clears the XYLim definition.
XYLimDef	Returns whether XYLim has been defined or not.
XYLimMode	Sets or displays the XYLim monitor method or returns the status.

XY	Returns a point from individual coordinates that can be used in a point expression.
Dist	Returns the distance between two robot points.
DiffToolOrientation Function	Returns the angle between the coordinate axes of Tool coordinate systems.
DiffPoint Function	Returns the difference between two specified points.
PTPBoost	Specifies or displays the acceleration, deceleration and speed algorithmic boost parameter for small distance PTP (point to point) motion.
PTPBoostOK	Returns whether or not the PTP (Point to Point) motion from a current position to a target position is a small travel distance.
PTPTime	Returns the estimated time for a point to point motion command without executing it.
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.
CR	Sets / returns the R axis coordinate of a point.
CS	Sets / returns the S axis coordinate of a point.
CT	Sets / returns the T 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.
JA	Returns a robot point specified in joint angles.
AglToPls	Converts robot angles to pulses.
DegToRad	Converts degrees to radians.
RadToDeg	Converts radians to degrees.
Joint	Displays the current position for the robot in joint coordinates.
JTran	Perform a relative move of one joint.
PTran	Perform a relative move of one joint in pulses.
RealPls	Returns the pulse value of the specified joint.
RealPos	Returns the current position of the specified robot.
RealAccel Function	Returns the Accel value automatically adjusted by OLAccel.
PPls	Return the pulse position of a specified joint value from a specified point.
LJM Function	Returns the point data with the orientation flags converted to enable least joint motion when moving to a specified point based on the reference point.
AutoLJM	Sets the Auto LJM
AutoLJM Function	Returns the state of the Auto LJM
AutoOrientationFlag	Changes orientation flag of N6-A1000**
AutoOrientationFlag Function	Returns the state of the AutoOrientationFlag

AvoidSingularity	Sets the Singularity avoiding function
AvoidSingularity Function	Returns the state of the Singularity avoiding function
SingularityAngle	Sets the singularity neighborhood angle for the singularity avoiding function
SingularityAngle Function	Returns the singularity neighborhood angle for the singularity avoiding function
SingularitySpeed	Sets the singularity neighborhood speed for the singularity avoiding function
SingularitySpeed Function	Returns the singularity neighborhood speed for the singularity avoiding function
SingularityDist	Sets the singularity neighborhood distance necessary for the singularity avoiding function.
SingularityDist Function	Returns the singularity neighborhood distance necessary for the singularity avoiding function.
AbortMotion	Aborts a motion command and puts the running task in error status.
Align Function	Returns point data converted to align robot orientation with the nearest coordinate axis in local coordinate system.
AlignECP Function	Returns point data converted to align robot orientation with a nearest coordinate axis in ECP coordinate system.
SoftCP	Sets / returns SoftCP motion mode.
SoftCP Function	Returns the status of SoftCP motion mode.
Here	Teach a robot point at the current position.
Where	Displays current robot position data.
PerformMode	Sets the mode of the robot.
PerformMode Function	Returns the robot performance mode number.
VSD	Sets the variable speed CP motion of SCARA robots.
VSD Function	Returns the variable speed CP motion setting of SCARA robots.
CP_Offset	Sets the offset time to start the subsequent motion command when executing CP On.
CP_Offset Function	Returns the offset time to start the subsequent motion command when executing CP On.
AvgSpeedClear	Clears and initializes the average of the joint speed.
AvgSpeed	Displays the average of the joint speed.
AvgSpeed Function	Returns the average value of the joint speed.
PeakSpeedClear	Clears and initializes the peak speed for one or more joints.
PeakSpeed	Displays the peak speed values for the specified joint.
PeakSpeed Function	Returns the peak speed for the specified joint.

Torque Commands

TC	Returns the torque control mode setting and current mode.
TCSpeed	Specifies the speed limit in the torque control.
TCLim	Specifies the torque limit of each joint for the torque control mode.
RealTorque	Returns the current torque instruction value of the specified joint.
ATCLR	Clears and initializes the average torque for one or more joints.
ATRQ	Displays the average torque for the specified joint.
PTCLR	Clears and initializes the peak torque for one or more joints.
PTRQ	Displays the peak torque for the specified joint.
OLAccel	Sets up the automatic adjustment of acceleration/deceleration that is adjusted.
OLRate	Display overload rating for one or all joints for the current robot.
LimitTorque	Sets / returns the upper torque value in High power mode.
LimitTorque Function	Returns the LimitTorque setting value.
LimitTorqueLP	Sets / returns the upper limit torque value in Low power mode.
LimitTorqueLP Function	Returns the LimitTorqueLP setting value.
LimitTorqueStop	Specifies /returns whether or not to stop the robot when torque reaches the upper limit in High power mode.
LimitTorqueStop Function	Returns the LimitTorqueStop setting value.
LimitTorqueStopLP	Specifies / returns whether or not to stop the robot when torque reaches the upper limit in Low power mode.
LimitTorqueStopLP Function	Returns the LimitTorqueStopLP setting value.

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.
MemInW	Returns the status of the specified memory I/O word port. Each word port contains 16 memory I/O bits.
MemOutW	Simultaneously sets 16 memory I/O bits.

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	Receives input data from the display device and stored in a variable(s).
InReal	Reads an input data of 2 words (32 bits) as a floating-point data (IEEE754 compliant) of 32 bits.
Print	Display characters on current display window.
Line Input	Input a string from the current display window.
Input #	Allows string or numeric data to be received from a file, communications port, or database and stored in one or more variables.
Print #	Outputs data to the specified file, communications port, database, or device.
Line Input #	Reads data of one line from a file, communication port, database, or the device.
Lof	Checks whether the specified RS-232 or TCP/IP port has any lines of data in its buffer.
SetIn	For Virtual IO, sets specified input port (8 bits) to the specified value.
SetInW	For Virtual IO, sets specified input word (16 bits) to the specified value.
SetSw	For Virtual IO, sets specified input bit to the specified value.
IOLabel$	Returns the I/O label for a specified input or output bit, byte, or word.
IONumber	Returns the I/O number of the specified I/O label.
IODef	Returns whether the specified I/O label is defined.
OpenCom	Open an RS-232 communication port.
OpenCom Function	Acquires the task number that executes OpenCom.
CloseCom	Close the RS-232C port that has been opened with OpenCom.
SetCom	Sets or displays parameters for RS-232C port.
ChkCom	Returns number of characters in the reception buffer of a communication port
OpenNet	Open a TCP/IP network port.
OpenNet Function	Acquires the task number that executes OpenNet.
OutReal	Output the output data of real value as the floating-point data (IEEE754 compliant) of 32 bits to the output port 2 words (32 bits).
CloseNet	Close the TCP/IP port previously opened with OpenNet.
SetNet	Sets parameters for a TCP/IP port.
ChkNet	Returns number of characters in the reception buffer of a network port
WaitNet	Wait for TCP/IP port connection to be established.

Read	Reads characters from a file or communications port.
ReadBin	Reads binary data from a file or communications port.
Write	Writes characters to a file or communication port without end of line terminator.
WriteBin	Writes binary data to a file or communications port.
InputBox	Displays a prompt in a dialog box, waits for the operator to input text or choose a button, and returns the contents of the box.
MsgBox	Displays a message in a dialog box and waits for the operator to choose a button.
RunDialog	Runs an EPSON RC+ dialog from a SPEL$^+$ program.
LatchEnable	Enable / Disable the latch function for the robot position by the R-I/O input.
LatchState Function	Returns the latch state of robot position using the R-I/O.
LatchPos Function	Returns the robot position latched using the R-I/O input signal.
SetLatch	Sets the latch function of the robot position using the R-I/O input.
AIO_In Function	Reads analog value form analog I/O input channel.
AIO_InW Function	Reads one word input data from analog I/O input channel.
AIO_Out	Output analog value on the analog I/O output channel.
AIO_Out Function	Returns the output state of analog I/O output channel.
AIO_OutW	Output the one word data to analog I/O output channel.
AIO_OutW Function	Returns the output state by one word of analog I/O output channel.
AIO_Set	Output the speed information to analog I/O output channel.
AIO_Set Function	Returns setting information of robot speed output which is set on optional analog I/O output channel.

Point Management Commands

ClearPoints	Clears all point data in memory.
LoadPoints	Loads point data from a file in memory.
SavePoints	Saves point data to a file in memory.
ImportPoints	Imports a point file into the current project for the specified robot.
ExportPoints	Exports a point file to the specified path in the PC.
P#	Defines a specified point.
PDef	Returns the definition status of a specified point.
PDel	Deletes specified position data.
PLabel	Defines a label for a specified point.
PLabel$	Returns the point label associated with a point number.
PNumber	Returns the point number associated with a point label.
PList	Displays point data in memory for the current robot.
PLocal	Sets the local attribute for a point.
PDescription	Define a description of specified point data.
PDescription$	Returns description of point that defined to the specified point number

WorkQue_Add	Adds the work queue data (point data and user data) to the specified work queue.
WorkQue_AutoRemove	Sets the auto delete function to the specified work queue.
WorkQue_AutoRemove Function	Returns the state of the auto delete function set to the work queue.
WorkQue_Get Function	Returns the point data from the specified work queue.
WorkQue_Len Function	Returns the number of the valid work queue data registered to the specified work queue.
WorkQue_List	Displays the work queue data list (point data and user data) of the specified work queue
WorkQue_Reject	Sets and displays the minimum distance for double registration prevention of the point data to the specified work queue
WorkQue_Reject Function	Returns the distance of the double registration prevention set to the specified work queue
WorkQue_Remove	Deletes the work queue data (point data and user data) from the specified work queue
WorkQue_Sort	Sets and displays the Sort type of the specified work queue
WorkQue_Sort Function	Returns the Sort type of the specified work queue
WorkQue_UserData	Resets and displays the user data (real number) registered to the specified work queue
WorkQue_UserData Function	Returns the user data (real number) registered to the specified work queue

Coordinate Change Commands

AreaCorrection Function	Returns point at which correction was made using correction area
AreaCorrectionClr	Clears correction area
AreaCorrectionDef	Returns correction area settings
AreaCorrectionInv Function	Returns corrected points to their original condition
AreaCorrectionOffset Function	Returns points relatively displaced from corrected points
AreaCorrectionSet	Sets and displays correction area
Arm	Sets / returns current arm.
ArmSet	Defines an arm.
ArmDef	Returns status of arm definition.
ArmClr	Clears an arm definition.
ArmCalib	Sets and returns enable or disable of arm length calibration.
ArmCalibClr	Clears arm length calibration.
ArmCalibDef	Returns status of arm length calibration.
ArmCalibSet	Defines and displays arm length calibration.
Tool	Sets / returns the current tool number.
TLSet	Defines or displays a tool coordinate system.
TLDef	Returns status of tool definition.
TLClr	Clears a tool definition.
ECP	Sets / returns the current ECP number.

ECPSet	Defines or displays an external control point.
ECPDef	Returns status of ECP definition.
ECPClr	Clears an ECP definition.
Base	Defines and displays the base coordinate system.
Local	Define a local coordinate system.
LocalDef	Returns status of local definition.
LocalClr	Clears (undefines) a local coordinate system.
Elbow	Sets / returns elbow orientation of a point.
Hand	Sets / returns hand (arm) 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.
J1Flag	Sets / returns the J1Flag setting of a point.
J2Flag	Sets / returns the J2Flag orientation of a point.
J1Angle	Returns the J1Angle attribute of a point.
J4Angle	Returns the J4Angle attribute of a point.
VxCalib	Creates the calibration data.
VxTrans	Converts the pixel coordinates to the robot coordinates and returns the converted the point data.
VxCalInfo	Returns the calibration completion status / calibration data.
VxCalDelete	Deletes the calibration data.
VxCalSave	Saves the calibration data to the file.
VxCalLoad	Loads the calibration data from the file.

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.
Do...Loop	Do...Loop construct.
Declare	Declares an external function in a dynamic link library (DLL).
Trap	Specify a trap handler.
OnErr	Defines an error handler.
Era	Returns the robot joint number for last error.
Erf$	Returns the function name for last error.

Erl	Returns the line number of error.
Err	Returns the error number.
Ert	Returns the task number of error.
Errb	Returns the robot number of error.
ErrMsg$	Returns the 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.
ErrorOn	Returns the error status of the controller.
Error	Generates a user error.
EResume	Resumes execution after an error-handling routine is finished.
PauseOn	Returns the pause status.
Exit	Exits a loop construct or function.

Program Execution Commands

Xqt	Execute a task.
Pause	Pause all tasks that have pause enabled.
Cont	Resumes the controller after a Pause statement has been executed and continues the execution of all tasks.
Halt	Suspend a task.
Quit	Quits a task.
Resume	Resume a task in the halt state.
MyTask	Returns current task.
TaskDone	Returns the completion status of a task.
TaskState	Returns the current state of a task.
TaskWait	Waits to for a task to terminate.
Restart	Restarts the current main program group.
Recover	Executes safeguard position recovery and returns status.
RecoverPos	Returns the position where a robot was in when safeguard was open.
StartMain	Executes the main function from a background task.

Pseudo Statements

#define	Defines a macro.
#ifdef ... #endif	Conditional compile.
#ifndef ... #endif	Conditional compile.
#include	Include a file.
#undef	Undefines an identifier previously defined with #define.

File Management Commands

ChDir	Changes and displays the current directory.
ChDisk	Sets the object disk for file operations.
MkDir	Creates a subdirectory on a controller disk drive.
RmDir	Removes an empty subdirectory from a controller disk drive.
RenDir	Rename a directory.
FileDateTime$	Returns the date and time of a file.
FileExists	Checks if a file exists.
FileLen	Returns the size of a file.
FolderExists	Checks if a folder exists.
FreeFile	Returns / reserves a file number that is currently not being used.
Del	Deletes one or more files.
Copy	Copies a file to another location.
Rename	Renames a file.
AOpen	Opens file in the appending mode.
BOpen	Opens file in binary mode.
ROpen	Opens a file for reading.
Uopen	Opens a file for read / write access.
WOpen	Opens a file for writing.
Input #	Allows string or numeric data to be received from a file, communications port, or database and stored in one or more variables.
Print #	Outputs data to the specified file, communications port, database, or device.
Line Input #	Reads data of one line from a file, communication port, database, or the device.
Read	Reads characters from a file or communications port.
ReadBin	Reads binary data from a file or communications port.
Write	Writes characters to a file or communication port without end of line terminator.
WriteBin	Writes binary data to a file or communications port.
Seek	Changes position of file pointer for a specified file.
Close	Closes a file.
Eof	Returns end of file status.
ChDrive	Changes the current disk drive for file operations.
CurDir$	Returns a string representing the current directory.
CurDrive$	Returns a string representing the current drive.
CurDisk$	Returns a string representing the current disk.
Flush	Writes a file's buffer into the file.

Fieldbus Commands

FbusIO_GetBusStatus	Returns the status of the specified Fieldbus.
FbusIO_GetDeviceStatus	Returns the status of the specified Fieldbus device.
FbusIO_SendMsg	Sends an explicit message to a Fieldbus device and returns the reply.

Numeric Value Commands

Ctr	Return the value of a counter.
CTReset	Resets a counter.
ElapsedTime	Measures a takt time.
ResetElapsedTime	Resets and starts a takt time measurement timer.
Tmr	Returns the value of a timer.
TmReset	Resets a timer to 0.
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.
BClr	Clears one bit in a number and return the new value.
BSet	Sets a bit in a number and returns the new value.
BTst	Returns the status of 1 bit in a number.
BClr64	Clears one bit in a number and return the new value.
BSet64	Sets a bit in a number and returns the new value.
BTst64	Returns the status of 1 bit in a number.
Fix	Returns the integer portion of a real number.
Hex	Returns a string representing a specified number in hexadecimal format.
Randomize	Initializes the random-number generator.
Redim	Redimension an array at run-time.
Rnd	Return a random number.
UBound	Returns the largest available subscript for the indicated dimension of an array.

String Commands

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.
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.
FmtStr$	Format a number or string.
InStr	Returns position of one string within another.
Tab$	Returns a string containing the specified number of tabs characters.

Logical Operators

And	Performs logical and bitwise AND operation.
Or	Or operator.
LShift	Shifts bits to the left.
LShift64	Shifts bits to the left.
Mod	Modulus operator.
Not	Not operator.
RShift	Shifts bits to the right.
RShift64	Shifts bits to the right.
Xor	Exclusive Or operator.
Mask	Performs bitwise AND operation in Wait statements.

Variable commands

Boolean	Declares Boolean variables.
Byte	Declares byte variables.
Double	Declares double variables.
Global	Declares global variables.
Int32	Declares 4-byte integer variables.
Integer	Declares 2-byte integer variables.
Long	Declares long integer variables.
Int64	Declares 8-byte integer variables.
Real	Declares real variables.
Short	Declares 2-byte integer variables.
String	Declares string variables.
UByte	Declares unsigned integer variables.
UInt32	Declares unsigned 4-byte integer variables.
UShort	Declares unsigned 2-byte integer variables.
UInt64	Declares unsigned 8-byte integer variables.

Security Commands

GetCurrentUser$	Returns the current EPSON RC+ user.
Login	Log into EPSON RC+ 6.0 as another user.

Conveyor Tracking Commands

Cnv_AbortTrack	Aborts tracking motion to a conveyor queue point.
Cnv_Accel Function	Returns acceleration and deceleration for the conveyor.
Cnv_Accel	Sets acceleration and deceleration for the conveyor.
Cnv_AccelLim	Sets limit of acceleration and deceleration after the conveyor tracked.
Cnv_AccelLim Function	Returns limit of acceleration and deceleration after the conveyor tracked.
Cnv_Adjust	Sets whether operate to tracking delay of conveyor.
Cnv_AdjustClear	Clear adjustment of tracking delay of conveyor.
Cnv_AdjustGet Function	Returns adjustment of tracking delay of conveyor.
Cnv_AdjustSet	Sets adjustment of tracking delay of conveyor.
Cnv_Downstream Function	Returns the downstream limit for the specified conveyor.
Cnv_Downstream	Sets the downstream limit for the specified conveyor.
Cnv_Fine Function	Returns the current Cnv_Fine setting.
Cnv_Fine	Sets the value of Cnv_Fine for one conveyor.
Cnv_Flag Function	Returns the tracking state of the tracking abort line.
Cnv_Mode Function	Returns the setting mode value of the conveyor
Cnv_Mode	Sets the setting mode value of the conveyor
Cnv_Name$ Function	Returns the name of the specified conveyor.
Cnv_Number Function	Returns the number of a conveyor specified by name.
Cnv_OffsetAngle	Sets the offset value for the conveyor queue data.
Cnv_OffsetAngle Function	Returns the offset value of the conveyor queue data.
Cnv_Point Function	Returns a robot point in the specified conveyor's coordinate system derived from sensor coordinates.

Cnv_PosErr Function	Returns deviation in current tracking position compared to tracking target.
Cnv_PosErrOffset	Sets an offset value to correct the deviation in current tracking position compared to tracking target.
Cnv_Pulse Function	Returns the current position of a conveyor in pulses.
Cnv_QueAdd	Adds a robot point to a conveyor queue.
Cnv_QueGet Function	Returns a point from the specified conveyor's queue.
Cnv_QueLen Function	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 and displays the queue reject distance for a conveyor.
Cnv_QueReject Function	Returns the current part reject distance for a conveyor.
Cnv_QueRemove	Removes items from a conveyor queue.
Cnv_QueUserData	Sets and displays user data associated with a queue entry.
Cnv_QueUserData Function	Returns the user data value associated with an item in a conveyor queue.
Cnv_RobotConveyor Function	Returns the conveyor being tracked by a robot.
Cnv_Speed Function	Returns the current speed of a conveyor.
Cnv_Trigger	Latches current conveyor position for the next Cnv_QueAdd statement.
Cnv_Upstream Function	Returns the upstream limit for the specified conveyor.
Cnv_Upstream	Sets the upstream limit for the specified conveyor.

Force Sensing Commands

Force_Calibrate	Sets zero offsets for all axes for the current force sensor.
Force_ClearTrigger	Clears all trigger conditions for the current force sensor.
Force_GetForces	Returns the forces and torques for all force sensor axes in an array.
Force_GetForce Function	Returns the force for a specified axis.
Force_Sensor	Sets the current force sensor for the current task.
Force_Sensor Function	Returns the current force sensor for the current task.
Force_SetTrigger	Sets the force trigger for the Till command.

DB Commands

CloseDB	Close the database that has been opened with the OpenDB command and releases the file number.
DeleteDB	Deletes data from the table in the opened database.
OpenDB	Opens a database or Excel workbook.
SelectDB	Searches the data in the table in an opened database.
UpdateDB	Updates data of the table in the opened database.

PG Commands

PG_FastStop	Stops the PG axes immediately.

PG_LSpeed	Sets the pulse speed of the time when the PG axis starts accelerating and finishes decelerating.
PG_Scan	Starts the continuous spinning motion of the PG robot axes.
PG_SlowStop	Stops slowly the PG axis spinning continuously.

Collision Detection Commands

CollisionDetect	Enables or disables the collision detection.
CollisionDetect Function	Returns the setting value of CollisionDetect command.

Parts Consumption Commands

HealthCalcPeriod	Sets the calculation period of parts consumption control.
HealthCalcPeriod Function	Returns the calculation period of parts consumption control.
HealthCtrlAlarmOn Function	Returns the status of the parts consumption alarm for the specified Controller parts.
HealthCtrlInfo	Displays the remaining months before the recommended replacement time for the specified Controller parts.
HealthCtrlInfo Function	Returns the remaining months before the recommended replacement time for the specified Controller parts.
HealthCtrlRateOffset	Sets the offset for the consumption rate of the specified parts.
HealthCtrlReset	Clears the consumption rate for the specified Controller parts.
HealthCtrlWarningEnable	Sets enable or disable the parts consumption alarm notification of the Controller parts.
HealthCtrlWarningEnable Function	Returns enable or disable the parts consumption alarm notification of the controller part.
HealthRateCtrlInfo Function	Returns the consumption rate of the specified Controller parts.
HealthRateRBInfo Function	Returns the consumption rate for the specified robot parts.
HealthRBAlarmOn Function	Returns the status of the parts consumption alarm for the specified robot parts.
HealthRBAnalysis	Displays the analysis result regarding the parts consumption (remaining months before the recommended parts replacement time) for the specified robot parts.
HealthRBAnalysis Function	Returns the analysis result regarding the parts consumption (remaining months before the recommended parts replacement time) for the specified robot parts.
HealthRBDistance	Displays the driving amount of the specified joint.
HealthRBDistance Function	Returns the driving amount of the specified joint.
HealthRBInfo	Displays the remaining months before the recommended replacement time for the specified robot parts.
HealthRBInfo Function	Returns the remaining months before the recommended replacement time for the specified robot parts.
HealthRBRateOffset	Sets the offset for the consumption rate of the specified parts.
HealthRBReset	Clears the consumption rate for the specified robot parts.
HealthRBSpeed	Displays the average speed of the specified joint.

HealthRBSpeed Function	Returns the average of the absolute speed of the specified joint.
HealthRBStart	Starts analysis of the parts consumption for the specified robot parts.
HealthRBStop	Stops analysis of the parts consumption for the specified robot parts.
HealthRBTRQ	Displays the torque value of the specified joint.
HealthRBTRQ Function	Returns the torque value of the specified joint.
HealthRBWarningEnable	Sets enable or disable the parts consumption alarm notification of the robot parts.
HealthRBWarningEnable Function	Returns enable or disable the parts consumption alarm notification of the robot parts.

Simulator Commands

SimSet	Sets the object settings, operations, and robot motions of simulator.
SimGet	Acquires the setting values of simulator object.

Hand Commands (For details, see Hand Function Manual.)

Hand_On	Gripper: Execute a gripping of hand. Electric screwdriver: Execute a tightening screw of hand.
Hand_On Function	Gripper: Returns “True” when hand is gripping state. Electric screwdriver: Returns “True” when hand is complete to tighten screws.
Hand_Off	Gripper: Execute releasing hand. Electric screwdriver: Execute loosen screw of the hand.
Hand_Off Function	Gripper: Returns “True” when hand is releasing state. Electric screwdriver: Returns “True” when hand is complete to loosen screws.
Hand_TW Function	Returns “True” when the most recent Hand_On command and Hand_Off command is time out.
Hand_Def Function	Returns “True” when hand is defined.
Hand_Type Function	Returns type number of the hand.
Hand_Label$ Function	Returns label of the hand.
Hand_Number Function	Returns hand number of the hand.

Safety Function Commands (For details, see Robot Controller Safety Function Manual.)

SF_GetParam Function	Returns information on the safety function parameters.
SF_GetParam$ Function	Returns text information on the safety function parameters.
SF_GetStatus Function	Returns the status bit of the safety function.
SF_LimitSpeedS Statement	Sets and displays the speed adjustment value for the function that adjusts the speed at the position set by the Tool command when SLS is enabled.
SF_LimitSpeedS Function	Returns the speed adjustment value of the function that adjusts the speed at the position set by the Tool command when SLS is enabled.
SF_LimitSpeedSEnable Statement	Sets and displays On/Off for the function that adjusts the speed of the position set by the Tool command when SLS is enabled.
SF_LimitSpeedSEnable Function	Returns the status of the function that adjusts the speed at the position set by the Tool command when SLS is enabled.
SF_PeakSpeedS Statement	Displays the peak speed value for the speed monitoring point.
SF_PeakSpeedS Function	Returns the peak speed value for the speed monitoring point.
SF_PeakSpeedSClear Statement	Clears and initializes the peak speed value for the speed monitoring point.
SF_RealSpeedS Statement	Displays the current speed of the speed monitoring point.
SF_RealSpeedS Function	Returns the current speed of the speed monitoring point.

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.
Note	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.

Operators

The following table shows the operators for the SPEL$^+$ language.

Keyword or Symbol	Example	Description
+	A+B	Addition
–	A-B	Subtraction
*	A*B	Multiplication
/	A/B	Division
**	A**B	Exponentiation
=	A=B	Equal
>	A>B	Greater than
<	A<B	Less than
>=	A>=B	Greater than or equal
<=	A<=B	Less or than equal
<>	A<>B	Not equal
And	A And B	Performs logical and bitwise AND operation.
Mod	A Mod B	Returns the remainder obtained by dividing a numeric expression by another numeric expression.
Not	Not A	Performs logical or bitwise negation of the operand.
Or	A Or B	Performs the bitwise Or operation on the values of the operands.
Xor	A Xor B	Performs the bitwise Xor operation on the values of the operand.

Priority Order of the Operators

The operators are processed in programs in the following order.

Priority level	Operator	Example	Description
1	()	(A+B)	Brackets
2	**	A**B	Exponentiation
3	*	A*B	Multiplication
	/	A/B	Division
4	Mod	A Mod B	Returns the remainder obtained by dividing a numeric expression by another numeric expression.
5	+	A+B	Addition
	-	A-B	Subtraction
6	=	A=B	Equal
	<>	A<>B	Not equal
	<	A<B	Less than
	>	A>B	Greater than
	<=	A<=B	Less or than equal
	>=	A>=B	Greater than or equal
7	Not	Not A	Performs logical or bitwise negation of the operand.
8	And	A And B	Performs logical and bitwise AND operation.
9	Or	A Or B	Performs the bitwise Or operation on the values of the operands.
10	Xor	A Xor B	Performs the bitwise Xor operation on the values of the operand.

!...! Parallel Processing

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, BGo, BMove, TGo, TMove.
statements	Any valid parallel processing I/O statement(s) which can be executed during motion. (See the 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 "D*n*" 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. "D*n*" 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* OpBCD *p,q* OutW *p,d*	Outputs data "*d*" to output port "*p*".
MemOut *p, d* MemOutW *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)
Wait other conditions	Wait other than the above two patterns are available. Refer to *Wait Statement* for details.
Print	Prints data to the display device.
Print #	Prints data to the specified communications port.
External functions	Executes the external functions declared with Declare statement.
Hand_Off *n* Hand_Off *n*	Executes Hand_On/Hand_Off operation of hand number "n".

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.

When the Till statement is used to stop the arm before completing the intended motion

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

When the AbortMotion statement or Trap is used to stop the arm before completing the motion

After the arm stops at an intermediate travel position, D statement cannot be executed.

Specifying "*n*" near 100% can cause path motion to decelerate

If a large value of "*n*" is used during CP 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.
It should be noted that execution of parallel processing statements which are used with the Jump3 statement begins after the depart motion has completed and ends at the start of approach motion.

The Here statement and Parallel Processing

You cannot use both of the Here statement and parallel processing in one motion command like this:

```
Go Here :Z(0) ! D10; MemOn 1 !
```

Be sure to change the program like this:

```
P999 = Here
Go P999 Here :Z(0) ! D10; MemOn 1 !
```

See Also

Arc, Arc3, Go, Jump, Jump3, Jump3CP, Move, BGo, BMove, TGo, TMove

!...! 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

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.

Note

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

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

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.

Note

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

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 and global variable declarations.

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

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.

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
```

#undef

Undefines an identifier previously defined with #define.

Syntax

#undef *identifier*

Parameters

identifier	Keyword used in a previous #define statement.

See Also

#define, #ifdef, #ifndef

AbortMotion Statement

Aborts a motion command and puts the running task in error status.
This command is for the experienced user and you need to understand the command specification before use.

Syntax

AbortMotion {*robotNumber* | All }

Parameters

robotNumber	Robot number that you want to stop the motion for.
All	Aborts motion for all robots.

Description

Depending on the robot status when AbortMotion is executed, the result is different as follows.
In each case, hook an error and handle the error processing with OnErr to continue the processing.
Error 2999 can use the constant ERROR_DOINGMOTION.
Error 2998 can use the constant ERROR_NOMOTION.

Write a program not to execute AbortMotion more than twice before executing the continuous execution (Cont).

When the robot is executing the motion command

The robot promptly pauses the arm motion immediately and cancels the remaining motions.
Error 2999 (ERROR_DOINGMOTION) occurs in the task which was running the motion command for the robot.
For the following motion commands, the robot directly moves to the next position from the point where it was paused.

When the robot has been paused immediately

When AbortMotion is executed, the remaining motion is canceled.
Error 2999 (ERROR_DOINGMOTION) occurs in the task which was running the motion command for the robot when specifying the Cont statement.
For the following motion commands, the robot directly moves to the next position from the point where it was paused.

When the robot is in WaitRecover status (Safeguard Open)

When AbortMotion is executed, the remaining motion is canceled.
The following motions can be selected with the Recover command flags.

When executing “Recover robotNumber, WithMove”, the robot motors turn on and the recovery motion is executed.
When Cont is executed, error 2999 (ERROR_DOINGMOTION) occurs in the task which was running the motion command for the robot.
For the following motion commands, the robot directly moves to the next position from the point where it was paused.

When executing “Recover robotNumber, WithoutMove”, the robot motors turn on.
When Cont is executed, error 2999 (ERROR_DOINGMOTION) occurs in the task which was running the motion command for the robot.
For the following motion commands, the robot directly moves to the next position from the point where it was paused, without the recovery motion.

When the robot is executing commands other than motion commands

Error 2998 (ERROR_NOMOTION) occurs in the task which was previously running the motion command for the robot. When the task is waiting with Wait or Input commands, the task is aborted promptly and error 2998 occurs.

When executing a motion command with CP On and a program has no more motion commands, error 2998 occurs even if the robot is running.

When the robot is not running from a program (task)

An error occurs.

Notes

About the Controllers to use

It cannot be used with T/VT series.

See Also

OnErr, Recover, Till

AbortMotion Statement Example

When memory I/O #0 turns on, AbortMotion is executed and the robot goes back to the home position.

```
Function main
   Motor On
   Xqt sub, NoEmgAbort
   OnErr GoTo errhandle

   Go P0
   Wait Sw(1)
   Go P1

   Quit sub
   Exit Function

errstart:
   Home
   Quit sub
   Exit Function

errhandle:
   Print Err
   If Err = ERROR_DOINGMOTION Then
      Print "Robot is moving"         ' Executing Go P0 or Go P1
      EResume errstart
   ElseIf Err = ERROR_NOMOTION Then
      Print " Robot is not moving "   ' Executes Wait Sw(1)
      EResume errstart
   EndIf

   Print "Error Stop"                 ' Other error occurs
   Quit All
Fend

Function sub
   MemOff 0
   Wait MemSw(0)
   AbortMotion 1
   MemOff 0
Fend
```

Abs Function

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 command window using the Print instruction.

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

Accel Statement

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 1 or more representing a percentage of maximum acceleration rate.
decel	Integer expression 1 or more representing a percentage of the maximum deceleration rate.
departAccel	Depart acceleration for Jump. Valid Entries are 1 or more. Optional. Available only with Jump command.
departDecel	Depart deceleration for Jump. Valid Entries are 1 or more. Optional. Available only with Jump command.
approAccel	Approach acceleration for Jump. Valid Entries are 1 or more. Optional. Available only with Jump command.
approDecel	Approach deceleration for Jump. Valid Entries are 1 or more. Optional. Available only with Jump command.

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 1 or more. This number represents a percentage of the maximum acceleration (or deceleration) allowed. Usually, the maximum value is 100. However, some robots allow setting larger than 100. Use AccelMax function to get the maximum value available for Accel.

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:

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

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.

Accel setting larger than 100

Usually, the maximum value is 100. However, some robots allow setting larger than 100.
In general use, Accel setting 100 is the optimum setting that maintains the balance of acceleration and vibration when positioning. However, you may require an operation with high acceleration to shorten the cycle time by decreasing the vibration at positioning. In this case, set the Accel to larger than 100. Except in some operation conditions, the cycle time may not change by setting Accel to larger than 100.

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.

<Example 1>

```
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>

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

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 1% or more

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
SRVJump pick
' Restore previous settings
Accel currAccel, currDecel
```

AccelMax Function

Returns maximum acceleration value limit available for Accel.

Syntax

AccelMax*(maxValueNumber)*

Parameters

maxValueNumber	Integer expression which can have the following values:
	1: acceleration maximum value
	2: deceleration maximum value
	3: depart acceleration maximum value for Jump
	4: depart deceleration maximum value for Jump
	5: approach acceleration maximum value for Jump
	6: approach deceleration maximum value for Jump

Return Values

Integer 1% or more

See Also

Accel

AccelMax Function Example

This example uses the AccelMax function in a program:

```
' Get maximum accel and decel
Print AccelMax(1), AccelMax(2)
```

AccelR Statement

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 (0.1 to 5000).
decel	Real expression in degrees / second^2 (0.1 to 5000).

Valid entries range of the parameters

	accel / *decel*
VT series	0.1 to 1000
C series, N series T series, G series, GX series RS series, LS-B series	0.1 to 5000

(deg/sec^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 when any one of the following conditions occurs:

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or QuitAll stops tasks

See Also

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

AccelR Statement Example

```
AccelR 360, 200
```

AccelR Function

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

Sets the acceleration and deceleration rates for the Straight Line and Continuous Path robot motion instructions such as Move, Arc, Arc3, Jump3, CVMove, etc.

Syntax

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

Parameters

accel	Real expression 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 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.

Valid entries range of the parameters (mm/sec^2)

	accel / *decel* *departAccel* / *departDecel* *approAccel* / *approDecel*
N2	0.1 to 5000
LS20-B, T series, VT series	0.1 to 10000
C4-*901**	0.1 to 15000
C4-*601**, C8-*1401**, G series, GX series, RS series, LS3-B, LS6-B, LS10-B C8-*701**W, C8-*901**W, N6, C12	0.1 to 25000
C8-*701**, C8-*701**R, C8-*901**, C8-*901**R	0.1 to 35000

Return Values

Displays Accel and Decel values when used without parameters
When displays Accel and Decel values, displays adjusted Accel and Decel values according to the currently configured hand weight, for each accel, decel, departAccel, departDecel, approAccel, approDecel.

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 when any one of the following conditions occurs:

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or QuitAll stops tasks

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.

Upper limit value

The AccelS upper limit value of SCARA robots (including RS series manipulators) varies depending on Weight setting and the position of the spline unit. For details, refer to the Manipulator manuals (ACCELS Setting for CP Motions).

The AccelS upper limit value of 6-Axis robots varies depending on Weight setting. For details, refer to the Manipulator manuals (Specifications).

See Also

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

AccelS Statement 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
```

AccelS Function

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 7: acceleration value adjusted by hand weight 8: deceleration value adjusted by hand weight 9: depart acceleration value for Jump3, Jump3CP adjusted by hand weight 10: depart deceleration value for Jump3, Jump3CP adjusted by hand weight 11: approach acceleration value for Jump3, Jump3CP adjusted by hand weight 12: approach deceleration value for Jump3, Jump3CP adjusted by hand weight

Return Values

Real value from 0 to 5000 mm/sec/sec

See Also

AccelS Statement, Arc3, SpeedS, Jump3, Jump3CP

AccelS Function Example

```
Real savAccelS

savAccelS = AccelS(1)
```

Acos Function

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. Values range is from -1 to 1. The value returned by Acos will range from 0 to PI radians. If *number* is < -1 or > 1, an error occurs.

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

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. The additional S axis is 8 and T axis is 9.

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 command window using the Print instruction.

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

AglToPls Function

Converts robot angles to pulses.

Syntax

AglToPls(*j1*, *j2*, *j3*, *j4* [, *j5*, *j6*] [, *j7*] [, *j8*, *j9*] **)**

Parameters

j1 - j6	Real expressions representing joint angles.
j7	Real expression representing the joint #7 angle. For the Joint type 7-axis robot.
j8	Real expression representing the additional S axis angle.
j9	Real expression representing the additional T axis angle.

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)
```

AIO_In Function

Reads analog value form optional analog I/O input channel.

Syntax

AIO_In*(Channel Number)*

Parameters

Channel Number	Specify the channel number of the analog I/O.

Return Values

Return the analog input value of the analog I/O channel which specified in channel number in real number.
Return value range differs depending on the input range configuration of the analog I/O board.

Description

InFunction

See Also

AIO_InW Function, AIO_Out, AIO_OutW, AIO_Out Function, AIO_OutW Function, AIO_Set, Wait

AIO_In Function Example

```
Function main
    Real var1
    var1 = AIO_In(2)   'Acquires input state of analog channel input 2
    If var1 > 5.0 Then
        Go P1
        Go P2
        'Execute other motion command here
        '.
        '.
    Else
        Print "Error in initialization!"
        Print "Sensory Inputs not ready for cycle start"
        Print "Please check analog inputs 2."
    EndIf
Fend
```

AIO_InW Function

Reads analog value from optional analog I/O input channel.

Syntax

AIO_InW(*Channel Number*)

Parameters

Channel Number	Specify the channel number of the analog I/O.

Return Values

Returns the input states (long integers from 0 to 65535) of specified analog I/O channel.

The following table shows input voltage (current) and return value of each input channel according to input range configuration of analog I/O board.

Input Data		Input Range Configuration				
Hexadecimal	Decimal	±10.24(V)	±5.12(V)	0-5.12(V)	0-10.24(V)	0-24(mA)
0xFFFF	65535	10.23969	5.11984	5.12000	10.24000	24.00000
0x8001	32769	0.00031	0.00016	2.56008	5.12016	12.00037
0x8000	32768	0.00000	0.00000	2.56000	5.12000	12.00000
0x0000	0	-10.24000	-5.12000	0.00000	0.00000	0.00000

See Also

AIO_InFunction, AIO_Out, AIO_OutW, AIO_OutFunction, AIO_OutWFunction, AIO_Set, Wait

AIO_In Function Example

```
Long word0

word0 = AIO_InW(1)
```

AIO_Out Statement

Output analog value from the optional analog I/O output channel.

Syntax

AIO_Out Channel Number, Outputdata [, Forced]

Parameters

Channel Number	Specify the channel number of the analog I/O.
Output data	Specify the real number of Real type which indicates output voltage [V] or current value [mA] in formula or value.
Forced	Optional. Usually omitted.

Description

Output the Real value indicating specified voltage [V] or current [mA] to analog output port which specified on channel port. Set the voltage output range of analog output port or selection of voltage and current output by the switch on the board. If setting a value which out of range of analog I/O port, output the border value (maximum and minimum value) which is not out of the range.

AIO_Out command becomes an error if outputting the speed information by specified channel. Stop the speed information output and execute the AIO_Out command.

Note

Forced Flag

Specify the flag if outputting the analog I/O when operating emergency stop or opening the Safety Door by NoPause task and NoEmgAbort task (special task specified NoPause or NoEmgAbort to start when executing Xqt).

Need to be careful about the system design since analog I/O output changes when operating emergency stop or opening the Safety Door.

See Also

AIO_InFunction, AIO_OutW, AIO_OutFunction, AIO_OutWFunction, AIO_Set

AIO_Out Example

Output 7.0 [V] from the analog I/O channel 1.

```
AIO_Out 1, 7.0
```

AIO_Out Function

Returns analog value in real number which is outputting in optional analog I/O output channel.

Syntax

AIO_Out(*Channel Number*)

Parameters

Channel Number	Specify the channel number of the analog I/O.

Return Values

Returns specified analog I/O channel voltage and current output state in real number. Unit of voltage output is [V] and current output is [mA].
This function is available when outputting the speed information of the robot on specified channel.

See Also

AIO_InFunction, AIO_Out, AIO_OutW, AIO_OutWFunction, AIO_Set, Wait

AIO_Out Function Example

```
Real rdata01

rdata01 = AIO_Out(1)
```

AIO_OutW Statement

Output 16 bits analog value from optional analog I/O output channel.

Syntax

AIO_OutW *Channel Number*, *Output data* [, Forced]

Parameters

Channel Number	Specify the channel number of the analog I/O.
Output data	Specify the output data (Integer expression from 0 to 65535) in formula or value.
Forced	Optional. Usually omitted.

Description

Output to analog I/O channel specified by channel number.
For the output data, specify integer expression from 0 to 65535 in formula or value.
Output voltage (current) is as follows according to output range configuration which is set by the switch on the board.

Output Data		Output Range Configuration					
Hexadecimal	Decimal	±10(V)	±5(V)	0-5(V)	0-10(V)	4-20(mA)	0-20(mA)
0xFFFF	65535	9.99970	4.99985	5.00000	10.00000	20.00000	20.00000
0x8001	32769	0.00031	0.00015	2.50008	5.00015	12.00024	10.00031
0x8000	32768	0.00000	0.00000	2.50000	5.00000	12.00000	10.00000
0x0000	0	-10.00000	-5.00000	0.00000	0.00000	4.00000	0.00000

Note

Forced Flag

Specify the flag if outputting the analog I/O when operating emergency stop or opening the Safety Door by NoPause task, NoEmgAbort task (special task specified NoPause or NoEmgAbort to start when executing Xqt), and background task.
Need to be careful about the system design since analog I/O output changes when operating emergency stop or opening the Safety Door.

See Also

AIO_InFunction, AIO_Out, AIO_OutFunction, AIO_OutWFunction, AIO_Set, Wait

AIO_OutW Example

```
AIO_OutW 1, &H8000
```

AIO_OutW Function

Returns output analog value in Long integers from 0 to 65535 which is output on optional analog I/O channel.

Syntax

AIO_OutW(*Channel Number*)

Parameters

Channel Number	Specify the channel number of the analog I/O.

Return Values

Returns the output state of specified analog I/O channel in Long integers from 0 to 65535.

The following table shows output voltage (current) and return value of each output channel according to output range configuration of analog I/O board.

Output Data		Output Range Configuration					
Hexadecimal	Decimal	±10(V)	±5(V)	0-5(V)	0-10(V)	4-20(mA)	0-20(mA)
0xFFFF	65535	9.99970	4.99985	5.00000	10.00000	20.00000	20.00000
0x8001	32769	0.00031	0.00015	2.50008	5.00015	12.00024	10.00031
0x8000	32768	0.00000	0.00000	2.50000	5.00000	12.00000	10.00000
0x0000	0	-10.00000	-5.00000	0.00000	0.00000	4.00000	0.00000

This function is available when outputting the speed information of the robot on specified channel.

See Also

AIO_InFunction, AIO_Out, AIO_OutW, AIO_OutFunction, AIO_Set, Wait

AIO_OutW Function Example

```
Long word0

word0 = AIO_OutW(1)
```

AIO_Set Statement

Output the speed information of the robot to optional analog I/O output channel.

Syntax

(1) AIO_Set *channelNumber*, On, {RefTCPSpeed | RealTCPSpeed | RefECPSpeed | RealECPSpeed }, MaximumOutputSpeed [, MiminumOutputSpeed] [, Cnv, *Conveyor Number*]
(2) AIO_Set *Channel Number*, Off
(3) AIO_Set [*Channel Number*]

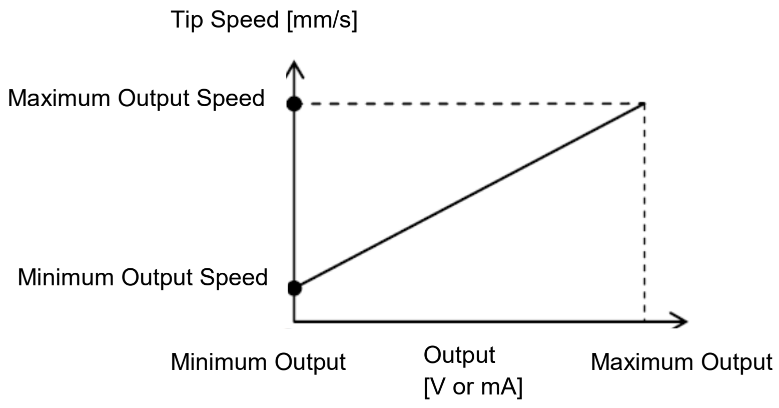
Parameters

Channel Number	Specify the channel number of the analog I/O.
On	Specify the output data (Integer expression from 0 to 65535) in formula or value.
Off	Finish analog output of the speed information and initializes to output “0”.
RefTCPSpeed	Output the commanded speed of TCP which is currently selected.
RealTCPSpeed	Output the actual speed of TCP which is currently selected.
RefECPSpeed	Output the commanded speed of ECP which is currently selected.
RealECPSpeed	Output the actual speed of ECP which is currently selected.
MaximumOutputSpeed	Specify the Real type real number (unit [mm/s]) indicating speed when outputting the maximum value of the output range in formula or value.
MinimumOutputSpeed	Specify the Real type real number (unit [mm/s]) indicating speed when outputting the minimum value of the output range in formula or value. Value is “0” [0mm/s] when omitting.
Cnv	Output the relative speed of TCP with the conveyor. Specify with the conveyor number.
Conveyor Number	Specify the conveyor number used to calculate the relative speed of TCP.

Description

Perform real-time output the speed of TCP (tool center point) or ECP (external control point) by analog voltage or current to analog I/O channel specified by channel number. Set the selection of analog voltage or current and output range configuration by a switch and jumper on the analog I/O board.

The robot speed corresponding to minimum and maximum value of the output range is determined by liner interpolation depending on specified minimum output speed and maximum output speed as shown in the figure below.



If specifying the commanded speed (RefTCPSpeed or RefECPSpeed), output the ideal speed waveform based on the applying command value on the robot.
If specifying the actual speed (RealTCPSpeed and RealECPSpeed), output the calculated speed waveform based on the actual robot move.
If specifying the TCP (RefTCPSpeed or RealTCPSpeed), output the center point speed of currently selected tool (default: Tool 0).
If specifying the ECP (RefECPSpeed or RealECPSpeed), output the speed of external control point (ECP) which is currently selected. If ECP is not selected (when ECP = 0), output the minimum output.
ECP and Cnv cannot be specified at the same time.
Only the conveyor number that has been calibrated with the manipulator can be specified.
If only channel number is specified, display the output configuration information of the specified analog channel I/O. If all argument is omitted, display the output configuration information of all analog channel I/O.

See Also

AIO_InFunction, AIO_Out, AIO_OutFunction, AIO_Out, AIO_OutWFunction, AIO_Set, Wait

AIO_Set Example

Set actual speed output of TCP of robot 1 and tool 1 to analog output channel.
Perform analog output the robot operating speed and disable the speed output configuration.

```
Robot 1
Tool 1
Motor On
Power High
SpeedS 2000
AccelS 5000
AIO_Set 1, On, RealTCPSpeed, 2000.0, 0.0
Move P1
AIO_Set 1, Off
```

AIO_Set Function

Returns the configuration information of the robot speed output which is set in optional analog I/O output channel.

Syntax

AIO_Set(*channelNumber, Index*)

Parameters

Channel number	Specify the channel number of the analog I/O.
Index	Specify the index of acquiring configuration information in integer.

Return Values

The following table shows the information that is available from the AIO_Set function:

Index	Information
1	On(1) / Off(0)
2	RefTCPSpeed(0)/ RealTCPSpeed(1)/ RefECPSpeed(2)/ RealECPSpeed(3)
3	Maximum output speed [mm/sec]
4	Minimum output speed [mm/sec]
5	Conveyor number is not specified (0)/Conveyor number (1 to 16)

See Also

AIO_InFunction, AIO_Out, AIO_OutW, AIO_OutFunction, AIO_OutWFunction, AIO_Set, Wait

AIO_Set Function Example

```
Print "Analog Ch#1 speed output is: ", AIO_Set(1, 1)
```

AIO_TrackingSet Statement

Sets the distance tracking function.

Syntax

(1) AIO_TrackingSet *channelNumber*, *Conversion coefficient of measured value and distance, Measured value at 0mm, Lower limit of available range for tracking, Upper limit of available range for tracking, [, Robot motions out of the available range for tracking [,*Axis to execute the distance tracking function*]]*

(2) AIO_TrackingSet *channelNumber*

Parameters

Channel Number

Integer expression from 1 to 8 representing the channel number of analog I/O which the distance sensor to be used is connected.

Conversion coefficient of measured value and distance

Convert the measured value (V, mA) of distance sensor to distance (mm). Specify the coefficient in read number between –500 to 500 excepting 0. (Unit: mm/V, mm/mA)

Measured value at 0mm

Specify the voltage or current value when the distance is 0mm (in case of displacement meter: amount of displacement). (Unit: V, mA)

Set the value within the input range setting of the analog I/O board.

Input range setting	Minimum value	Maximum value
±10.24 V	–10.24 V	10.24 V
±5.12 V	–5.12 V	5.12 V
0-5.12 V	0 V	5.12 V
0-10.24 V	0 V	10.24 V
0-24 mA	0 mA	24 mA

Lower limit of available range for tracking

Lower limit of the available range for tracking is the same as the lower limit of the allowable displacement amount when executing the distance tracking function. Specify the limit between –300 to 300 in real number. (Unit: mm)
Be sure to specify a larger value than the lower limit of the measurable range of the distance sensor.
For lower limit of the available range for tracking, specify a smaller value than its upper limit.

Upper limit of available range for tracking

Upper limit of the available range for tracking is the same as the upper limit of the allowable displacement amount when executing the distance tracking function. Specify the limit between –300 to 300 in real number. (Unit: mm)
Be sure to specify a smaller value than the upper limit of the measurable range of the distance sensor.
For upper limit of the available range for tracking, specify a larger value than its lower limit.

Robot motions out of the available range for tracking

When the robot is out of the available range for tracking (between the upper and lower limits as described in previous page), specify 0 or 1 to stop /continue the robot motion.
The value can be omitted. If omitted, “0” is set.
Constants are as follows:

Constant	Value	Description
AIOTRACK_ERRSTOP	0	Robot stops due to an error outside of the available range for tracking.
AIOTRACK_CONTINUE	1	Robot continues motion outside of the available range for tracking.

Axis to execute the distance tracking function

Specify an axis (integer value from 0 to 5) to execute the distance tracking function. Specify the axis which is matched with the measured direction of the distance sensor to be used.
The value can be omitted. If omitted, “2” is set.
Constants are as follows:

Constant	Value	Description
AIOTRACK_TOOL_X	0	Too coordinate X axis
AIOTRACK_TOOL_Y	1	Tool coordinate Y axis
AIOTRACK_TOOL_Z	2	Tool coordinate Z axis
AIOTRACK_ECP_X	3	ECP coordinate X axis
AIOTRACK_ECP_Y	4	ECP coordinate Y axis
AIOTRACK_ECP_Z	5	ECP coordinate Z axis

Values: 3 to 5 can be specified when the external control point (ECP) option is enabled.

Return Values

Syntax (2) shows the current set value on the console.

The following is a correspondence table of the above mentioned parameter names and parameter names displayed on the console.

Parameter names	Names displayed on the console
Conversion coefficient of measured value and distance	ScaleFactor
Measured value at 0mm	RefVoltage
Lower limit of available range for tracking	ThresholdMin
Upper limit of available range for tracking	ThresholdMax
Robot motions out of the available range for tracking	OutOfRangeMode
Axis to execute the distance tracking function	TrackingAxis

Displayed examples are as follows:
Ex 1: When channel #1 is set

```
 Ch1:
     ScaleFactor 1.000[V/mm or mA/mm]
     RefVoltage 0.000 [V or mA]
     ThresholdMin –10.000[mm]
     ThresholdMax 10.000[mm]
     OutOfRangeMode AIOTRACK_ERRSTOP
     TrackingAxis AIOTRACK_TOOL_Z
```

Ex2: When channel #1 is not set

```
     Ch1: Undefined
```

Description

AIO_TrackingSet sets parameters for the distance tracking function. Parameters to be set are determined by the distance sensor or the working environment. After booting the controller, AIO_TrackingSet must be executed before executing AIO_TrackingStart. Set parameters keep values until the robot controller is turned OFF or rebooted.

Detailed descriptions for parameters are as follows:

Conversion coefficient of measured value and distance:

When the distance sensor indicates displacement: +2mm per +1V, conversion coefficient is 2. At this time, +2mm is the displacement to direction where the distance becomes longer. Depending on the displacement meter, the voltage is set to positive to the direction where the distance becomes shorter. In this case, the conversion coefficient will be negative.

Measured value at 0mm:

For distance sensor, especially the displacement meter, voltage or current value at distance: 0mm differs depending on the products. Also, some of products can set any value for voltage or current value at distance: 0mm by user setting. Specify values depending on the using distance sensor. If the output voltage is of distance sensor is 0V when the distance (or displacement) is 0mm, this parameter is “0”.

Upper/lower limit of available range for tracking:

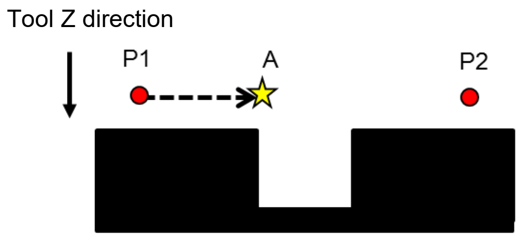
Set the upper and lower limits depending on the variations allowed by applications.
Set values must be within the measurable range of the distance sensor. The measurable range of the distance sensor differs depending on each sensor and user settings. Be sure to set the limits before executing the distance tracking function. If this parameter is set outside the measurable range of the distance sensor, the distance tracking function cannot work properly and the robot may move unintentionally.

Robot motions out of the available range for tracking:

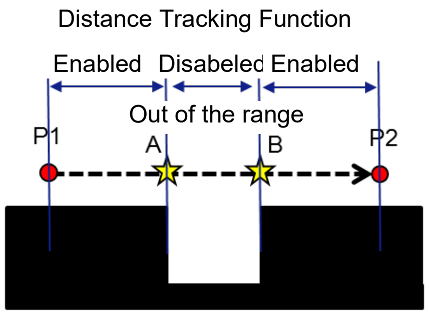
The following figures indicates the motion trajectory of the robot when the distance tracking function is executed to Z direction in Tool (when the “*Robot motions out of the available range for tracking*” parameter is set to “0” or “1”).

P1: Start point of the distance tracking function
P2: Target point

The figures indicate an object which will move outside of the measurable range at point A and return inside the range at point B.
Set the measured value (displacement) in Tool Z direction at P1 (start point of the function) as a reference value. The distance tracking function controls the robot so that the measured value always becomes the reference value. Therefore, when the robot moves from P1 to P2, the measured values between P1 and point A will be constant.
When the robot is arrived at point A, it stops due to an error if the parameter is set to “0”. If the parameter is set to “1”, the robot keeps moving to P2 from point A. However, the distance tracking function is disabled while the robot is out of the available range. When the robot moved to point B, the function is enabled since the robot is within the available range. The robot moves as with the motion from P1 to point A so that the measured value will be constant.

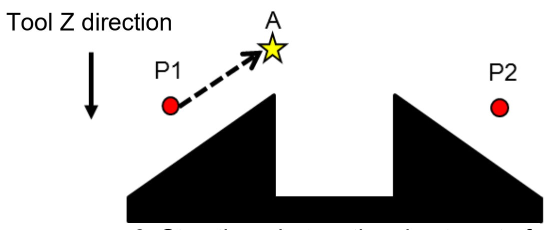


0: Stop the robot motion due to out of the range

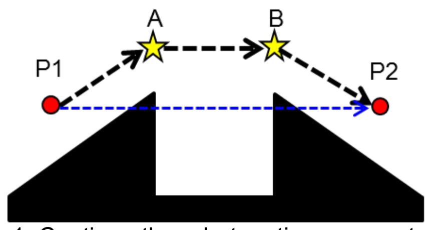


1: Continue the robot motion even out of the range

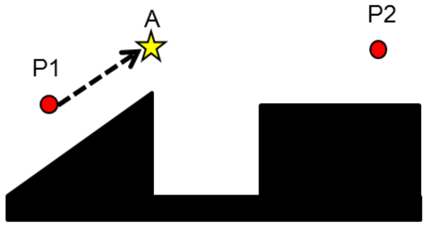
When the parameter is set to “1” and the robot moves outside of the range, the robot moves on the trajectory from the start point (P1) to the target point (P2) with CP motion. As shown in the figures below, the trajectory between A and B (outside of the available range) will become parallel to its of P1-P2. When the robot arrived to point B, the robot returned to inside the available range. Therefore, the robot is controlled based on the measured value and may move suddenly.



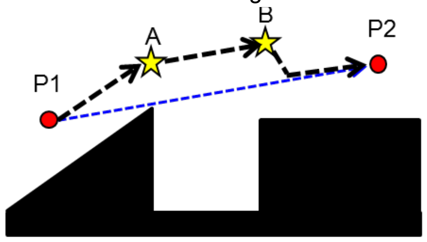
0: Stop the robot motion due to out of the range



1: Continue the robot motion even out of the range



0: Stop the robot motion due to out of the range



1: Continue the robot motion even out of the range

CAUTION

- If each parameter is not set correctly, the robot may move unintentionally when AIO_TrackingStart is executed.

  Be sure to set properly depending on the using device and working environment.

  If the robot moves abnormally, immediately hold down the emergency button.

See Also

AIO_TrackingStart, AIO_TrackingEnd, AIO_TrackingOn Function

AIO_TrackingSet Function Example

The following is an example program which moves the robot by using the distance tracking function. (P1: Start point, P2: End point)

⚠ CAUTION	■ The parameters set in the example are reference values. Please note that the operation may not be successful or the motion may be vibratory depending on the set parameters and some operating conditions. If the robot moves abnormally, immediately hold down the emergency button.

```
Function Main
      Motor On
      Power High
      SpeedS 30
      AccelS 300,300

      Go P1                             ‘ Move to P1: start point
      AIO_TrackingSet 1,1,0,-5,5,0,2    ‘ Set the distance tracking function
      AIO_TrackingStart 1,5,5,5         ‘ Start the distance tracking function
      Move P2                           ‘ Move to P2 with executing the distance tracking
                                          function

      AIO_TrackingEnd                   ‘ End the distance tracking function
      Motor Off
Fend
```

AIO_TrackingStart Statement

Starts the distance tracking function.

Syntax

AIO_TrackingStart *channelNumber*,*ProportionalGain [,IntegralGain [,DifferentialGain]]*

Parameters

Channel Number	Integer expression from 1 to 8 representing the channel number of analog I/O which the distance sensor to be used is connected.
ProportionalGain	Real value (less than 50 except 0) representing a proportional gain of the distance tracking function. Optimum value differs depending on the robot motion speed or workpiece shape. Therefore, the value needs to be set depending on the using environment.
IntegralGain	Real value (less than 100) representing an integral gain of the distance tracking function. Optional. If omitted, "0" is set. To increase accuracy of the distance tracking, adjust the integral gain.
DifferentialGain	Real value (less than 100) representing a differential gain of the distance tracking function. Optional. If omitted, "0" is set. To increase accuracy of the distance tracking, adjust the differential gain.

Description

The distance tracking function controls the robot so that a constant distance can be kept between the robot and the workpiece using the value measured by distance sensor which is connected to the analog I/O.
Direction of the robot axis to be controlled is specified by the *"Axis to execute the distance tracking function"* parameter of AIO_TrackingSet. If the kept distance is set as "reference value", the measured value by the distance sensor when executing the command will be the reference value.
Execute AIO_TrackingStart to start the distance tracking function and the function ends by executing AIO_TrackingEnd. The function is working until AIO_TrackingEnd is executed. If you do not use the function, execute AIO_TrackingEnd immediately to end the function.
If AIO_TrackingStart is executed before AIO_TrackingSet, an error occurs. Be sure to execute AIO_TrackingSet before executing AIO_TrackingStart.
The distance tracking function is available for SCARA robots (including RS series manipulators) and 6-Axis robots (including N series manipulators).
The robot can move while the function is working. However, the robot moves in CP motion only and PTP motion is not available.
If the robot passes singularity neighborhood while the distance tracking function is working, an error occurs.

The following commands cannot be used while the distance tracking function is executed.

Command to turn OFF the motor	Motor off, SFree
PTP motion commands	BGo, Go, JTran, Jump, Jump3, Jump3CP, JumpTLZ, Pass, Ptran, Pulse, TGo
Force control commands	FCKeep, Motion commands with FC, FS#.Reset, FS.Reboot
Torque control command	TC
Conveyor tracking commands	Motion command + Cnv_QueGet
VRT commands	VRT, VRT_CPMotion
Setting commands	AIO_TrackingSet, Arm, ArmSet, Base, Calib, CalPls, ECP, ECPSet, Hofs, Inertia, MCal, Power, TLSet, Tool, Weight (For AIO_TrackingSet, ArmSet, ECPSet, and TLSet, an error occurs when changing the using number.)
Others	Brake, Here, Home, VCal, WaitPos

Settings for ProportionalGain, IntegralGain, and DifferentialGain

In ProportionalGain, the larger value you set, the faster the robot tracks. However, if the set value is too large, the robot moves too fast and may result in an error.
IntegralGain and DifferentialGain can be omitted. To increase the correction accuracy, the setting is required. If the setting is not proper, the robot may move fast or vibrate.
For details on each gain setting, refer to the following manual.
EPSON RC+ User's Guide: 19. Distance Tracking Function

⚠ CAUTION	■ If too large value is set for ProportionalGain, IntegralGain, and DifferentialGain, the robot may move unintentionally. Please increase values of each parameter gradually. Changing the value to a larger one at one time is extremely hazardous and the robot may move unintentionally. If the robot moves abnormally, immediately hold down the emergency button.

See Also

AIO_TrackingSet, AIO_TrackingEnd, AIO_TrackingOn Function

AIO_TrackingStart Statement Example

The following is an example program which moves the robot by using the distance tracking function. (P1: Start point, P2: relay point, P3: End point)

⚠ CAUTION	■ The parameters set in the example are reference values. Please note that the operation may not be successful or the motion may be vibratory depending on the set parameters and some operating conditions. If the robot moves abnormally, immediately hold down the emergency button.

```
Function Main
      Motor On
      Power High
      SpeedS 30
      AccelS 300,300

      Go P1                            ' Move to P1: start point
      AIO_TrackingSet 1,1,0,-5,5,0,2   ' Set the distance tracking function
      AIO_TrackingStart 1,1,0,0        ' Start the distance tracking function
      Move P2                          ' Move to P2 with executing the distance tracking
                                         function
      Move P3                          ' Move to P3 with executing the distance tracking
                                         function
      AIO_TrackingEnd                  ' End the distance tracking function
      Motor Off
Fend
```

AIO_TrackingEnd Statement

Ends the distance tracking function.

Syntax

AIO_TrackingEnd

Description

End the distance tracking function started by AIO_TrackingStart.

See Also

AIO_TrackingSet, AIO_TrackingStart, AIO_TrackingOn Function

AIO_TrackingEnd Statement Example

The following is an example program which moves the robot by using the distance tracking function. (P1: Start point, P2: relay point, P3: End point)

⚠ CAUTION	■ The parameters set in the example are reference values. Please note that the operation may not be successful or the motion may be vibratory depending on the set parameters and some operating conditions. If the robot moves abnormally, immediately hold down the emergency button.

```
Function Main
      Integer ChNo
      Motor On
      Power High
      SpeedS 30
      AccelS 300,300
      ChNo=1

      Go P1                                ' Move to P1: start point
      AIO_TrackingSet ChNo,10,0,-3,3,0,2   ' Set the distance tracking function
      AIO_TrackingStart ChNo,1,0,0         ' Start the distance tracking function
      Move P2                              ' Move to P2 with executing the distance
                                             tracking function
      Move P3                              ' Move to P3 with executing the distance
                                             tracking function
      AIO_TrackingEnd                      ' End the distance tracking function
      Motor Off
Fend
```

AIO_TrackingOnFunction

Returns whether the specified robot is executing the distance tracking function or not.

Syntax

AIO_TrackingOn(robotNumber)

Parameters

robotNumber	An integer expression representing a robot number which you want to acquire.

Return Values

True (−1) when the distance tracking function is executed, False(0) when it stopped.

See Also

AIO_TrackingSet, AIO_TrackingStart, AIO_TrackingEnd

AIO_TrackingOn Function Example

```
Function Main
      Integer i
      i = AIO_TrackingOn(1)
      print i
Fend
```

Example on command window

```
＞print AIO_TrackingOn(1)
0
```

Align Function

Returns the point data converted to align the robot orientation (U, V, W) at the specified point in the tool coordinate system with the nearest or specified axis of the specified local coordinate system.

Syntax

(1) **Align** (*Point*[, *localNumber*[, *axisNumber*]])

Parameters

Point	The point data.
localNumber	The local coordinate system number to be a reference for the alignment of orientation. If omitted, the base coordinate system is used.
axisNumber	Specify the axis number to align the robot orientation. If omitted, the robot orientation will be aligned to the nearest coordinate axis.

Constant	Value	
COORD_X_PLUS	1:	+X axis
COORD_Y_PLUS	2:	+Y axis
COORD_Z_PLUS	3:	+Z axis
COORD_X_MINUS	4:	-X axis
COORD_Y_MINUS	5:	-Y axis
COORD_Z_MINUS	6:	-Z axis

Description

While operating the 6-axis robot (including N series), the robot orientation may have to be aligned with an axis of the specified local coordinate system without changing the tool coordinate system position (origin) defined with the point data.

Align Function converts the orientation data (U, V, W) of the specified point data and aligns with the nearest or specified axis of the specified local coordinate system.

For robots except for the 6-axis robots (including N series), it returns a specified point.

See Also

AlignECP Function, LJM Function

Align Function Example

```
Move Align(P0) ROT

P1 = Align(P0, 1)
Move P1 ROT

P2 = Align(P0, 1, 3)
Move P2 ROT
```

AlignECP Function

Returns the point data converted to align the robot orientation (U, V, W) at the specified point in the tool coordinate system with the nearest axis of the specified ECP coordinate system.

Syntax

(1) **AlignECP** (*Point*, *ECPNumber*)

Parameters

Point	The point data.
ECPNumber	The ECP coordinate system number to be a reference for the alignment of orientation.

Description

While operating the 6-axis robot (including N series), the robot orientation may have to be aligned with an axis of the specified local coordinate system without changing the tool coordinate system position (origin) defined with the point data.

AlignECP Function converts the orientation data (U,V,W) of the specified point data and aligns with the nearest axis of the specified local coordinate system.

For robots except for the 6-axis robots (including N series), it returns a specified point.

See Also

Align Function, LJM Function

AlignECP Function Example

```
Move AlignECP(P0) ROT

P1 = AlignECP(P0, 1)
Move P1 ROT
```

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

Opens file in the appending mode.

Syntax

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

Parameters

fileName	String expression that specifies valid path and file name. If specifying only a file name, the file must be in the current directory. See *ChDisk* for details.
fileNumber	Integer expression representing values from 30 to 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. If the specified file is not found, create a new file.
The specified *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#, Write, Flush, and Close.

Use the Close statement to close the file and release the file number.

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.

Notes

A network path is available.

File write buffering

File writing is buffered. The buffered data can be written with Flush statement. Also, when closing a file with Close statement, the buffered data can be written.

See Also

Close, Print #, BOpen, ROpen, UOpen, WOpen, FreeFile, Flush

AOpen Statement Example

```
Integer fileNum, i

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

Arc, Arc3 Statements

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.
These two commands are available for SCARA robots (including RS series) and 6-axis robots (including N series).

Syntax

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

Parameters

midPoint	Point expression or XY function. The middle point which the arm travels through on its way from the current point to *endPoint*.
endPoint	Point expression or XY function. The end point 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.)
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

Description

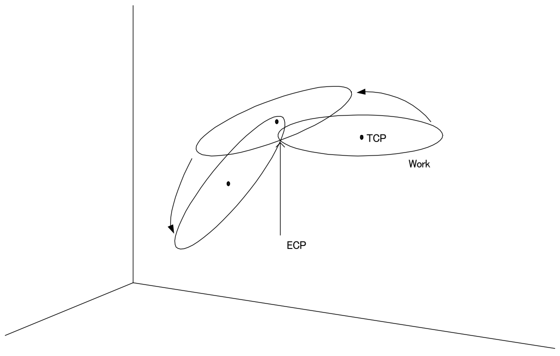
Arc and Arc3 are used to move the arm in a circular type motion from the current position to *endPoint* by way of *midPoint*. The system automatically calculates a curve based on the 3 points (current position, *endPoint*, and *midPoint*) and then moves along that curve until the point defined by *endPoint* is reached.
Also, for SCARA robots, U coordinate moves to move from current point to end point. However, for 6-Axis robot, U, V and W coordinates moves with the shortest posture for rotation to move from current point to end point.
If using this, check the actual movement in advance.

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

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 (arm) 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.

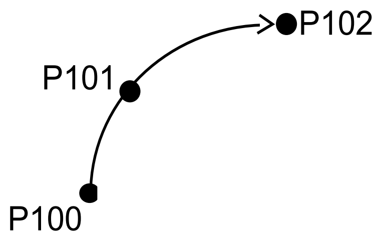
See Also

!Parallel Processing!, AccelS, Move, SpeedS

Arc, Arc3 Statements 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

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

Syntax

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

Parameters

archNumber	Integer expression representing the Arch number to define. Valid Arch numbers are from 0 to 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) For Jump3 and Jump3CP, it specifies the depart distance before a span 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) For Jump3 and Jump3CP, it specifies the approach distance before a span motion. (specified in millimeters)

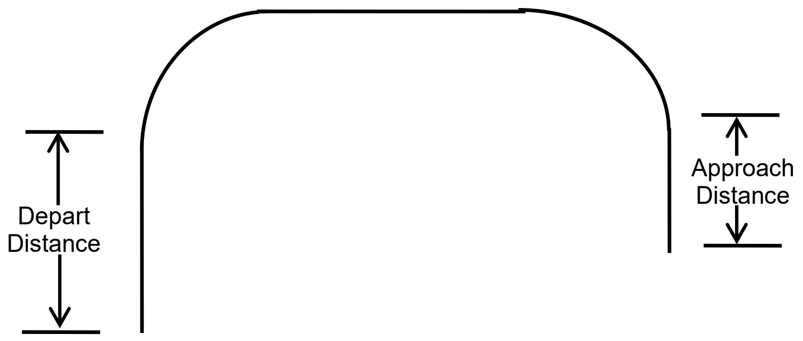
Return Values

Displays Arch Table when used without parameters.
The Arch table of the specified Arch number will be displayed when only the Arch number is specified.

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 the 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

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.)

Arch values are Maintained

The Arch Table values are permanently saved and are not changed until either the user changes them.

Caution for Arch motion

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.

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.

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.

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.

See Also

Jump, Jump3, JumpCP

Arch Statement Example

The following are examples of Arch settings done from the command 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

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 Values

Real number containing distance.

See Also

Arch Statement

Arch Function Example

```
Double 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
```

AreaCorrection Function

Returns point at which correction was made using correction area

Syntax

AreaCorrection(*Point, AreaNum*)

Parameters

Point	Specifies the point data to be corrected.
AreaNum	Specifies the area number (integer from 1 to 8) as an expression or numerical value.

Description

Points resulting from a correction are returned based on the predefined correction area. These are coordinates defined in the same local coordinate system as the points before correction. This function can be used with a motion command (such as Go or Jump command) to move the robot to a specified position. This command improves the positional accuracy of the specified point. In the point expression, enter the position on the drawing.
The correction is applied only to the position. No correction is applied for additional axes, UVW coordinate values, or orientation flags. The input point data values are output as is.
Specifying a correction area that has not been set will result in an error.

Notes

Taught points

Do not apply the AreaCorrection function to taught point data. A correction is made for the aligned teaching position, resulting in misalignment.

When far from the correction area

When the area is far from the correction area set in AreaCorrectionSet, the effect of the correction is reduced. Set reference points such that the correction area surrounds the operating point.
When plane is selected for Kind (the type of correction), the effect of the correction is reduced at points vertically distant from the plane selected for the correction area. Set the correction area at an appropriate height or, if reference points can be established, specify space for Kind (the type of correction).

When different from the orientation flag used to set the correction area

When the orientation flags and reference points set up with AreaCorrectionSet differ, an error occurs.

When different from the orientation (U, V, W) used to set the correction area

Corrections can be made for SCARA robots (including RS series).
Vertical 6-axis robots (including N series) can be corrected if the tool coordinate system's Z-axis at the point before correction matches the tool coordinate system's Z-axis at the reference point of the correction area. If there is no match, the correction is not applicable and an error occurs. The angle of the tool coordinate system's Z axis can be acquired by specifying COORD_Z_PLUS as the axis number for the DiffToolOrientation function.

See Also

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionDef Function, AreaCorrectionInv, AreaCorrectionOffset Function, DiffToolOrientation Function

AreaCorrection Statement Example

```
Function sample
    ' P(1:4) Reference point
    P1 = XY(-100, 200, -20, 0)
    P2 = XY(100, 200, -20, 0)
    P3 = XY(-100, 400, -20, 0)
    P4 = XY(100, 400, -20, 0)
    ' P(11:14) Actually use the point where P(1:4) was taught
    P11 = XY(-100, 200.5, -20, 0)
    P12 = XY(100.3, 200.1, -20, 0)
    P13 = XY(-100.4, 400.8, -20, 0)
    P14 = XY(100.2, 400.4, -20, 0)
    ' Set correction area
    AreaCorrectionSet 1, P(1:4), P(11:14), MODE_PLANE
    P999 = AreaCorrection(P1, 1)   ' P999 is a corrected point
    Print Dist(P11, P999)
    P999 = AreaCorrection(XY(0, 300, -20, 0), 1) ' Correct points in area
    Print P999
Fend
```

[Output]

```
0
 X:    0.100 Y:  300.450 Z:  -20.000 U:    0.000 /R /0
```

AreaCorrectionClr Statement

Clears correction area.

Syntax

AreaCorrectionClr *AreaNum*

Parameters

AreaNum	Specifies the area number (integer from 1 to 8) as an expression or numerical value.

Return Values

Clears the correction area corresponding to the area number.
This instruction cannot be executed while the robot is operating. Use in a stopped state.

See Also

AreaCorrectionSet, AreaCorrectionDef Function, AreaCorrectionInv, AreaCorrectionOffset Function

AreaCorrectionClr Statement Example

```
AreaCorrectionClr 1
```

AreaCorrectionDef Function

Returns status as to whether the specified correction area has been set.

Syntax

AreaCorrectionDef(*AreaNum*)

Parameters

AreaNum	Specifies the area number (integer from 1 to 8) as an expression or numerical value.

Return Values

Returns "True" if correction area has been set and "False" if not.

See Also

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionInv, AreaCorrectionOffset Function

AreaCorrectionDef Function Example

```
Function DisplayAreaCorrectionDef(areaNum As Integer)

   If AreaCorrectionDef(areaNum) = False Then
      Print "Area", areaNum, " is not defined"
   Else
      Print "Area Definition:"
      AreaCorrectionSet areaNum
   EndIf
Fend
```

AreaCorrectionInv Function

Returns corrected points to their original condition

Syntax

AreaCorrectionInv(*Point, AreaNum*)

Parameters

Point	Specifies the point data to be corrected.
AreaNum	Specifies the area number (integer from 1 to 8) as an expression or numerical value.

Description

For points that have been corrected with the AreaCorrection function, returns the point data that existed before correction.
By applying AreaCorrectionInv to a point that was actually created by teaching or to a point that has been corrected, you can obtain the point data that existed before correction.
Specifying a correction area that has not been set will result in an error.

See Also

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionDef Function, AreaCorrectionOffset Function

AreaCorrectionInv Function Example

```
Function AreaCorrectionTest
    ' P(1:4) Reference point
    P1 = XY(-100, 200, -20, 0)
    P2 = XY(100, 200, -20, 0)
    P3 = XY(-100, 400, -20, 0)
    P4 = XY(100, 400, -20, 0)
    ' P(11:14) Actually use the point where P(1:4) was taught
    P11 = XY(-100, 200.5, -20, 0)
    P12 = XY(100.3, 200.1, -20, 0)
    P13 = XY(-100.4, 400.8, -20, 0)
    P14 = XY(100.2, 400.4, -20, 0)
    ' Set correction area
    AreaCorrectionSet 1, P(1:4), P(11:14), MODE_PLANE
    P888 = AreaCorrection(P1, 1) ' P888 is a corrected point
    P999 = AreaCorrectionInv(P888, 1) ' P999 is the point prior to conversion
    Print Dist(P11, P888)
    Print Dist(P1, P999)
Fend
```

[Output]

```
0
0
```

AreaCorrectionOffset Function

Returns points relatively displaced from corrected points

Syntax

AreaCorrectionOffset(*Point, offset, AreaNum[, TOOL]*)

Parameters

Point	Specifies the point data of the position to serve as reference for relative movement.
offset	Specifies the amount of relative movement in point data.
AreaNum	Specifies the area number (integer from 1 to 8) as an expression or numerical value.
TOOL	Indicates which coordinate system to use as reference when making relative movement. If omitted, local coordinate system will be reference. When movement is based on a local coordinate system, the coordinates are relative to a coordinate system in which the point expression is defined. When movement is based on a tool coordinate system, the coordinates are relative to a point expression position.

TOOL	Constant	Value
Local coordinate reference	AC_LOCAL	0
Tool coordinate reference	AC_TOOL	1

Description

Returns a point that has moved relative to a point corrected by the AreaCorrection function. The coordinates are defined in the same local coordinate system as the specified point. This function can be used in conjunction with a motion command (such as Go or Jump command) to move the robot to a specified position.
When used in conjunction with the Here function, it can perform the same operation as BGo and TGo. The amount of relative movement is more accurate if it is within the correction area.
Specifying a correction area that has not been set will result in an error.

Notes

When a relative movement in orientation is made

If a relative movement in orientation is performed, the orientation after the relative movement will not be subject to correction and an error may occur.

See Also

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionDef Function, AreaCorrectionInv, Here

AreaCorrectionOffset Function Example

```
' Assume correction area 1 is already defined
' Similar to BGo XY(50, 0, 0, 0)
Go AreaCorrectionOffset(Here, XY(50, 0, 0, 0), 1)
' Similar to TGo XY(50, 0, 0, 0)
Go AreaCorrectionOffset(Here, XY(50, 0, 0, 0), 1, AC_TOOL)
```

AreaCorrectionSet Statement

Sets and displays the correction area.

Syntax

(1) AreaCorrectionSet *AreaNum, refPointList, taughtPointList, Kind*
(2) AreaCorrectionSet *AreaNum*
(3) AreaCorrectionSet

Parameters

AreaNum	Specifies the area number (integer from 1 to 8) as an expression or numerical value.
refPointList	Specifies sequential numbers for point data you wish to use as reference points, where two point numbers for the starting and ending points are joined by a colon, such as P(1:4). To maximize the effect of the correction, select reference points that surround the point to be corrected.
taughtPointList	Specifies taught point data corresponding to a refPointList such as P(1:4), where two point numbers for the starting and ending points are joined by a colon. The order of the point data should be set to correspond to the refPointList.
Kind	Integer value indicating the type of correction. Plane correction allows you to make corrections to points on a plane composed of the points you have selected as reference points. If plane correction is selected, place the refPointList on a plane. A minimum of three reference points is required. Space correction allows you to make corrections to points in three-dimensional space composed of the points you have selected as reference points. When selecting 3D correction, make sure that the refPointList surround the area to be corrected. A minimum of four reference points is required.

Kind	Constant	Value
Plane	MODE_PLANE	2
Space	MODE_SPACE	3

Return Values

When specified in syntax (1), the correction area is set with the specified area number.
When specified in syntax (2), the content of the specified area number is displayed.
When specified in syntax (3), the entire content of the defined correction area is displayed.

Description

Sets the correction area to be used by the area correction function. By setting the correction area and using the AreaCorrection function, AreaCorrectionInv function, and AreaCorrectionOffset function, you can improve the positioning accuracy of points within the correction area.
This instruction cannot be executed during operation. Use in a stopped state.
For information on how to select the reference position, see the following manual:
EPSON RC+ User's Guide, 22.3 Area Correction Function

Notes

About Correction Area Data

The correction area remains in effect until the controller is turned off. When the controller is started, the correction area is not defined.

About the Tool

When performing corrections using the AreaCorrection function, use the same tool that was used when teaching the reference points for the correction area. Using different tools may reduce the correction effect.

When far from the correction area

When the area is far from the correction area set in AreaCorrectionSet, the effect of the correction is reduced. Set reference points such that the correction area surrounds the operating point.
When plane is selected for Kind (the type of correction), the effect of the correction is reduced at points vertically distant from the plane selected for the correction area. Set the correction area at an appropriate height or, if a reference point can be established in the height direction, specify space for Kind (the type of correction).

When different from the orientation flag used to set the correction area

When the orientation flags and reference points set up with AreaCorrectionSet differ, an error occurs. For the reference point, set the same orientation flag as the point at which the operation is to be performed.

When different from the orientation (U, V, W) used to set the correction area

Corrections can be made for SCARA robots (including RS series).
Vertical 6-axis robots (including N series) can be corrected if the tool coordinate system's Z-axis at the point before correction matches the tool coordinate system's Z-axis at the reference point of the correction area. If there is no match, the correction is not applicable and an error occurs. The angle of the tool coordinate system's Z axis can be acquired by specifying COORD_Z_PLUS as the axis number for the DiffToolOrientation function.

See Also

AreaCorrectionClr, AreaCorrectionDef Function, AreaCorrectionInv, AreaCorrectionOffset Function, DiffToolOrientation Function

AreaCorrectionSet Statement Example

An example of use is shown below. Use the taught points for P11 to P14.
If P1 to P4 are the positions of the reference points on the drawing as shown below, the correction area will be a square 200 mm wide with P1, P2, P3, and P4 as its vertices.

```
  Function AreaCorrectionTest
    ' P(1:4) Reference point
    P1 = XY(-100, 200, -20, 0)
    P2 = XY(100, 200, -20, 0)
    P3 = XY(-100, 400, -20, 0)
    P4 = XY(100, 400, -20, 0)
    ' P(11:14) Actually use the point where P(1:4) was taught
    P11 = XY(-100, 200.5, -20, 0)
    P12 = XY(100.3, 200.1, -20, 0)
    P13 = XY(-100.4, 400.8, -20, 0)
    P14 = XY(100.2, 400.4, -20, 0)
    ' Set correction area
    AreaCorrectionSet 1, P(1:4), P(11:14), MODE_PLANE
Fend
```

Arm Statement

Selects or displays the arm number to use.

Syntax

(1) **Arm** *armNumber*
(2) **Arm**

Parameters

armNumber	Optional integer expression. Valid range is from 0 to 15. The user may select up to 16 different arms. Arm 0 is the standard (default) robot arm. Arm 1 to 15 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 (including RS series) with rotating joints 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.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

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.

Using the Arm Length Calibration Option

Applies the arm length calibration value to Arm 0 and automatically switches to Arm 0 when ArmCalib is turned On. Use Arm 0 to apply the arm length calibration value when moving the robot. The arm length calibration value will not be applied even if ArmCalib is On when using an arm number other than Arm 0. To use standard robot arm parameters, use Arm 0 with ArmCalib turned Off.

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, ArmCalibSet

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 command 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

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
```

ArmCalib Statement

Enables or disables arm length calibration for the robot currently selected.

Syntax

ArmCalib On | Off

Parameters

On | Off	Select On to enable arm length calibration. Select Off to disable arm length calibration.

Description

The ArmCalib On instruction enables arm length calibration.
Execute ArmCalib On to set the calibration value specified using ArmCalibSet to Arm 0, and switch the Arm to 0.

The ArmCalib Off instruction disables arm length calibration.
Execute ArmCalib Off to set the standard parameters to Arm 0. Note that the Arm will not switch automatically when ArmCalib Off is executed.

Arm length calibration is enabled by default when purchasing the Arm Length Calibration Option.
Even when arm length calibration is enabled, switching to an Arm other than 0 will use the settings configured for the Arm number set.

Robot parameter data is stored in the compact flash memory in the controller. Therefore, writing to the compact flash memory occurs when executing this command. Frequent writing to the compact flash memory affects the lifetime of the compact flash memory. We recommend limiting the use of this command at a minimum.

Note

This command is only available when the Arm Length Calibration Option is installed.

Do not restore backup files with ArmCalib On on a controller that has the Arm Length Calibration Option disabled.

Attempting to restore backup files with ArmCalib On on a controller that has the Arm Length Calibration Option disabled will automatically set ArmCalib Off. Keep this in mind when restoring to a separate controller.

See Also

ArmCalibClr, ArmCalibSet, ArmCalibDef

ArmCalib Statement Example

The following example is executed from the Command window.

```
> ArmCalib On

> ArmCalib Off
```

ArmCalib Function

Returns the arm length calibration status for the robot currently selected.

Syntax

ArmCalib

Return Values

0 = Arm length calibration disabled
1 = Arm length calibration enabled

See Also

ArmCalib

ArmCalib Function Example

```
If ArmCalib = Off then
   Print “Arm length calibration disabled.”
Else
   Print “Arm length calibration enabled.”
Endif
```

ArmCalibClr Statement

Clears arm length calibration settings.

Syntax

ArmCalibClr

Description

Robot parameter data is stored in the compact flash memory in the controller. Therefore, writing to the compact flash memory occurs when executing this command. Frequent writing to the compact flash memory affects the lifetime of the compact flash memory. We recommend limiting the use of this command at a minimum.

Note

Do not use ArmCalibClr unless absolutely necessary.

ArmCalibClr clears the arm length calibration parameters set using ArmCalibSet. ArmCalibSet contains factory settings that have been precisely set. (When purchasing the Arm Length Calibration Option) Accidentally clearing these settings will require precise measurements to be performed by the factory again. Do not use ArmCalibClr unless absolutely necessary.

See Also

ArmCalib, ArmCalibSet

ArmCalibClr Statement Example

```
ArmCalibClr
```

ArmCalibDef Statement

Returns the arm length calibration configuration status.

Syntax

ArmCalibDef

Return Values

Returns “True” when arm length calibration is set. Returns “False” when arm length calibration is not set.

See Also

ArmCalib, ArmCalibClr, ArmCalibSet

ArmCalibDef Statement Example

```
Function DisplayArmCalibDef
  Integer i

  If ArmCalibDef = False Then
    Print "ArmCalib is not defined"
  Else
    Print "ArmCalib Definition:"
    For i = 1 to 3
      Print ArmCalibSet(i)
    Next i
  EndIf
Fend
```

ArmCalibSet Statement

Configures and displays arm length and joint offset settings.

Syntax

(1) ArmCalibSet *setValue1*, *setValue2*, *setValue3*
(2) ArmCalibSet

Parameters

setValue	SCARA robots
1	Horizontal distance from Joint #1 to Joint #2 (mm)
2	Horizontal distance from Joint #2 to orientation center (mm)
3	Joint #2 angle offset (°)

Return Values

When the ArmCalibSet instruction is executed without all parameters, the system displays arm length calibration parameters.

Description

ArmCalibSet sets parameters related to the arm length and joint offsets. Distance accuracy can be increased by using the length and joint offset of each arm that were precisely calculated at the factory and configuring settings using this command.

Robot parameter data is stored in the compact flash memory in the controller. Therefore, writing to the compact flash memory occurs when executing this command. Frequent writing to the compact flash memory affects the lifetime of the compact flash memory. We recommend limiting the use of this command at a minimum.

Note

Do not use ArmCalibSet unless absolutely necessary.

ArmCalibSet contains factory settings that have been precisely set. (When purchasing the Arm Length Calibration Option)
Accidentally changing these settings adversely affects distance accuracy and trajectory accuracy. Do not change ArmCalibSet unless absolutely necessary.

See Also

ArmCalib, ArmCalibClr, ArmCalibDef

ArmCalibSet Statement Example

The following example is executed from the Command window.

```
> ArmCalibSet 299.989, 250.001, 0.012
```

ArmCalibSet Function

Returns an arm length calibration setting.

Syntax

ArmCalibSet(*paramNumber*)

Parameters

paramNumber	Specifies the parameter number to be referenced (integer number between 0 and 3) as an expression or a numerical value. (See below.)

SCARA robots

paramNumber	Return value
1	Horizontal distance from Joint #1 to Joint #2 (mm)
2	Horizontal distance from Joint #2 to orientation center (mm)
3	Joint #2 angle offset (°)

Return Values

Returns the parameter setting specified from one of the above as a real number.

See Also

ArmCalibClr, ArmCalibSet

ArmCalibSet Function Example

```
Double L1, L2, Angle

L1 = ArmCalibSet(1)
L2 = ArmCalibSet(2)
Angle = ArmCalibSet(3)
```

ArmClr Statement

Clears (undefines) an arm definition.

Syntax

ArmClr *armNumber*

Parameters

armNumber	Integer expression representing which of 15 arms to clear (undefine). (Arm 0 is the default arm and cannot be cleared.)

Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

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

ArmClr Statement Example

```
ArmClr 1
```

ArmDef Function

Returns arm definition status.

Syntax

ArmDef (*armNumber*)

Parameters

armNumber	Integer expression representing which arm to return status for.

Return Values

True if the specified arm has been defined, otherwise False.

See Also

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

ArmDef Function Example

```
Function DisplayArmDef(armNum As Integer)

   Integer i

   If ArmDef(armNum) = False Then
      Print "Arm ", ArmNum, "is not defined"
   Else
      Print "Arm ", armNum, " Definition:"
      For i = 1 to 5
        Print ArmSet(armNum, i)
      Next i
   EndIf
Fend
```

ArmSet Statement

Specifies and returns auxiliary arms.

Syntax

(1) **ArmSet** *armNumber*, *link2Dist*, *joint2Offset, zOffset* [*, link1Dist*] [, *orientAngOffset*]: SCARA (including RS series)
(2) **ArmSet** *armNumber*, link1Dist, link2Dist, link3Dist, link4Dist, link5Dist, link6Dist, joint1Offset, joint2Offset, joint3Offset, joint4Offset, joint5Offset, joint6Offset: 6-Axis (including N series)
(3) **ArmSet** *armNumber*
(4) **ArmSet**

Parameters

armNumber Integer expression: Valid range from 1 to 15. The user may define up to 15 different auxiliary arms.

paramNumber	SCARA Robots (including RS series)	6-Axis Robots (including N series)
1	Horizontal distance from joint #2 to orientation center (mm)	Vertical distance from base to joint #2 (mm)
2	Joint #2 angle offset (degree)	Horizontal distance from joint #1 to joint #2 (mm)
3	Height offset (mm)	Distance from joint #2 to joint #3 (mm)
4	Horizontal distance from joint #1 to joint #2 (mm)	Vertical distance from joint #3 to joint #5 (mm)
5	Joint #4 angle offset in degrees.	Horizontal distance from joint #3 to joint #5 (mm)
6	-	Distance from joint #5 to orientation center (mm)
7	-	Joint #1 angle offset in degrees.
8	-	Joint #2 angle offset in degrees.
9	-	Joint #3 angle offset in degrees.
10	-	Joint #4 angle offset in degrees.
11	-	Joint #5 angle offset in degrees.
12	-	Joint #6 angle offset in degrees.

Return Values

When the ArmSet instruction is initiated without parameters, the system displays all the auxiliary arm numbers and parameters.
The specified arm numbers and parameters will be displayed when only the arm number is specified.

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

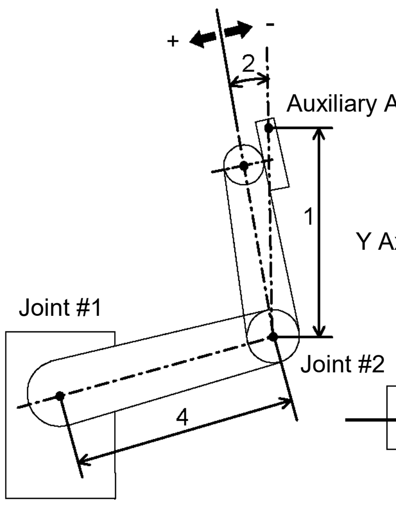
For SCARA robots (including RS series) with rotating joints used with a Cartesian coordinate system, joint angle calculations are based on the parameters defined by the ArmSet parameters. Therefore, this command is critical if any auxiliary arm or hand definition is required.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

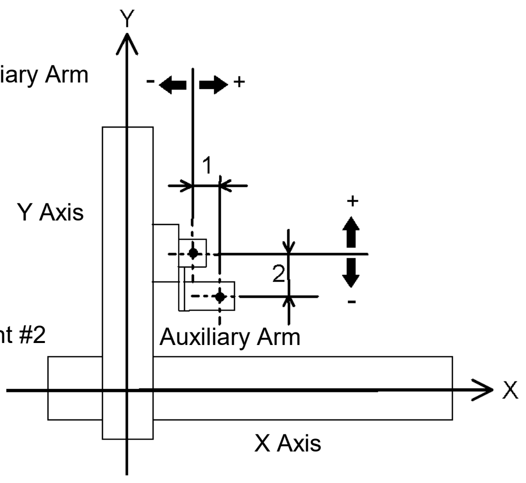
Note

Arm 0

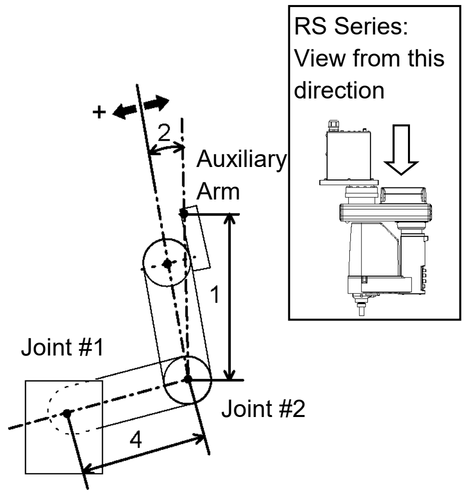
Arm 0 cannot be defined or changed by the user. It is reserved since it is used to define the standard robot configuration. When the user sets Arm to 0 this means to use the standard robot arm parameters.



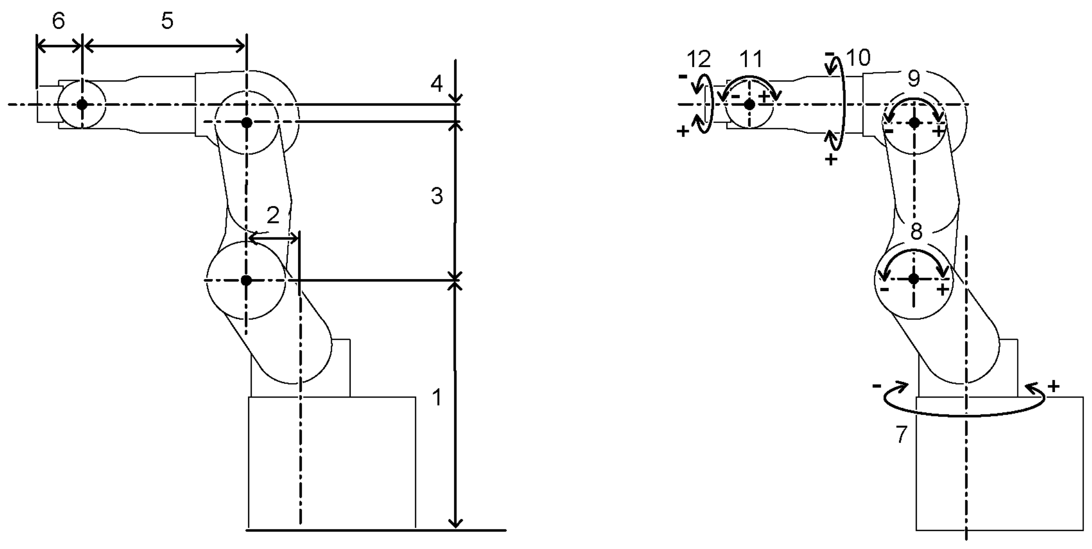
SCARA Robot



Cartesian Robot



SCARA Robots (RS Series)



6-Axis Robot

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 command window:

```
> ArmSet 1, 300, -12, -30, 300, 0
> ArmSet
  Arm 0: 125.000, 0.000, 0.000, 225.000, 0.000
  Arm 1: 300.000, -12.000, -30.000, 300.000, 0.000

> 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

Returns one ArmSet parameter.

Syntax

ArmSet(*armNumber, paramNumber*)

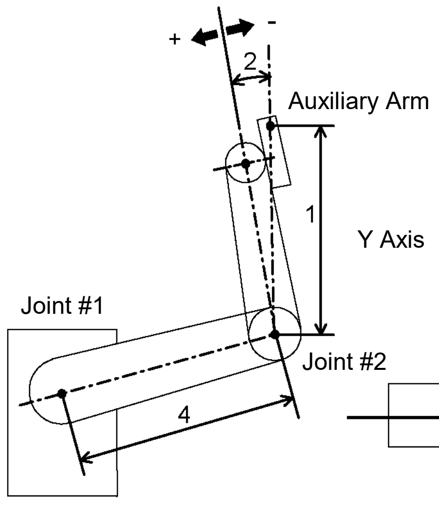
Parameters

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.

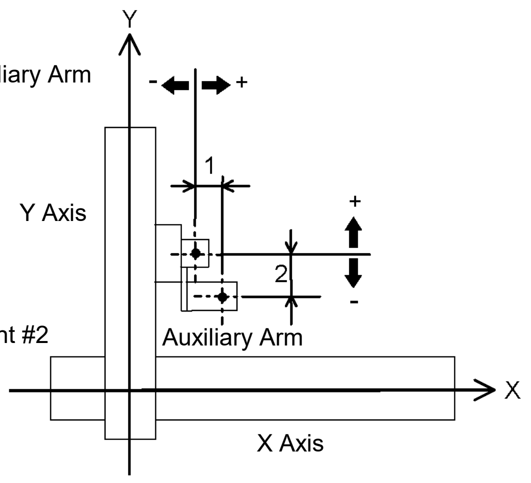
paramNumber	SCARA Robots (including RS series)	6-Axis Robots (including N series)
1	Horizontal distance from joint #2 to orientation center (mm)	Vertical distance from base to joint #2 (mm)
2	Joint #2 angle offset (degree)	Horizontal distance from joint #1 to joint #2 (mm)
3	Height offset (mm)	Distance from joint #2 to joint #3 (mm)
4	Horizontal distance from joint #1 to joint #2 (mm)	Vertical distance from joint #3 to joint #5 (mm)
5	Joint #4 angle offset in degrees.	Horizontal distance from joint #3 to joint #5 (mm)
6	-	Distance from joint #5 to orientation center (mm)
7	-	Joint #1 angle offset in degrees.
8	-	Joint #2 angle offset in degrees.
9	-	Joint #3 angle offset in degrees.
10	-	Joint #4 angle offset in degrees.
11	-	Joint #5 angle offset in degrees.
12	-	Joint #6 angle offset in degrees.

Return Values

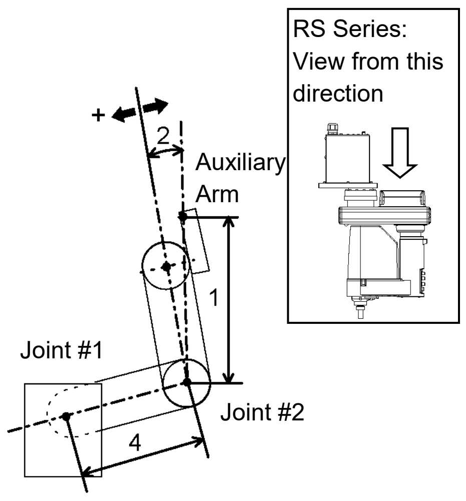
Real number containing the value of the specified parameter, as described above.



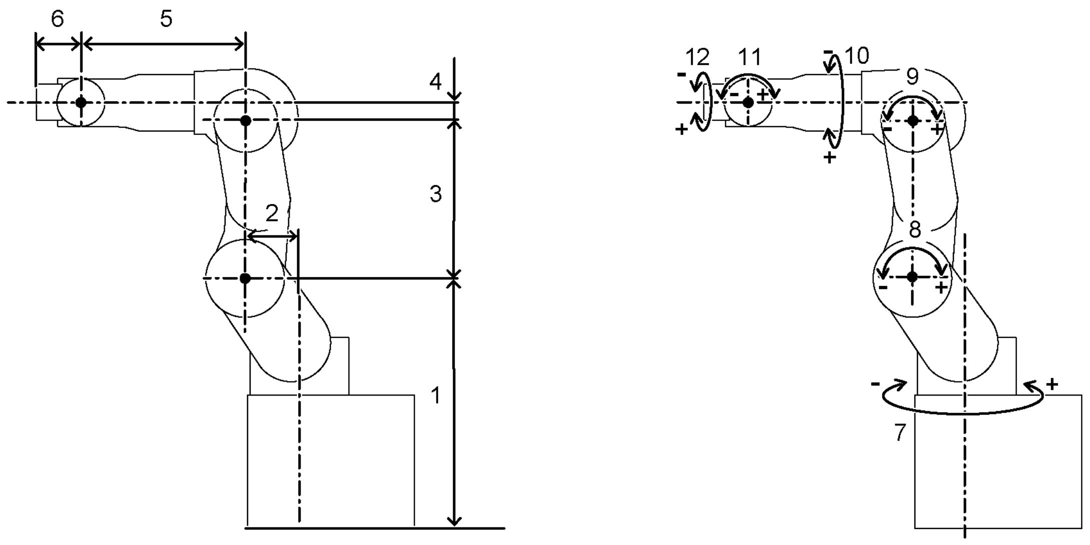
SCARA Robot



Cartesian Robot



SCARA Robots (RS Series)



6-Axis Robot

See Also

ArmClr, ArmSet Statement

ArmSet Function Example

```
Real x

x = ArmSet(1, 1)
```

Asc Function

Returns the ASCII code of the first character in a character string.
(Returns the character code in a decimal number.)

Syntax

Asc(*string*)

Parameters

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

Return Values

Returns an integer representing the ASCII code of the first 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.

Note

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.

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 command 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 command window:

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

Asin Function

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. Values range is from -1 to 1. The value returned by Asin will range from -PI / 2 to PI / 2 radians. If *number* is < -1 or > 1, an error occurs.

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
```

AtHome Function

Returns if the current robot is in its Home position or not.

Syntax

AtHome

Return Values

True if the current robot is in its Home position, otherwise False.

Description

The AtHome function returns if the current robot is in its Home position or not. To register the Home position, use HomeSet command or Robot Manager. To move to the Home position, use the Home command.

See Also

Home, HomeClr, HomeDef, HomeSet, Hordr, MCalComplete

Atan Function

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 radians.

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

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

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

Syntax

ATCLR [*j1* [,*j2* [, *j3* [, *j4* [, *j5* [, *j6* [, *j7* [, *j8* [, *j9*]]]]]]]]]

Parameters

j1 – j9	Integer expression representing the joint number. If no parameters are supplied, then the average torque values are cleared for all joints. The additional S axis is 8 and T axis is 9. If non-existent joint number is supplied, an error occurs.

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

<Example 1>

The following is the example to display the torque values of specified joints after clearing the effective torque values of all joints.

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

<Example 2>

The following is the example to display the torque values of specified joints after clearing the effective torque values of J1, J4, and J5 for the vertical multi-axis robots.

```
> atclr 4, 1, 5
> go p1
> ptrq 1
    0.227
> ptrq 4
    0.083
```

ATRQ Statement

Displays the average torque for the specified joint.

Syntax

ATRQ [*jointNumber*]

Parameters

jointNumber	Optional. Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

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 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

Returns the average torque for the specified joint.

Syntax

ATRQ (*jointNumber*)

Parameters

jointNumber	Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

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 occurs.

See Also

ATRQ Statement, PTCLR, PTRQ

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
```

AutoLJM Statement

Sets the Auto LJM function.

Syntax

AutoLJM { On | Off }

Parameters

On | Off	On: Enables the Auto LJM. Off: Disables the Auto LJM.

Description

AutoLJM is available for following commands.
Arc, Arc3, Go, Jump3, Jump3CP, Move

When AutoLJM is On, the manipulator operates with a least joint motion, just like using the LJM function, whether the LJM function is applied to the position data to be passed to each command or not. For example, to get the same effect as Go LJM(P1), you can write a program as follows.

```
AutoLJM On
    Go P1
AutoLJM Off
```

Since AutoLJM can enable LJM within a particular section of a program, it is not necessary to edit each motion command.

When AutoLJM is Off, the LJM function is only enabled when it is applied to the position data to be passed to each motion command.

In any of the following cases, AutoLJM has the setting specified in the controller settings (factory default: Off).

Controller startup
Reset
All task stop
Motor On
Switching the Auto / Programming operation mode

Notes

Double application of AutoLJM and LJM function

If LJM function is applied to the point data to be passed to the motion command while AutoLJM is On, LJM will be doubly applied at the command execution.
For Move LJM(P1, Here) and Move LJM(P1), enabling AutoLJM will not affect the motion. However, if AutoLJM is enabled for Move LJM(P1, P0), motion completion positions of Move LJM(LJM(P1, P0), Here), which enabled AutoLJM, and the one of Move LJM(P1, P0), which did not enable AutoLJM, may be different.
It is recommended to write a program not to duplicate AutoLJM and LJM functions.

AutoLJM Usage Precaution

You can set the AutoLJM function to be enabled at the controller startup by setting the controller preferences. However, if Auto LJM is enabled at all times by controller preferences or commands, this function automatically adjusts the posture of the manipulator to reduce the motion distance, even when you intended to move the joint widely. Therefore, it is recommended to create a program to apply the LJM function only when necessary by using LJM function or AutoLJM command.

See Also

AuoLJM Function, LJM Function

AutoLJM Statement Example

```
AutoLJM On
Go P1
Go P2
AutoLJM Off
```

AutoLJM Function

Returns the state of the AutoLJM.

Syntax

AutoLJM

Return Values

0	= Auto LJM OFF
1	= Auto LJM ON

See Also

AutoLJM

AutoLJM Function Example

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

AutoOrientationFlag Statement

Changes orientation flag of N6-A1000**.

Syntax

AutoOrientationFlag { On | Off }

Parameters

On | Off	On: Enables the AutoOrientationFlag. Off: Disables the AutoOrientationFlag. (Default)

Description

AutoOrientationFlag is available for following commands:
Go, BGo, TGo, Jump3, JumpTLZ

Change the following orientation flag:

Model	Parameter OFF/ON	Orientation flag			Remark
		Hand	Elbow	Wrist	
N6-A1000**	OFF	-	-	-	Move with the orientation flag which is selected by user. (Default)
	ON	-	✓	✓ *1	Set "ON" when you cannot select the orientation flag.

✓: When setting the AutoOrientationFlag to "ON", the orientation flag is changed

*1: Wrist orientation flag is changed only when you change the elbow orientation flag. When you change the wrist orientation flag, it will be the orientation flag which minimizes the movement of Joint #4.

Use AutoOrientationFlag with LJM Function

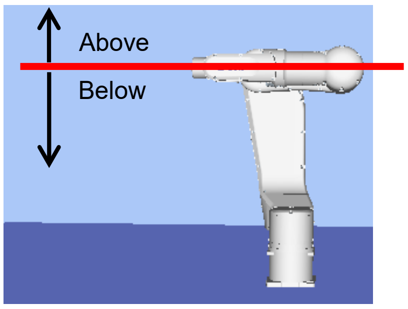
When you use the command with LJM Function, Wrist Flag, J4Flag, and J6Flag will be the orientation selected by LJM Function.
For example, when you set orientationFlag of LJM Function to "3", "Wrist Flag", "J4Flag", and "J6Flag" are selected so that Joint #5 will be the shortest movement. When you do not use LJM Function, "Wrist Flag", "J4Flag", and "J6Flag" are selected so that Joint #4 will be the shortest movement.

AutoOrientationFlag Example

```
Motor On
Power High
AutoOrientationFlag On

Go P1
Go P2
```

When setting the AutoOrientationFlag to "ON":
Flag is changed as follows due to the position of point P and the red line.

Point P is above the red line: Above
Point P is below the red line: Below



AutoOrientationFlag Function

Returns the state of the AutoOrientationFlag

Syntax

AutoOrientationFlag

Return Values

0 = AutoOrientationFlag OFF
1 = AutoOrientationFlag ON

See Also

AutoOrientationFlag

AutoOrientationFlag Function Example

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

AvgSpeedClear Statement

Clears and initializes the average of the absolute speed values for one or more joints.

Syntax

AvgSpeedClear [*j1* [,*j2* [, *j3* [, *j4* [, *j5* [, *j6* [, *j7* [, *j8* [, *j9*]]]]]]]]]

Parameters

j1 – j9	Integer expression representing the joint number. If no parameters are supplied, then the average values for all joints are cleared. The additional S axis is 8 and T axis is 9. If non-existent joint number is supplied, an error occurs.

Description

AvgSpeedClear clears the average of the absolute speed values for the specified joints.
You must execute AvgSpeedClear before executing AvgSpeed.
This command does not support the PG additional axes.

See Also

AvgSpeed, PeakSpeed

AvgSpeedClear Statement Example

<Example 1>

The following is the example to display the average speed values of specified joints after clearing the average speed values of all joints.

```
> AvgSpeedClear
> Go P1
> AvgSpeed 1
     0.073
> AvgSpeed
     0.073      0.044
     0.021      0.069
     0.001      0.108
     0.000      0.000
     0.000
>
```

<Example 2>

The following is the example to display the average speed values of specified joints after clearing the average speed values of J1, J4, and J5 for the vertical multi-axis robots.

```
> AvgSpeedClear 4, 1, 5
> Go P1
> AvgSpeed 1
     0.226
> AvgSpeed 4
     0.207
```

AvgSpeed Statement

Displays the average of the absolute speed values for the specified joints.

Syntax

AvgSpeed [*jointNumber*]

Parameters

jointNumber	Optional. Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

Return Values

Displays the average of the absolute values of current speed for the specified joints. If no joint is specified, the average of the absolute speed values for all joints will be displayed.

Description

AvgSpeed displays the average value of the absolute speed values for the specified joints. 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 the maximum average speed value.
If the average value is below 0.001, the result will be displayed as 0.

You must execute AvgSpeedClear before this command is executed.

This instruction is time restricted. You must execute AvgSpeed within 60 seconds after AvgSpeedClear is executed. When this time is exceeded, error 4088 occurs.

When using the virtual controller or conducting dry-run, the average of the absolute speed values is calculated from the commanded speed instead of the actual speed.
This command does not support the PG additional axes.

See Also

AvgSpeedClear, AvgSpeed Function, PeakSpeed

AvgSpeed Statement Example

```
> AvgSpeedClear
> Go P1
> AvgSpeed 1
      0.226
> AvgSpeed
      0.226      0.133
      0.064      0.207
      0.003      0.314
      0.000      0.000
      0.000
>
```

AvgSpeed Function

Returns the average value of the absolute speed values for the specified joints.

Syntax

AvgSpeed (*jointNumber*)

Parameters

jointNumber	Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

Return Values

Real value from 0 to 1.

Description

AvgSpeed function returns the average value of the absolute speed values for the specified joints. The loading state of the motor can be obtained by this function. The result is a real value from 0 to 1 with 1 being the maximum average speed value.

You must execute AvgSpeedClear before this command is executed.

This instruction is time restricted. You must execute AvgSpeed function within 60 seconds after AvgSpeed statement is executed. When this time is exceeded, error 4088 occurs.

When using the virtual controller or conducting dry-run, the average of the absolute speed values is calculated from the commanded speed instead of the actual speed.
This command does not support the PG additional axes.

See Also

AvgSpeed, AvgSpeedClear, PeakSpeed

AvgSpeed Function Example

This example uses the AvgSpeed function in a program:

```
Function CheckAvgSpeed
  Integer i

  Go P1
  AvgSpeedClear
  Go P2
  Print "Average speeds:"
  For i = 1 To 6
    Print "Joint ", i, " = ", AvgSpeed (i)
  Next i
Fend
```

AvoidSingularity Statement

Sets the singularity avoiding function.

Syntax

AvoidSingularity { mode }

Parameters

mode Integer expression representing a singularity avoiding mode to use

Constant	Value	Mode
SING_NONE	0	Disables the singularity avoiding function.
SING_THRU	1	Enables the singularity avoiding function.
SING_THRUROT	2	Enables the singularity avoiding function in CP motions with an ROT modifier.
SING_VSD	3	Enables variable speed CP motion function.
SING_AUTO	4	Selects the singularity avoiding function or variable speed CP motion function automatically.
SING_AVOID	5	Enables the elbow singularity avoiding function.

Description

AvoidSingularity is available for following commands.
Move, Arc, Arc3, Jump3, Jump3CP, JumpTLZ

A singularity avoiding function is to prevent acceleration errors when the vertical 6-axis (including N series) or RS series robot approaches to the singularity in CP motion by passing a different trajectory and returning to the original trajectory after passing the singularity. Since the singularity avoiding function is usually set to “1: Enabled” at the controller startup, it is not necessary to change the setting. If you do not want a singularity avoidance to ensure compatibility with software which does not support the singularity avoiding function, or to avoid a trajectory gap, disable the function.
A variable speed CP motion function automatically controls speed while keeping the trajectory when the vertical 6-axis (including N series) or RS series robot approaches to the singularity in order to avoid the acceleration error and overspeed error, and returns to the normal speed command after leaving the singularity. To pass the singularity while keeping the trajectory, Joint #1, #2, #4, and #6 may move largely.
If the AvoidSingularity parameter is changed, this function remains enabled until the next controller startup. At the controller startup, AvoidSingularity has the setting specified in the controller setting (factory default: 1). Also, parameters for SingularityAngle, SingularitySpeed, and SingularityDist are reset to the default values when AvoidSingularity setting is changed.

SING_AUTO mode is the combination of SING_THRU and SING_VSD modes. SING_THRU or SING_VSD is selected depending on the motion or speed.

Notes

Condition setting of singularity neighborhood for vertical 6-axis robot and N series robot

To determine whether the manipulator approaches to the wrist singularity neighborhood, angle of Joint #5 and angular velocity of Joint #4 are used. By default, Joint #5 angle is set to ±10 degree, and Joint #4 angle is set to ±10 % with respect to the maximum joint velocity. To change these settings, use SingularityAngle and SingularitySpeed commands.
Also, to determine whether the manipulator approaches to the hand singularity neighborhood, the coordinates of the point P is used. By default, distance between the point P and Joint #1 rotation axis is set to 30 mm. To change this setting, use SingularityDist command.

Condition setting of singularity neighborhood for RS series robot

To determine whether the manipulator approaches to the hand singularity neighborhood, the coordinates of the origin point in the default tool 0 coordinate system is used. By default, distance between the origin point and Joint #1 rotation axis is set to 30 mm. To change this setting, use SingularityDist command.

Cautions for N series robot

For N2 series, unlike other models, the default setting of singularity avoidance function is "3: Enables variable speed CP motion function."
For N6 series, like other models, the default setting of singularity avoidance function is "1: Enables the singularity avoiding function."
N series robots have the elbow singularity other than the wrist and hand singularities.
The elbow singularity area is where the Joint #3 is at 0 degree (the Joint #3 and Joint #2 overlap each other). For details of avoiding motion near the elbow singularity area, refer to the *EPSON RC+ User's Guide*.

Difference between SING_THRU and SING_AVOID

SING_THRU avoids the wrist and shoulder singularities, but not the elbow singularity.
To avoid the elbow singularity, use SING_AVOID. Note, however, that the elbow singularity avoiding motion changes the trajectory largely than the other singularity avoiding motions.
When SING_AVOID is selected for the manipulator models other than N series, an error 4002 occurs.

See Also

AvoidSingularity Function, SingualrityAngle, SingularitySpeed, SingularityDist

AvoidSingularity Statement Example

```
AvoidSingularity 0  'Disables the singularity avoidance and operate the manipulator
Move P1
Move P2
AvoidSingularity 1
```

AvoidSingularity Function

Returns the state of AvoidSingularity.

Syntax

AvoidSingularity

Return Values

0	= Singularity avoiding function disabled
1	= Singularity avoiding function enabled
2	= Singularity avoiding function enabled for CP motion commands with an ROT modifier
3	= Variable speed CP motion function enabled
4	= Automatic selection of the singularity avoiding function or the variable speed CP motion function
5	= Elbow singularity avoiding function enabled

See Also

AvoidSingularity

AvoidSingularity Function Example

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

Base Statement

Defines and displays the base coordinate system.

Syntax

(1) **Base** *pCoordinateData*
(2) **Base** *pOrigin, pXaxis, pYaxis* [, { **X** | **Y** }]

Parameters

pCoordinateData	Point data representing the coordinate data of the origin and direction.
pOrigin	Integer expression representing the origin point using robot coordinate system.
pXaxis	Integer expression representing a point along the X axis using robot coordinate system if X alignment is specified.
pYaxis	Integer expression representing a point along the Y axis using robot 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 omitted, 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)
```

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Note

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

Clears one bit in a number and returns 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 (integer from 0 to 31) to be cleared by an expression or numeric value.

Return Values

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

See Also

BClr64, BSet, BSet64, BTst, BTst64

BClr Function Example

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

BClr64 Function

Clears one bit in a number and returns the new value.

Syntax

BClr64 (*number, bitNum*)

Parameters

number	Specifies the numeric value to clear the bit by an expression or numeric value.
bitNum	Specifies the bit (integer from 0 to 63) to be cleared by an expression or numeric value.

Return Values

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

See Also

BClr, BSet, BSet64, BTst, BTst64

BClr64 Function Example

```
flags = BClr64(flags, 1)
```

BGo Statement

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

Syntax

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

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.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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 (including N series), the arm orientation attributes are automatically changed in such a way that joint travel distance is as small as possible. This is equivalent to specifying the LJM modifier parameter for Move statement. Therefore, if you want to change the arm orientation larger than 180 degrees, execute it in several times.

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

The CP parameter causes acceleration of the next motion command to start when the deceleration starts for the current motion command. In this case the robot will not stop at the destination coordinate and will continue to move to the next point.

See Also

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

BGo Statement Example

```
> BGo XY(100, 0, 0, 0) 'Move 100 mm in X direction (in the local 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, XY(0, 0, 0, 45)

  GoP1
  Print Here
  BGo XY(0, 50, 0, 0)
  Print Here

  Go P2
  Print Here
  BGo XY(0, 50, 0, 0)
  Print Here

  BGo XY(0, 50, 0, 0) /1
  Print Here

Fend
```

```
[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /R /0
 X:  300.000 Y:  350.000 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /R /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

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

Syntax

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

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.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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 (including N series), the arm orientation attributes are automatically changed in such a way that joint travel distance is as small as possible. This is equivalent to specifying the LJM modifier parameter for Move statement. Therefore, if you want to change the arm orientation larger than 180 degrees, execute it in several times.

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.

Note

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 Statement Example

```
> BMove XY(100, 0, 0, 0)'Move 100 mm 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, XY(0, 0, 0, 45)

  Go P1
  Print Here
  BMove XY(0, 50, 0, 0)
  Print Here

  Go P2
  Print Here
  BMove XY(0, 50, 0, 0)
  Print Here

  BMove XY(0, 50, 0, 0) /1
  Print Here

Fend
```

```
[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /R /0
 X:  300.000 Y:  350.000 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /R /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

Declares variables of type Boolean. (2 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 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

Boolean is used to declare variables as type Boolean. Variables of type Boolean can contain one of two values, False and 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, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UINT64, UShort

Boolean Statement Example

```
Boolean partOK
Boolean A(10)          'Single dimension array of boolean
Boolean B(10, 10)      'Two dimension array of boolean
Boolean C(5, 5, 5)     'Three dimension array of boolean

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

BOpen Statement

Opens file in binary mode.

Syntax

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

Parameters

fileName	String expression that specifies valid path and file name. If specifying only a file name, the file must be in the current directory. See ChDisk for the details.
fileNumber	Integer expression representing values from 30 to 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. If the specified file is not found, it will create a new file. If the file exists, it will read and write the data from the beginning.
Use the ReadBin and WriteBin commands to read and write data in binary mode.

Note

A network path is available.

The specified *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 ReadBin, WriteBin, Seek, Eof, Flush, and Close.

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

Use the Close statement to close the file and release the file number.

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, UOpen, WOpen, WriteBin

BOpen Statement Example

```
Integer fileNum, i

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

Flush #fileNum
Seek #fileNum, 10
ReadBin #fileNum, i
Print "data = ", i
Close #fileNum
```

Box Statement

Specifies and displays the approach check area.

Syntax

(1) **Box** *AreaNum* [, *robotNumber*], *minX*, *maxX*, *mixY*, *maxY*, *minZ*, *maxZ* [*localNumber*]
(2) **Box** *AreaNum, robotNumber*, *minX*, *maxX*, *mixY*, *maxY*, *minZ*, *maxZ, remote OutLogic* [*localNumber*]
(3) **Box** *AreaNum, robotNumber*
(4) **Box**

Parameters

AreaNum	Integer expression representing the area number from 1 to 15.
robotNumber	Integer expression that specifies which robot you want to configure. If *robotNumber* is omitted in syntax (1), the current robot number is used. You cannot omit *robotNumber* in syntax (2) and (3).
minX	The minimum X coordinate position which can be set to the approach check area.
maxX	The maximum X coordinate position which can be set to the approach check area.
minY	The minimum Y coordinate position which can be set to the approach check area.
maxY	The maximum Y coordinate position which can be set to the approach check area.
minZ	The minimum Z coordinate position which can be set to the approach check area.
maxZ	The maximum Z coordinate position which can be set to the approach check area.
Remote OutLogic	On | Off Set the Remote output logic. To set I/O output to On when the Box approaches, use On. To set I/O output to Off when the Box approaches, use Off. When the parameter is omitted, On will be used.
localNumber	Specify the local coordinate system number from 0 to 15. Be sure to add "/LOCAL" before the number. When the parameter is omitted, the local coordinate system number "0" will be used.

Return Values

When Syntax (3) is used, the area setting of the specified area is displayed.
When Syntax (4) is used, the area settings for all area numbers of the current robot are displayed.

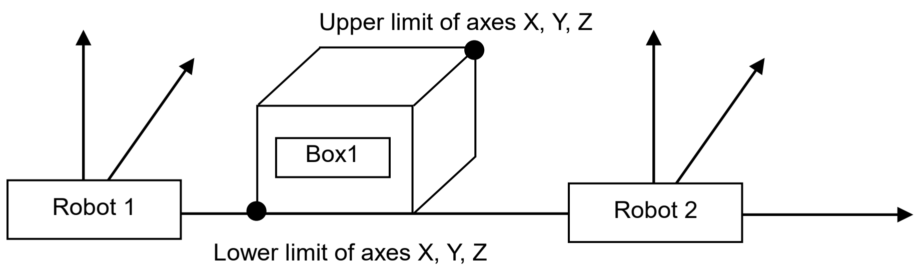
Description

Box is used to set the approach check area. The approach check area is for checking approaches of the robot end effector in the approach check area. The position of the end effector is calculated by the current tool. The approach check area is set on the base coordinate system of the robot or the local coordinate system specified by *localNumber,* and is between the specified maximum and minimum X, Y, and Z of the specified coordinate system.

When the approach check area is used, the system detects approaches in any motor power status during the controller is ON.

You can also use GetRobotInsideBox function or InsideBox function to get the result of the approach check. GetRobotInsideBox function can be used for wait condition of Wait command. You can provide the check result to the I/O by setting the remote output setting.

When several robots use one area, you should define the area from each robot coordinate system.



Configure the Box 1 from Robot 1 position

```
Box 1, 1, 100, 200, 0, 100, 0, 100
```

Lower limit of axes X, Y, Z is (100, 0, 0) and upper limit is (200, 100, 100)

Configure the Box 1 from Robot 2

```
Box 1, 2, -200, -100, 0, 100, 0, 100
```

Lower limit of axes X, Y, Z is (–200, 0, 0) and upper limit is (–100, 100, 100)

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Notes

Turning Off Approach Check Area by coordinate axis

You can turn off the approach check area of each coordinate axis. To turn off only the Z axis, define *minZ* and *maxZ* to be 0. For example Box 1, 200, 300, 0, 500, 0, 0.
In this case, it checks if the robot end effector is in the XY dimensional area.

Default values of Approach Check Area

The default values for the Box statement are “0, 0, 0, 0, 0, 0”. (Approach Check Area Checking is turned off.)

Tool Selection

The approach check is executed for the current tool. When you change the tool, the approach check may display the tool approach from inside to outside of the area or the other way although the robot is not operating.

Additional axis

For the robot which has the additional ST axis (including the running axis), the approach check plane to set doesn’t depend on the position of additional axis, but is based on the robot base coordinate system.

Tip

Set Box statement from Robot Manager

EPSON RC+ has a point and click dialog box for defining the approach check area. The simplest method to set the Box values is by using the Box page on the Robot Manager .

See Also

BoxClr, BoxDef, GetRobotInsideBox, InsideBox, Plane

Box Statement Example

<Example 1>

These are examples to set the approach check area using **Box** statement.

```
> Box 1, -200, 300, 0, 500, -100, 0

> Box
Box 1: 1, -200.000, 300.000, 0.000, 500.000, -100.000, 0.000, ON
/LOCAL0
```

<Example 2>

The following is a simple program to set the Box values by specifying the local coordinate system numbers 1 and 2.

```
Function SetBox

   Integer i

   Box 1, -200, 300, 0, 500, -100, 0 /LOCAL1

   i = 2
   Box 2, 100, 200, 0, 100, -200, 100 /LOCAL(i)

Fend
```

Box Function

Returns the specified approach check area.

Syntax

Box(*AreaNum*[, *robotNumber*], *limit*)

Parameters

AreaNum	Integer expression representing the area number.
robotNumber	Optional. Integer expression that specifies which robot you want to configure. If omitted, the current robot number is used.
limit	Integer expression that specifies which limit to return. 1: Lower limit 2: Upper limit

Return Values

When you select 1 for *limit*, the point contains the lower limit of the X, Y, Z coordinates.
When you select 2 for *limit*, the point contains the upper limit of the X, Y, Z coordinates.

See Also

Box, BoxClr, BoxDef, GetRobotInsideBox, InsideBox

Box Function Example

```
P1 = Box(1,1)
P2 = Box(1,2)
```

BoxClr Statement

Clears the definition of approach check area.

Syntax

BoxClr *AreaNum*[, *robotNumber*]

Parameters

AreaNum	Integer expression representing the area number from 1 to 15.
robotNumber	Optional. Integer expression that specifies which robot you want to configure. If omitted, the current robot number is used.

Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

Box, BoxDef, GetRobotInsideBox, InsideBox

BoxClr Statement Example

This example uses BoxClr function in a program.

```
Function ClearBox

   If BoxDef(1) = True Then
      BoxClr 1
   EndIf
Fend
```

BoxDef Function

Returns whether Box has been defined or not.

Syntax

BoxDef(*AreaNum*[, *robotNumber*])

Parameters

AreaNum	Integer expression representing an area number from 1 to 15.
robotNumber	Integer expression representing a robot number you want to configure. If omitted, the current robot will be specified.

Return Values

True if approach check area is defined for the specified area number, otherwise False.

See Also

Box, BoxClr, GetRobotInsideBox, InsideBox

BoxDef Function Example

This example uses BoxDef function in a program.

```
Function ClearBox

   If BoxDef(1) = True Then
      BoxClr 1
   EndIf
Fend
```

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 of the 6-axis robot (including N series). It's not available for SCARA Robot (include RS series).
This command is intended for use by maintenance personnel only.

When the Brake statement is executed, the robot control parameter is initialized.
See *Motor On* for the details.

⚠ 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 personnel.

Note

Before releasing the brake, be ready to use the emergency stop switch

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, SFree, SLock

Brake Statement Example

```
> brake on, 1

> brake off, 1
```

Brake Function

Returns brake status for specified joint.

Syntax

Brake (*jointNumber*)

Parameters

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

Return Values

0 = Brake off, 1 = Brake on.

See Also

Brake

Brake Function Example

```
If brake(1) = Off Then
  Print “Joint 1 brake is off”
EndIf
```

BSet Function

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 (integer from 0 to 31) 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, BClr64, BSet64, BTst, BTst64

BSet Function Example

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

BSet64 Function

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

Syntax

BSet64 (*number*, *bitNum*)

Parameters

number	Specifies the value to set the bit with an expression or numeric value.
bitNum	Specifies the bit (integer from 0 to 63) 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, BClr64, BSet, BTst, BTst64

BSet64 Function Example

```
flags = BSet64(flags, 1)
```

BTst Function

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 (integer from 0 to 31) to be tested.

Return Values

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

See Also

BClr, BClr64, BSet, BSet64, BTst64

BTst Function Example

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

BTst64 Function

Returns the status of 1 bit in a number.

Syntax

BTst64 (*number*, *bitNum*)

Parameters

number	Specifies the number for the bit test with an expression or numeric value.
bitNum	Specifies the bit (integer from 0 to 63) to be tested.

Return Values

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

See Also

BClr, BClr64, BSet, BSet64, BTst

BTst64 Function Example

```
If BTst64(flags, 1) Then
    Print "Bit 1 is set"
EndIf
```

Byte Statement

Declares variables of type Byte. (2 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 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

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, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Byte Statement 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(5, 5, 5)     '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"
    EndIf
Fend
```

Calib Statement

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

Syntax

Calib *joint1*[, *joint2*][, *joint3*][, *joint4*][, *joint5*][, *joint6*][, *joint7*][, *joint8*][, *joint9*]

Parameters

joint	Integer number from 1 to 9 that specifies the joint number to calibrate. While normally only one joint may need calibration at a time, up to all nine joints may be calibrated with the Calib command at the same time. Additional S axis is 8 and T axis is 9.

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.

Note

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.

For supported robots of joint accuracy offset, when Calib is executed, offset value set in JointAccuracy for the specified axis becomes “0”.

After executing this command, start the Safety Function Manager (only for the Controller with Safety Board)

For the Controllers with Safety Board, the Hofs value of the Controller and the Hofs value of the Safety Board that implements the safety function must match.
If this command executed with these Controllers, a warning occurs because only the Hofs value of the Controller is changed and there is a difference with the Safety Board setting.
Therefore, after executing this command, start the Safety Function Manager to refresh the Safety board settings.
For more details, refer to the following manual.
Robot Controller Safety Function Manual

Potential Error

No Joint Number Specified Error

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

See Also

CalPls, JointAccuracy, HofsJointAccuracy, Hofs

Calib Statement Example

Example from the command window.

```
> CalPls          'Display current CalPls values
   65523, 43320, -1550, 21351
> Pulse           'Display current Pulse values
PULSE: 1: 65526 pls 2: 49358 pls 3: 1542 pls 4: 21299 pls
> Calib 2         'Execute calibration for joint 2 only
> Pulse           'Display (changed) Pulse values
PULSE: 1: 65526 pls 2: 43320 pls 3: 1542 pls 4: 21299 pls
>
```

Call Statement

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. For the argument, use the following syntax: [*ByRef*] *varName* [( )], or numerical expression
ByRef	Optional. Specify ByRef when you refer to the variable to be seen by the calling function. In this case, the argument change in a function can be reflected to the variable of the calling side. You can change the values received as a reference.

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
```

You can call an external function in a dynamic link library (DLL). For details, refer to *Declare Statement*.

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

You can specify a variable as an argument. Specifying the ByRef parameter, you can reflect the change of argument in the function to the variable of the calling side.

When specifying the ByRef parameter, you need to specify ByRef as well for the argument list of the function definition (Function statement) and DLL function definition (Declare statement).

ByRef is necessary when giving an array variable as an argument.

See Also

Function, GoSub

Call Statement Example

<File1: MAIN.PRG>

```
Function main
    Call InitRobot
Fend
```

<File2: INIT.PRG>

```
Function InitRobot

    If Motor = Off Then
        Motor On
    EndIf
    Power High
    Speed 50
    Accel 75, 75
Fend
```

CalPls Statement

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

Syntax

(1) **CalPls** *j1Pulses, j2Pulses, j3Pulses, j4Pulses*[*, j5Pulses*, *j6Pulses*] [, *j7Pulses*] [, *j8Pulses*, *j9Pulses*]
(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.
j7Pulses	Optional. Seventh joint pulse value. This is a long integer expression.
j8Pulses	Optional. Eighth joint pulse value. This is a long integer expression.
j9Pulses	Optional. Nineth 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.

Note

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

Monitor window operation

```
> 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

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. The additional S axis is 8 and T axis is 9.

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

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
```

ChDir Statement

Changes and displays the current directory.

Syntax

(1) **ChDir** *pathName*
(2) **ChDir**

Parameters

pathName	String expression representing the name of the new default path. See *ChDisk* for the details.

Description

(1) Changes to the specified directory by specifying the parameter.
(2) When the parameter is omitted, the current directory is displayed. This is used to display the current directory when it is not known.

ChDir is available only with the PC disk.
When executing this command by a program, enclose the name of path with ["].

When the power is ON, the root directory will be the current directory if no project is open, and if a project is open, the project directory will be the current directory.

If you change the drive with ChDrive, the root directory will be the current directory.

The parameter is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

ChDrive, ChDisk, CurDir$

ChDir Statement Example

The following examples are done from the command window.

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

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

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

Program execution example

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

ChDisk Statement

Sets the object disk for file operations.

Syntax

ChDisk *PC|USB|RAM*

Parameters

PC	Folders (such as Hard disk) on the Windows Part
USB	USB memory on the Real Part
RAM	Memory on the Real Part

Description

Specifies which disk to use for file operations. Default is PC disk.
The Robot Controller supports the following disks as the object of file operations.

PC	Folders on the Windows Part The initial setting is PC and normally you don't have to change the setting from PC. Accesses to the files on the project folders.
USB	USB memory connected to the controller memory port This is useful to exchange files when you don't use the Windows Part (RC+). USB cannot be specified as a parameter for the T/VT series.
RAM	Temporary files on the memory These files are not saves when you turn off the controller. This is useful to save the data temporary.

Some of the SPEL$^+$ commands change the object of the file operations according to the ChDisk setting. Also, the ChDisk setting is available only with the PC disk for some commands.

ChDisk ChDrive ChDir don't affect…	Curve CVMove LoadPoints SavePoints ImportPoints file name	Object is always the project folders. File name can be specified. If path is specified, an error occurs.
ChDisk don't affect…	Access, Excel file name of OpenDB ImportPoints source path VLoadModel VSaveImage VSaveModel	Object is always the Windows folders. If only file name is specified, it can be affected by the current drive and folder. You can also specify a full path.
Executable when ChDisk is PC	ChDir FolderExists MkDir RenDir RmDir	If you execute without setting ChDisk to PC, an error occurs. If only file name and directory name are specified, it can be affected by the current drive and folder. You can also specify a full path. USB and RAM have no idea of directory.

Executable when ChDisk is USB or RAM	Copy Del FileDataTime FileExist FileLen AOpen, BOpen, ROpen, UOpen, WOpen Rename	When ChDisk is PC: If only file name and directory name are specified, it can be affected by the current drive and folder. You can also specify a full path. When ChDisk is USB or RAM: Only file name can be specified and if a path is specified, an error occurs.
Special	Declare	See *Declare* for the details. Any specified file name can be accepted. It cannot be affected by the current drive and folder

How to decide a full path when ChDisk is PC is as follows:

Only file name	"abc.txt"	Current drive + Current directory + Specified file name "C:\EpsonRC70\Projects\ProjectName\abc.txt"
Full path without a drive	"\abc.txt"	Current drive + Specified full path "C:\abc.txt"
Full path with a drive	"d:\abc.txt"	Specified full path "d:\abc.txt"
Drive is a network folder	"k:\abc.txt"	Specified full path "k:\abc.txt"
Network path	"\\Epson\data\abc.txt"	Specified full path "\\Epson\data\abc.txt"

You can have one ChDisk setting per controller.
If you want to set more than one disk as a system, take an exceptional control to switch the ChDisk setting.

See Also

ChDir, ChDrive, CurDisk$

ChDisk Statement Example

Examples from the Command window.

```
> ChDisk PC
```

ChDrive Statement

Changes the current disk drive for file operations.

Syntax

ChDrive *drive*

Parameters

drive	String expression or literal containing a valid drive letter.

Description

ChDrive is available only with the PC disk.

When the power is turned on, the "C" drive will be the current drive if a project is closed. If a project is open, the drive of the opened project will be the current drive.

See ChDisk for the details.

The parameter is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

ChDir, ChDisk, CurDrive$

ChDrive Statement Example

The following examples are done from the command window.

```
> ChDrive d
```

ChkCom Function

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

Syntax

ChkCom (*portNumber* As Integer)

Parameters

portNumber	Integer value that specifies the RS-232C port number	
	Real Part	1 to 8
	Windows Part	1001 to 1008

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 Function Example

```
Integer numChars

numChars = ChkCom(1)
```

ChkNet Function

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

Syntax

ChkNet (*portNumber* As Integer)

Parameters

portNumber	TCP/IP port number (201 to 216)

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 Function Example

```
Integer numChars

numChars = ChkNet(201)
```

Chr$ Function

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 from 1 to 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
```

ClearPoints Statement

Erases the robot position data memory.

Syntax

ClearPoints

Description

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

See Also

Plist, LoadPoints, SavePoints

ClearPoints Statement Example

The example below shows simple examples of using the ClearPoints command (from the command window). Notice that no teach points are shown when initiating the Plist command once the ClearPoints 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
>clearpoints
>plist
>
```

Close Statement

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 to 63.

Description

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

See Also

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

Close Statement 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.

```
Integer fileNumber, i, j

f ileNumber = FreeFile
WOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Close #fileNum

FileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print j
Next i
Close #fileNum
```

CloseCom Statement

Closes the RS-232C port that has been opened with OpenCom.

Syntax

CloseCom *#portNumber* **| All**

Parameters

portNumber	RS-232C port number to close. Real Part 1 to 8 Windows Part 1001 to 1008 If All is specified, the task will close all the open RS-232C ports.

See Also

ChkCom, OpenCom

CloseCom Statement Example

```
CloseCom #1
```

CloseDB Statement

Closes the database that has been opened with the OpenDB command and releases the file number.

Syntax

CloseDB *#fileNumber*

Parameters

fileNumber	Database number specified with OpenDB from 501 to 508

Description

CloseDB closes the database and Excel book, and releases the database number.

Note

- Connection of PC with installed RC+ is required.

See Also

OpenDB, SelectDB, UpdateDB, DeleteDB, Input #, Print #

CloseDB Statement Example

Refer to OpenDB use example.

CloseNet Statement

Closes the TCP/IP port previously opened with OpenNet.

Syntax

CloseNet *#portNumber* **| All**

Parameters

portNumber	TCP/IP port number to close (201 to 216) If All is specified, the task will close all the open TCP/IP ports.

See Also

ChkNet, OpenNet

CloseNet Statement Example

```
CloseNet #201
```

Cls Statement

Clears the EPSON RC+ Run, Operator, or Command window text area. Clears also the TP print panel.

Syntax

(1) **Cls #*deviceID***
(2) **Cls**

Parameters

deviceID	21 RC+ 24 TP (TP1 only) 20 TP3 When *deviceID* is omitted, the display device is cleared.

Description

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

If Cls is executed from a program that was started from the Command window, the command window text area is cleared.

When *deviceID* is omitted, the display of the current display device is cleared.

Cls Statement Example

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

```
Function main
    Integer i

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

Cnv_AbortTrack Statement

Aborts tracking motion 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 specified, the robot will move up to this value only if the Z axis position at the time of abort is below *stopZHeight* and will then be decelerated to a stop.

If *stopZHeight* is omitted, the robot is decelerated to a stop without the depart motion in the Z direction.

Note

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

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 0.01
  Loop
Fend
```

Cnv_Accel Statement

Sets acceleration and deceleration of the tracking motion in the Conveyor Tracking.

Syntax

Cnv_Accel (*conveyorNumber*) , *accel/decel*

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)
accel/decel	Acceleration and deceleration of tracking motion

Description

Sets acceleration and deceleration of the tracking motion in Conveyor Tracking.
Acceleration and deceleration cannot be set separately.
Change the parameters when acceleration setting error occurs, or when it is required to reduce work picking time. The default value is 2000[mm/sec^2].

Note

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

See Also

Cnv_Accel Function

Cnv_Accel Statement Example

```
Cnv_Accel 1,2000
```

Cnv_Accel Function

Returns acceleration and deceleration of tracking motion in Conveyor Tracking.

Syntax

Cnv_Accel (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)

Return Values

Real value in millimeters.

Note

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

See Also

Cnv_Accel

Cnv_Accel Function Example

```
Print Cnv_Accel (1)
```

Cnv_AccelLim Statement

Sets limit of acceleration and deceleration after the conveyor tracked.

Syntax

Cnv_AccelLim *conveyorNumber*, *modeNumber*, *accel/decel*

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)
modeNumber	Integer expression representing the tracking mode of the conveyor tracking (0 to 2)
accel/decel	Sets real value (unit: mm/sec2) of acceleration limit and deceleration limit after the conveyor tracked. Setting range is the same as the AccelS statement.

Initial value

Picking quantity-priority mode	500 [mm/sec^2]
Picking accuracy-priority mode	2000 [mm/sec^2]
Variable speed conveyor mode	6000 [mm/sec^2]

Description

Repeating the operation of stopping and starting the conveyor during conveyor tracking causes delay in the robot's tracking due to the conveyor’s speed changes. Set the limit of acceleration and deceleration when improve tracking ability.
The acceleration and deceleration cannot be set separately.
If the limit is too high, robot motion gets oscillatory due to variation of the conveyor speed or noise. If it is lowered too much, the robot will not stop tracking the conveyor even it stops, and it may move out of the operating area of the robot. In that case, set a tracking abort line or program to stop tracking at the downstream limit.

Notes

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

See Also

Cnv_AccelLim Function

Cnv_AccelLim Example

```
Cnv_AccelLim 1,2,7000
```

Cnv_AccelLim Function

Returns limit of acceleration and deceleration after the conveyor tracked.

Syntax

Cnv_AccelLim (*conveyorNumber*, *modeNumber*)

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)
modeNumber	Integer expression representing the tracking mode of the conveyor tracking (0 to 2)

Return Values

Real value in millimeters.

Notes

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

See Also

Cnv_AccelLim

Cnv_AccelLim Example

```
Print Cnv_AccelLim (1,2)
```

Cnv_Adjust Statement

Sets whether to acquire the follow-up delay offset value for conveyor tracking.

Syntax

Cnv_Adjust *conveyorNumber*, On | Off

Parameters

conveyorNumber	An expression or a number (1 to 16) representing the conveyor number.
On | Off	Set to "On" to acquire the follow-up delay offset value for conveyor tracking. Set to "Off" to stop acquiring.

Description

Sets whether to acquire the follow-up delay offset value for conveyor tracking.

Execute the Cnv_QueGet function with Cnv_Adjust set to "On" to acquire the offset value. When picking up a workpiece, set Cnv_Adjust to "Off" and execute the Cnv_QueGet function.
If Cnv_Adjust is set to "On", always turn it "Off" once the offset value has been acquired.

Cnv_Adjust can only be used with linear conveyors. It cannot be used with circular conveyors. For circular conveyors, the offset value will not be acquired even when turning Cnv_Adjust "On".

The offset value acquired will be cleared when power to the controller is turned off. When turning power to the controller back on, the initial value of "0" will be set. Therefore, after acquiring the offset value, use the print Cnv_AdjustGet statement to return the offset value, and set this offset value using Cnv_AdjustSet before running Cnv_QueGet in the program.

Note

This command is only available when the Conveyor Tracking option is installed.

See Also

Cnv_AdjustGet function, Cnv_AdjustSet, Cnv_AdjustClear, Cnv_QueGet function

Cnv_Adjust Statement Example

```
Cnv_Adjust 1, On
Jump Cnv_QueGet(1)
 .
 .
Cnv_Adjust 1, Off
```

Cnv_AdjustClear Statement

Clears the follow-up delay offset value for conveyor tracking.

Syntax

Cnv_AdjustClear *conveyorNumber*

Parameters

conveyorNumber	An expression or a number (1 to 16) representing the conveyor number.

Description

Clears the results of actions to acquire the follow-up delay offset value for conveyor tracking, the offset amount, and offset time, and resets values to “0”.

Note

This command is only available when the Conveyor Tracking option is installed.

See Also

Cnv_Adjust, Cnv_AdjustSet function, Cnv_AdjustGet, Cnv_QueGet function

Cnv_AdjustClear Statement Example

```
Cnv_AdjustClear 1
```

Cnv_AdjustGet Function

Returns the follow-up delay offset value for conveyor tracking.

Syntax

Cnv_AdjustGet(*conveyorNumber*, *modeNumber*)

Parameters

conveyorNumber	An expression or a number (1 to 16) representing the conveyor number.
modeNumber	0: Offset acquisition result 1: Offset amount 2: Offset time

Return Values

*modeNumber*0: Returns a real number between 0 and 2.
0: Offset value acquisition action not performed
1: Offset value acquired
2: Failed to acquire offset value
modeNumber 1: Returns a real number (unit: mm).
modeNumber 2: Returns a real number (unit: seconds).

Description

If the follow-up delay offset value for conveyor tracking has not been acquired using the Cnv_Adjust statement and the Cnv_QueGet function, the return value for each mode number (0 to 2) will be "0".

Note

This command is only available when the Conveyor Tracking option is installed.

See Also

Cnv_Adjust, Cnv_AdjustSet, Cnv_AdjustClear, Cnv_QueGet function

Cnv_AdjustGet Function Example

```
Print Cnv_AdjustGet(1, 1)
```

Cnv_AdjustSet Statement

Sets the follow-up delay offset value for conveyor tracking.

Syntax

Cnv_AdjustSet *conveyorNumber*, *offsetAmount*, *offsetTime*

Parameters

conveyorNumber	An integer value (1 to 16) representing the conveyor number.
offsetAmount	A real number representing the offset amount (unit: mm).
offsetTime	A real number representing the offset time (unit: seconds).

Description

If Cnv_AdjustSet is not performed, the previously set offset amount and offset time will be applied.
If the offset value has not been acquired since turning the controller on, the initial value of "0" will be set for the offset amount and the offset time.
Only when the mode number of Cnv_Mode command is 1(Picking accuracy-priority mode), the setting of the Cnv_AdjustSet command is applied.
Cnv_AdjustSet statement cannot be used with circular conveyors.

Note

This command is only available when the Conveyor Tracking option is installed.

See Also

Cnv_Adjust, Cnv_AdjustGet, Cnv_AdjustClear, Cnv_QueGet function

Cnv_AdjustSet Statement Example

```
Cnv_AdjustSet 1, 4.5, 0.1
```

Cnv_Downstream Statement

Sets the downstream limit of the specified conveyor.

Syntax

Cnv_Downstream (*conveyorNumber*) , *lowerLimit*

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)
lowerLimit	A border on the downstream side of the tracking area

Return Values

By using Cnv_Downstream, you can change the downstream limit which was set in the calibration wizard. However, if skewed downstream limit is used, you cannot change the value by Cnv_Downstream.

Note

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

See Also

Cnv_Upstream

Cnv_Downstream Statement Example

```
Cnv_Downstream 1,500
```

Cnv_Downstream Function

Returns the downstream limit for the specified conveyor.

Syntax

Cnv_Downstream (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)

Return Values

Linear conveyors: Real value in millimeters.
Circular conveyors: Real value in degrees.

Note

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

See Also

Cnv_Upstream

Cnv_Downstream Function Example

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

Cnv_Fine Statement

Sets the value of Cnv_Fine for one conveyor.

Syntax

Cnv_Fine *conveyorNumber* [, *fineValue*]

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)
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 omitted, the current Cnv_Fine setting is displayed.

Description

After confirming the tracking operation is complete, specify the distance from the part that is acceptable for the next command. When specifying “0”, the Cnv_Fine setting will not be used and the next command will be accepted when the motion command is complete.

The default value of “0” mm is automatically set when the following conditions occur:
Conveyor is created.
Controller is started.

Note

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

See Also

Cnv_Fine Function

Cnv_Fine Statement Example

```
Cnv_Fine 1, 5
```

Cnv_Fine Function

Returns the current Cnv_Fine setting.

Syntax

Cnv_Fine (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16).

Return Values

Real value of Cnv_Fine in millimeters.

Note

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

See Also

Cnv_Fine Statement

Cnv_Fine Function Example

```
Real f

f = Cnv_Fine(1)
```

Cnv_Flag Function

Returns the tracking state for the tracking abort line.

Syntax

Cnv_Flag (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16).

Return Values

0: Tracking is executed properly. (Tracking is not canceled or aborted.)
1: Tracking has been canceled because work piece is expected to cross the tracking abort line.
2: Tracking has been aborted because work piece is crossed the tracking abort line.
Z position is not dropped to the specified height.
3: Tracking has been aborted because work piece is crossed the tracking abort line.
Z position is dropped to the specified height.
4: Tracking has been canceled because work pieces are out of tracking area.

The return values other than "0" are returned only when the tracking abort line is defined.
For details on the tracking abort line, refer to the *User's Guide*.

Note

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

Cnv_Flag Function Example

```
Print Cnv_Flag (1)
```

Cnv_LPulse Function

Returns the pulse value latched by the conveyor trigger.

Syntax

Cnv_LPulse (*conveyorNumber*)

Parameters

ConveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Description

Returns the latest conveyor pulses latched by the hardware trigger wires or Cnv_Trigger.

Return Values

Long value that contains the latched pulses of the specified conveyor.

Note

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

See Also

Cnv_Trigger, Cnv_Pulse

Cnv_LPulse Function Example

```
Print "Latched conveyor position: ", Cnv_LPulse(1)
```

Cnv_Mode Statement

Sets a tracking mode of Conveyor Tracking.

Syntax

Cnv_Mode (*conveyorNumber*, *modeNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
modeNumber	0: Picking quantity-priority mode 1: Picking accuracy-priority mode 2: Variable speed conveyor mode

Description

Sets a tracking mode of Conveyor Tracking.
Cnv_Mode is only available for linear conveyors.
Sets the tracking mode before starting the tracking motion. If the parameters are not set, the picking quantity priority mode will be set.

Picking quantity-priority mode:	Although this mode is inferior in picking accuracy to the picking Accuracy-priority mode, it takes less time to catch up with the moving work pieces. Therefore, this mode is suitable for the conveyor systems in which space between the work pieces is narrow or the fast-speed conveyor systems.
Picking accuracy-priority mode:	Although this mode takes longer time to catch up with the work pieces compared to the picking quantity-priority mode, this improves the picking accuracy. Therefore, this mode is suitable for the conveyor systems for small work pieces.
Variable speed conveyor mode:	This mode can be used for conveyor system which stops and moves a conveyor while contacting with a workpiece.

The mode “0” is only supported by the circular conveyors. When “1” or “2” are specified, the manipulator moves as same as the mode “0”.

Note

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

See Also

Cnv_Mode Function

Cnv_Mode Statement Example

```
Cnv_Mode 1, 1
```

Cnv_Mode Function

Returns a tracking mode of Conveyor Tracking.

Syntax

Cnv_Mode (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

Returns a real value from 0 to 2.
0: Picking quantity-priority mode
1: Picking accuracy-priority mode
2: Variable speed conveyor mode

Note

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

See Also

Cnv_Mode Statement

Cnv_Mode Function Example

```
Print Cnv_Mode (1)
```

Cnv_Name$ Function

Returns the name of the specified conveyor.

Syntax

Cnv_Name$ (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

A string containing the conveyor name.

Note

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

See Also

Cnv_Number

Cnv_Name$ Function Example

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

Cnv_Number Function

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 active.

See Also

Cnv_Name$

Cnv_Number Function Example

```
Integer cnvNum

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

Cnv_OffsetAngle Statement

Sets the offset value for the conveyor queue data.

Syntax

Cnv_OffsetAngle *conveyorNumber* [, *offsetAngle*]

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
offsetAngle	Real value representing the offset value for the conveyor queue data (unit: degree). Optional. If omitted, the current offset is displayed.

Description

Sets the offset value for the conveyor queue data.
Cnv_OffsetAngle is available for the circular conveyor.
Conveyor Tracking may have tracking delay according to the conveyor speed. If the tracking delay is occurred, the robot handles the parts in the wrong position moved by the tracking delay.
Cnv_OffsetAngle gives the offset value to the queue in order to move the robot back to the correct position.

Note

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

See Also

Cnv_OffsetAngle Function

Cnv_OffsetAngle Statement Example

```
Cnv_OffsetAngle 1, 5
```

Cnv_OffsetAngle Function

Returns the offset value of the conveyor queue data.

Syntax

Cnv_OffsetAngle (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

Integer value (unit: degree).

Note

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

See Also

Cnv_OffsetAngle Statement

Cnv_OffsetAngle Function Example

```
Real offsetAngle

offsetAngle = Cnv_OffsetAngle (1)
```

Cnv_Point Function

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 that specifies the conveyor number (1 to 16)
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 active.

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

Returns deviation in current tracking position compared to tracking target.

Syntax

Cnv_PosErr (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

Real value in millimeters.

Note

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

See Also

Cnv_MakePoint

Cnv_PosErr Function Example

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

Cnv_PosErrOffset Statement

Sets value to correct the position deviation between the current tracking position and target.

Syntax

Cnv_PosErrOffset conveyorNumber, offsetValue

Parameters

conveyorNumber	An integer value (1 to 16) representing the conveyor number.
offsetValue	A real number (0 to 255; unit: msec) representing the time at which to predict the conveyor speed.

Description

Repeating the operation of stopping and starting the conveyor during conveyor tracking worsens the position deviation between the tracking position and target due to the delay in the robot's tracking of the conveyor's speed changes.
Conveyor tracking can improve the position deviation by predicting the conveyor speed after a time set by the offset value.
Cnv_PosErrOffset is available only in variable speed conveyor mode. In picking quantity-priority mode and picking accuracy-priority mode, the position deviation cannot be improved by setting an offset value.

Notes

This command is only available when the Conveyor Tracking option is installed.

See Also

Cnv_Mode, Cnv_PosErr

Cnv_PosErrOffsetStatement Example

```
Cnv_Mode 1, 2             ' Variable speed conveyor mode
Cnv_PosErrOffset 1, 10    ' Offset value 10 msec
```

Cnv_Pulse Function

Returns the current position of a conveyor in pulses.

Syntax

Cnv_Pulse (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

Long value of current pulses for specified conveyor.

Note

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

See Also

Cnv_Trigger, Cnv_LPulse

Cnv_Pulse Function Example

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

Cnv_QueAdd Statement

Adds a robot point to a conveyor queue.

Syntax

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

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
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 1000.

Note

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

See Also

Cnv_RobotConveyor

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

Returns a point from the specified conveyor's queue.

Syntax

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

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
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 active.

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

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

Syntax

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

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
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 pickup area.

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

Note

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

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

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

Syntax

Cnv_QueList *conveyorNumber*[, *numOfItems*]

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
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 active.

See Also

Cnv_QueGet

Cnv_QueList Statement Example

```
Cnv_QueList 1
```

Cnv_QueMove Statement

Moves data from upstream conveyor queue to downstream conveyor queue.

Syntax

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

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
index	Optional. Integer expression that specifies the index of the queue to move. (The first item in the queue is index #0.)
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, the first item (*index #0)* of the queue is moved.

Note

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

See Also

Cnv_QueGet

Cnv_QueMove Statement Example

```
Cnv_QueMove 1
```

Cnv_QueReject Statement

Sets and displays the queue reject distance for a conveyor.

Syntax

Cnv_QueReject *conveyorNumber* [, *rejectDistance*]

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
rejectDistance	Optional. Real expression specifying the minimum distance between parts allowed in the queue in millimeters. If a negative value is specified, 0 mm will be set. If omitted, the current *rejectDistance* is displayed.

Description

Use Cnv_QueReject to specify the minimum 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 active.

See Also

Cnv_QueReject Function

Cnv_QueReject Statement Example

```
Cnv_QueReject 1, 20
```

Cnv_QueReject Function

Returns the current part reject distance for a conveyor.

Syntax

Cnv_QueReject (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

Real value in millimeters.

Note

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

See Also

Cnv_QueReject Statement

Cnv_QueReject Function Example

```
Real rejectDist

RejectDist = Cnv_QueReject(1)
```

Cnv_QueRemove Statement

Removes items from a conveyor queue.

Syntax

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

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
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 active.

See Also

Cnv_QueAdd

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

Sets and displays user data associated with a queue entry.

Syntax

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

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
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 active.

See Also

Cnv_QueUserData Function

Cnv_QueUserData Statement Example

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

Cnv_QueUserData Function

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

Syntax

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

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)
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 active.

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

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 active.

See Also

Cnv_MakePoint Statement

Cnv_RobotConveyor Function Example

```
Integer cnvNum

cnvNum = Cnv_RobotConveyor(1)
```

Cnv_Speed Function

Returns the current speed of a conveyor.

Syntax

Cnv_Speed (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

For straight conveyors, a real value in millimeters per second. For circular conveyors, a real value in degrees per sec.

Note

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

See Also

Cnv_Pulse

Cnv_Speed Statement Example

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

Cnv_Trigger Statement

Latches current conveyor position for the next Cnv_QueAdd statement.

Syntax

Cnv_Trigger *conveyorNumber*

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

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 active.

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 Statement

Sets the upperstream limit of the specified conveyor.

Syntax

Cnv_Upstream (*conveyorNumber*), *upperLimit*

Parameters

conveyorNumber	Integer expression representing the conveyor number (1 to 16)
upperLimit	A border on the upperstream side of the tracking area

Return Values

By using Cnv_Upstream, you can change the upperstream limit which was set in the calibration wizard. However, if skewed upperstream limit is used, you cannot change the value by Cnv_Upstream.

Note

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

See Also

Cnv_Downstream

Cnv_Upstream Statement Example

```
    Cnv_Upstream 1,200
```

Cnv_Upstream Function

Returns the upstream limit for the specified conveyor.

Syntax

Cnv_Upstream (*conveyorNumber*)

Parameters

conveyorNumber	Integer expression that specifies the conveyor number (1 to 16)

Return Values

Linear conveyors: Real value in millimeters.
Circular conveyors: Real value in degrees.

Note

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

See Also

Cnv_Downstream

Cnv_Upstream Function Example

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

CollisionDetect Statement

Enables or disables the collision detection (detection of robot motion error) of the current robot.

Syntax

(1) CollisionDetect *status*
(2) CollisionDetect *status*, *jointNumber*
(3) CollisionDetect

Parameters

status	On: Enables the collision detection (detection of robot motion error).	
	Off: Disables the collision detection (detection of robot motion error).	
jointNumber	SCARA robots (including RS series):	Specify the joint by a joint number from 1 to 4
	Vertical 6-axis robots (including N series):	Specify the joint by a joint number from 1 to 6

Result

Returns the current CollisionDetect status when the parameters are omitted.

Description

Detect the robot motion error from differentiation between desired speed and the actual speed (speed deviation value). Errors can be detected by this function is classified into A and B.

A: Collision or contact of robot arm or hand occurs

B: Robot motion errors other than collision or contact

Also, error B is classified into below according to the power condition.

Error in high power

Torque saturation due to little setting of Weight or Inertia.

Torque saturation due to combined motion of multiple joints and throwing around the long object.

Torque saturation due to supply voltage reduction.

Error motion due to hardware error or software malfunction.

Error in low power

Error motion due to hardware error or software malfunction.

Torque saturation in low power due to holding a hand or long object that exceeds the weight described in the specifications.

The collision detection is available for the general-purpose robots supported by the EPSON RC+ 7.0 Ver.7.2 or later (vertical 6-axis and SCARA robots). If this command is used while unsupported robot (X5 series, etc.) is connected, an error occurs.
Execution of this command takes a little time. If cycle time is prioritized, minimize the use of this command in the program.
This function can be enabled or disabled for each joint or all joints. The default is “all joints on”.
(The default is off if the firmware version is before Ver 7.2.0.x.)
The setting returns to the default when the Controller is turned off. In other cases, the setting does not change unless otherwise configured by this command explicitly.
Output the following messages and stop the robot when the collision is detected.

Error 5057 “Collision was detected in High power mode” (detection of robot motion error).

Error 5058 “Collision was detected in Low power mode” (detection of robot motion error).

For reducing damage in High power mode, using the command together with the upper limit torque restriction by LimitTorque is also effective. For reducing damage in Low power mode, using the command together with the upper limit torque restriction by LimitTorqueLP is also effective.

Also refer to EPSON RC+ 7.0 User's Guide "*6.18.10 Collision Detection Function (detection of robot motion error)*".

See Also

LimitTorque, LimitTorque Function, LimitTorqueLP, LimitTorqueLP Function

CollisionDetectStatement Example

```
CollisionDetect On          ' Turns On the collision detection for all joints
CollisionDetect Off, 5      ' Turns On the collision detection for only Joint #5
CollisionDetect             ' The result will be displayed as "on, on, on, on, off, on".
```

CollisionDetect Function

Returns the setting value of CollisionDetect command.

Syntax

CollisionDetect(*jointNumber*)

Parameters

jointNumber	Specify the joint by a joint number from 1 to 6.

Return Values

Returns the setting value of CollsionDetect command by an integer.

1	= ON
0	= OFF

See Also

CollisionDetect

CollisionDetect Function Example

```
Print CollisionDetect(1)    'Displays CollisionDetect value of the Joint #1.
```

Cont Statement

Resumes the controller after a Pause statement has been executed and continues the execution of all tasks.
This command is for the experienced user and you need to understand the command specification
before the use.

Syntax

Cont

Description

To execute the Cont statement from a program, you need to set the [Enable advanced task commands] checkbox in Setup | System Configuration | Controller | Preferences page of the EPSON RC+. However, even if this preference is enabled, you cannot execute the Cont statement from a task executed by Trap SGClose.

The Cont command resumes the controller tasks paused by the Pause statement or safeguard open and continues all tasks execution. It has the same function as the <Continue> button on the Run Window, Operator Window, and the Continue Remote input.

If you execute the Cont command during WaitRecover status (waiting for the recover after safeguard open), it will turn on all the robot motors and execute the recover motion. Then, the program will be resumed.
If you just want to turn on motors and execute recover motion, use the Recover command.

⚠ CAUTION	■ When executing Cont command from a program, you must understand the command specification and confirm that the system has the proper conditions for the Cont command. Improper use such as continuous execution of a command within a loop may deteriorate the system safety.

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 and Sw(cswitch) = On
            Cont
        EndIf
    Loop
Fend
```

Copy Statement

Copies a file to another location.

Syntax

Copy *source*, *destination*

Parameters

source	Pathname and filename of the source location of the file to copy. See ChDisk for the details.
destination	Pathname and filename of the destination to copy the specified source file to. See ChDisk for the details.

Description

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

The same pathname and filename may not be specified for both source and destination files.
An error occurs if the destination already exists.

Note

A network path is available.

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

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

See Also

ChDir, MkDir

Copy Command Example

The following example is done from the Command window.

```
> copy TEST.DAT TEST2.DAT


> Copy TEST.DAT c:          'NG
!! Error: 7203 Access is denied.
> Copy TEST.DAT c:\         'OK
>
```

Cos Function

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 Command 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

Sets CP (Continuous Path) motion mode.

Syntax

CP { **On | Off** }

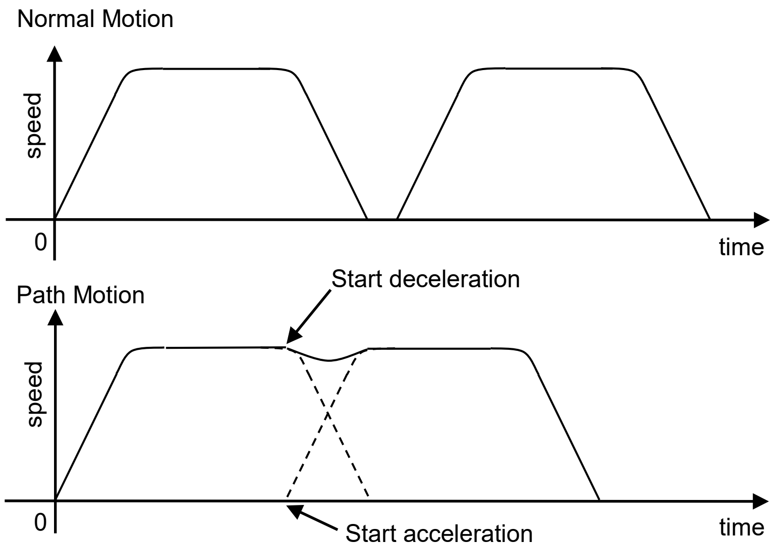
Parameters

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

Description

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

When CP mode is On, each motion command executes the next statement as deceleration starts.
Continuous path motion will continue regardless of whether the CP parameter is specified in each motion command or not.
When CP is Off, this function is active only when the CP parameter is specified in each motion command.



When CP is On, path motion will continue without full deceleration between two CP motions (Arc, Arc3, Jump3, Jump3CP, JumpTLZ, and Move), or two PTP motions (Go, Jump).
In contrast, full deceleration will occur between a CP motion and a PTP motion.
In addition, in the CP motion which target is wrist singular point in a vertical 6-axis robot (including the N series), the next motion and the motion trajectory will not be combined and full deceleration will occur.

CP will be set to Off in the following cases

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

See Also

CP Function, Arc, Arc3, Go, Jump, Jump3, Jump3CP, JumpTLZ, Move

CP Statement Example

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

CP Function

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
```

CP_Offset Statement

Sets the offset time to start the subsequent motion command when executing CP On.

Syntax

(1) CP_Offset [On [, OffsetTime]]
(2) CP_Offset Off

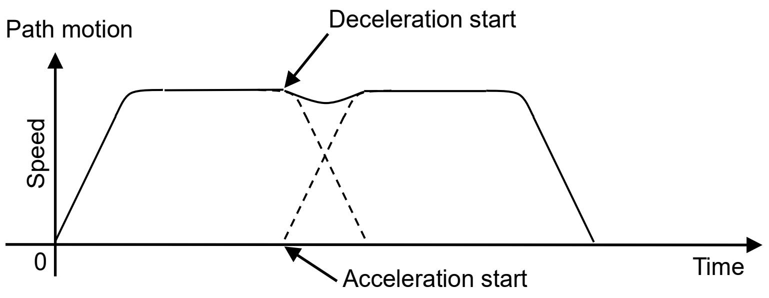
Parameters

On | Off	On: Enables the motion command start offset function in CP On. If omitted, current setting will be displayed. Off: Disables the motion command start offset function in CP On.
OffsetTime	Specify the offset time to start the subsequent command in CP On by a real value from 10 to 24 (unit: ms). If omitted, the default value (10 ms) will be set.

Description

CP_Offset is available for following commands.
Move, Arc, Arc3, CVMove

If the CP parameter is added to CP On or motion commands, the subsequent command will be executed at the same time as the prior motion starts decelerating.
As a result, the motions become a path motion as shown below, where deceleration of the first command and acceleration of the subsequent command overlap.
At this moment, the start of deceleration for the first command and the start of acceleration for the subsequent command are not strictly simultaneous due to the processing overhead time for starting the statement. Therefore, the speed declines at the switching point in the path motion, and the motion will not be constant velocity.
CP_Offset solves this problem by accelerating the starting time of the subsequent motion command.



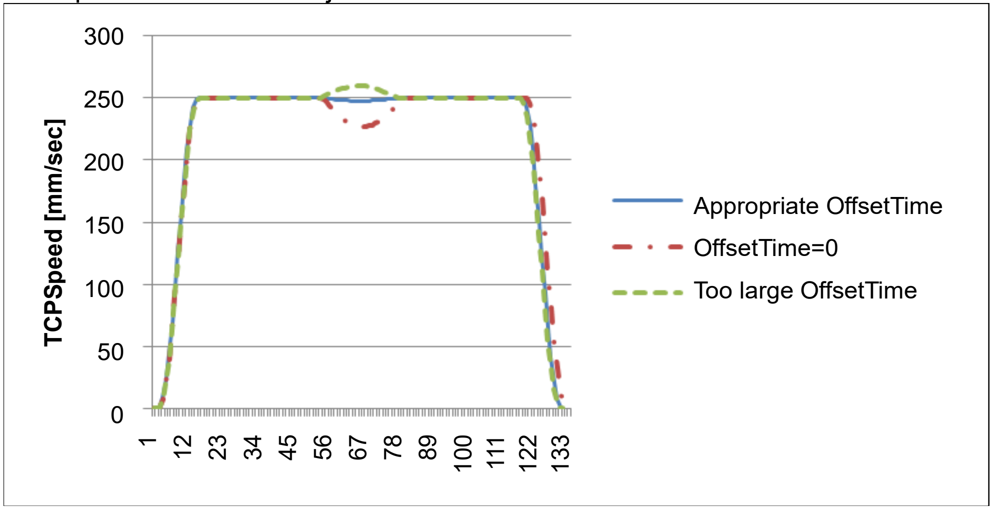
By setting CP_Offset on, the processing start of the subsequent motion command will be accelerated by the time specified for the OffsetTime parameter, and deceleration start of the actual robot and acceleration start of the subsequent command will be synchronized. As a result, the constant velocity can be improved.
The OffsetTime parameter is set by default. Adjust the parameter according to your application.
Especially when the subsequent motion command has “!Parallel Processing!”, the overhead time required for the motion start gets longer. Therefore, set the OffsetTime parameter higher than the default value, approximately 16 ms.

To set the OffsetTime parameter for CP_Offset, measure the speed of the tool center point for the target motion by using TCPSpeed. Setting an appropriate value for the OffsetTime parameter improved the motion at the switching point to be close to constant.
TCPSpeed increases when OffsetTime is too large, and TCPSpeed decreases when OffsetTime is too small.
Adjustment of CP_Offset must be done in actual system. Appropriate adjustment cannot be done in the simulator because the processing time to start the command differs from the actual controller.

Sample program for measurement using TCPSpeed

```
Function main
    Motor On
    Power High

    SpeedS 250; AccelS 1500
    Speed 50; Accel 50, 50

    Go XY(300, 500, 500, 90, 0, 180)

    CP_Offset On
    Xqt printTcPSpeed

    Move XY(0, 500, 500, 90, 0, 180) CP
    Move XY(-300, 500, 500, 90, 0, 180)

    Quit printTcPSpeed
    CP_Offset Off
Fend

Function printTcPSpeed
    Do
          Print TCPSpeed
    Loop
Fend
```

Example of OffsetTime adjustment



This command is not intended for PTP motion. In PTP motion, the motion will be an usual path motion.

CP_Offset is off when any of the following conditions occur:

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or Quit All stops tasks

See Also

CP_Offset Function, CP, Move, Arc, Arc3, CVMove

CP_Offset Statement Example

```
CP_Offset On
Move P1
Move P2
CP_Offset Off
```

CP_Offset Function

Returns the offset time to start the subsequent motion command when executing CP On.

Syntax

CP_Offset

Return Values

Real number representing the offset time to start the motion command.

See Also

CP_Offset Statement

CP_Offset Function Example

```
If CP_Offset = 0 Then
    Print "CP_Offset is off"
EndIf
```

Ctr Function

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. (Integer expression from 0-65535)

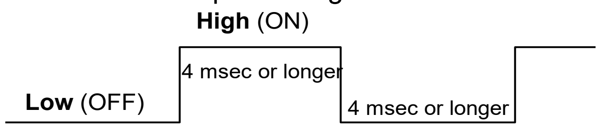
Description

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
Wait Ctr(3) >= 5
Off 0      'When 5 input cycles are counted for Input 3 turn switch off (output 0 off)
```

CTReset Statement

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 Statement 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
Wait Ctr(3) >= 5
Off 0       'When 5 input cycles are counted for Input 3 turn switch off (output 0 off)
```

CtrlDev Function

Returns the current control device number.

Syntax

CtrlDev

Return Values

21	PC
22	Remote I/O
26	Remote Ethernet
29	Remote RS232C
20	TP3

See Also

CtrlInfo Function

CtrlDev Function Example

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

CtrlInfo Function

Returns controller information.

Syntax

CtrlInfo (*index***)**

Parameters

index Integer expression that represents the index of the information to retrieve.

Description

The following table shows the information that is available from the CtrlInfo function:

Index	Bit	Value	Description
0	N/A		Obtained for compatibility. Use index 9 to get the firmware version of the controller.
1	Controller status		
	0	&H1	Ready state
	1	&H2	Start state
	2	&H4	Pause state
	3-7		Undefined
	8	&H100	Estop state
	9	&H200	Safeguard open
	10	&H400	Error state
	11	&H800	Critical error state
	12	&H1000	Warning
	13	&H2000	WaitRecover state (Waiting for recover from safeguard open)
	14	&H4000	Recover state (Recovering from the safeguard open)
	15-31		Undefined
2	0	&H1	Enable switch is on
	1-31		Undefined
3	0	&H1	Teach mode circuit problem detected
	1	&H2	Safeguard circuit problem detected
	2	&H4	Estop circuit problem detected
	3-31		Undefined
4	N/A		0 – Normal mode 1 – Dry run mode
5	N/A		Control device: 21 – RC+ 22 – Remote 26 – Remote Ethernet 29 – Remote RS232C 20 – TP3
6	N/A		Number of defined robots
7	N/A		Operation mode: 0 – Program mode 1 – Auto mode
8	N/A		Undefined
9	N/A		Firmware version of the Controller Major No.*1000000 + Minor No.*10000 + Rev No.*100 + Build No. (Example) Version 1.6.2.4 is 1060204

Index	Bit	Value	Description
10	N/A		SMART status of hard disk 0 : SMART status is normal 1 : SMART status is not normal If SMART status is not normal, the hard disk can be broken. You need to back up the data promptly and replace the hard disk with new one. When using the RAID option, you cannot use the SMART status, it always returns that it is normal.
15	N/A		Input DC Voltage The program returns the value 100 times greater than the input value. For example, when the input value is 48.01V, it returns 4801. Note that an error occurs if Controller does not support DC power supply.
16	N/A		PLC vender type 0: None 1: Allen Bradley 2: CODESYS

Return Values

Long value of the desired data

See Also

RobotInfo, TaskInfo

CtrlInfo Function Example

```
Print "The controller version: ", CtrlInfo(6)
```

CurDir$ Function

Returns a string representing the current directory.

Syntax

CurDir$

Return Values

A string that includes the current drive and path.

See Also

ChDir, CurDrive$, CurDisk$

CurDir$ Function Example

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

CurDisk$ Function

Returns a string representing the current disk.

Syntax

CurDisk$

Return Values

A string that contains the current disk letter.

See Also

ChDisk, CurDir$, CurDrive$

CurDisk$ Function Example

```
Print "The current disk is: ", CurDisk$
```

CurDrive$ Function

Returns a string representing the current drive.

Syntax

CurDrive$

Return Values

A string that contains the current drive letter.

See Also

ChDrive, CurDir$, CurDisk$

CurDrive$ Function Example

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

CurPos Function

Returns the current target position of the specified robot.

Syntax

CurPos

Return Values

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

See Also

InPos, FindPos, RealPos

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

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 name of the file in which the point data is stored. The specified fileName will have the extension .CVT appended to the end so no extension is to be specified by the user. When the Curve instruction is executed, file will be created.

You cannot specify a file path and fileName doesn't have any effect from ChDisk.
See ChDisk for the details.

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.

When generating a curve considering the additional axis position included in the point data, specify the ninth bit as 1. For example, when using no orientation offset or ECP and generating a curve considering the additional axis position, specify &H100.

When generating a curve for the additional axis, join the continuous point data of S axis and T axis separately from the robot coordinate system.

However if the additional axis is consisted of the PG axis, it doesn't generate a curve with the continuous point but creates the data to move to the final point.

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. (except for 6-Axis robots (including N series))

3 - Generate a curve in the XYZ space with no U axis rotation. (except for 6-Axis robots (including N series))

4 - Generate a curve in the XYZ space with U-Axis rotation. (except for 6-Axis robots (including N series))

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

The curve file is stored in the Compact Flash inside of the controller. Therefore, Curve starts writing into the Compact Flash. Frequent writing into the Compact Flash will shorten the Compact Flash lifetime. We recommend using Curve only for saving the point data.

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 any time 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.

Notes

Use tangential correction when possible

It is recommended that you use tangential correction whenever possible, especially when using CVMove in a continuous loop through 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 1000 points.

Closed Curve Min and Max Number of Points Allowed

For the controller of RC700 series and RC90 series, Closed Curves may be specified by using from 3 to 1000 points.
For the T/VT series controller, Closed Curves may be specified by using from 3 to 300 points. (If using T/VT controller in the simulator, it can be specified from 3 to 1000 points, but with the actual controller, it is up to 300.)

The Processing Time Gets Longer If the Number of Point Is Large

If the Curve command executed with max points, it takes several seconds for Closed Curves, several tens of seconds for Open Curve.
Especially, Closed Curves takes lots of time to process so using Open Curve is recommended when the number of points is large.
Example of creating trajectory similar to Closed Curve with using Open Curve is described in the Curve Statement Example 2.
However, if you generate a curved path file once with the Curve command and perform CVMove motion multiple times on the same file, the above time will be taken only once when the Curve command is executed.

Compatibility of file

Files created after firmware Ver.7.5.1 are not available for earlier versions of firmware. Also, files created firmware Ver.7.5.1 or earlier can be used with firmware Ver.7.5.1 or later.

Potential Error

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 1

The following example designates the free curve data file name as MYCURVE.CVT, 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"
```

Curve Statement Example 2

Shown below is an example for motion that similar to Closed Curve by using (1) Open Curve, (2) Closed Curve, and (3) Open Curve.
Reference points are as shown below.

```
     P0  = XY(0, 300, -50, 0) 'Start/End
     P1  = XY(300, 200, -50, 0)
     P2  = XY(300, 400, -50, 0)
     P3  = XY(-300, 400, -50, 0)
     P4  = XY(-300, 200, -50, 0)
     P10 = XY(10, 299.7, -50, 0)    'Right after start
     P11 = XY(-10, 299.7, -50, 0)   'Right before end
```

(1) Open Curve

Specifying a curve of Open Curve

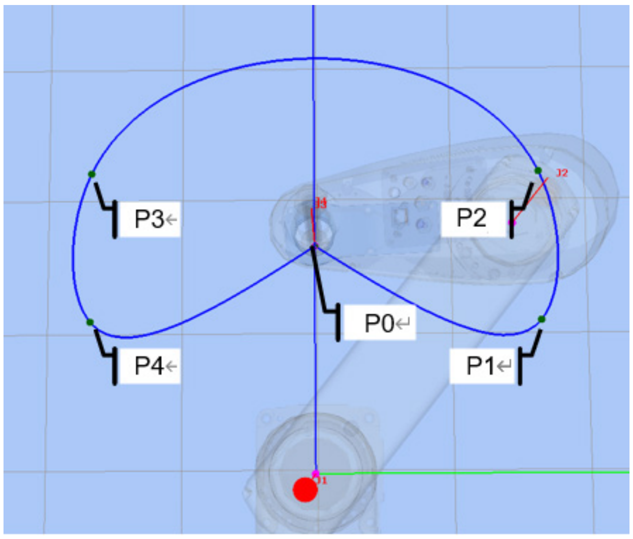
```
> Curve "mycurve_O", O, 0, 2, P(0:4), P0
```

Moving the arm in a straight line to P0

```
> jump P0
```

Move the arm with the specified Open Curve "mycurve_O"

```
> CVMove "mycurve_O"
```



Because it is an Open Curve, the start point and end point are the same but they do not connect smoothly.

(2) Closed Curve

Specifying a curve of Closed Curve

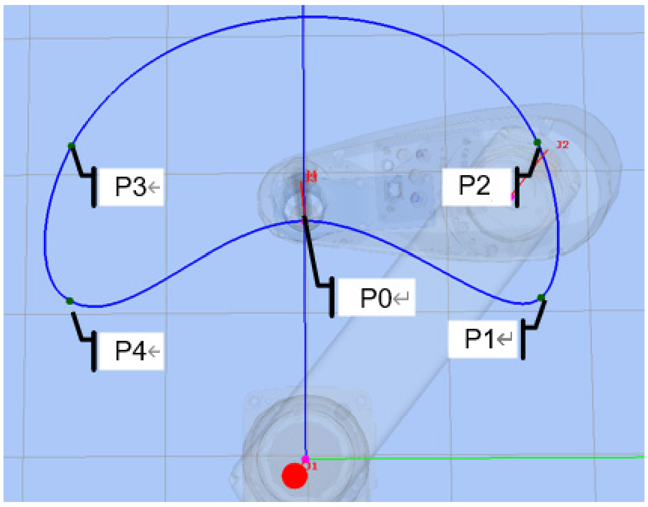
```
> Curve "mycurve_C", O, 0, 2, P(0:4)
```

Moving the arm in a straight line to P0

```
> jump P0
```

Move the arm with the specified Closed Curve “mycurve_C”

```
> CVMove "mycurve_C"
```



Because it is Closed Curve, the start point and end point connect smoothly.

(3) OpenCurve similar to the Closed Curve

Specifying a curve of Open Curve. Set the points right after the start point and right before the end point.

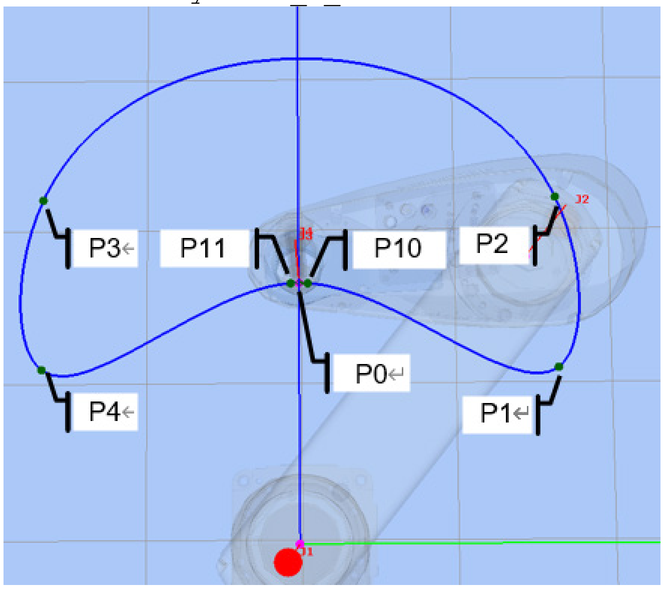
```
> Curve "mycurve_O_mod", O, 0, 2, P0, P10, P(1:4), P11, P0
```

Moving the arm in a straight line to P0

```
> jump P0
```

Move the arm with the specified Open Curve “mycurve_O_mod”

```
> CVMove "mycurve_O_mod"
```



It is Open Curve but the start point and end point connect smoothly by going through P10 and P11.

CVMove Statement

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

Syntax

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

Parameters

fileName	String expression for the file name. This file must be previously created by the Curve instruction and stored on a PC hard disk. You cannot specify a file path and fileName doesn't have any effect from ChDisk. See ChDisk for the details.
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**}

SYNC	Reserves a motion command. A robot will not move until the SyncRobots gives instructions.

Description

CVMove performs the continuous spline path motion defined by the data in the file *fileName*, which is located in the controller memory. The file must be previously created with the Curve command.
Multiple files may exist at the same time on the system. If the file name does not have an extension, .CVT is added automatically.
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 (including N series) 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 (arm) orientation between adjacent points suddenly.
The CP parameter causes acceleration of the next motion command to start when the deceleration starts for the current motion command. In this case the robot will not stop at the destination coordinate and will continue to move to the next point.

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.CVT, 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, CR, CS, CT Statements

Sets the coordinate value of a point data.
CV, CW are for only 6-axis robots (including N series).
CR is only for Joint type robots.
CS, CT are only for robots with additional axes.

Syntax

CX(*point***)** = *value*
CY(*point***)** = *value*
CZ(*point***)** = *value*
CU(*point***)** = *value*
CV(*point***)** = *value*
CW(*point***)** = *value*
CR(*point***)** = *value*
CS(*point***)** = *value*
CT(*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, CR, CS, CT Functions

CX, CY, CZ, CU, CV, CW, CR, CS, CT Statements Example

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

CX, CY, CZ, CU, CV, CW, CR, CS, CT Functions

Retrieves a coordinate value from a point
CV, CW functions are only for 6-axis robots (including N series).
CS, CT are only for robots with additional axes.

Syntax

CX (*point***)**
CY (*point***)**
CZ (*point***)**
CU (*point***)**
CV (*point***)**
CW (*point***)**
CR (*point***)**
CS (*point***)**
CT (*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.
Return values of CS, CT functions: Real values in mm or deg. It depends on the additional axis setting.

Description

Used to retrieve an individual coordinate value from a point.

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

See Also

CX, CY, CZ, CU, CV, CW, CR, CS, CT Statements

CX, CY, CZ, CU, CV, CW, CR, CS, CT 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

Displays the date.

Syntax

Date

Return Values

The current date is displayed.

See Also

Time, Date$

Date Statement Example

Example from the command window.

```
> Date
2009/08/01
```

Date$ Function

Returns the system date.

Syntax

Date$

Return Values

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

See Also

Date, Time, Time$

Date$ Function Example

```
Print "Today's date: ", Date$
```

Declare Statement

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	Optional. 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. If omitted, a function name specified by *funcName* can be used as a name of function in DLL.
arglist	Optional. List of the DLL arguments. See syntax below.

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

ByRef	Optional. Specify ByRef when you refer to the variable to be seen by the calling function. In this case, the argument change in a function can be reflected to the variable of the calling side. You can change the values received as a reference.
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. If you use an array variable as argument, you must specify ByRef.
varType	Required. You must declare the type of argument.

type	Required. You must declare the type.

Description

Use Declare to call DLL functions from the current program. 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 255 characters. You must ensure that you do not exceed the maximum length.

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 returning value for any data type, except String. When it is needed to return string, refer to “Passing String Variables ByRef” described above and specify string variables as an argument.
If string variables are specified to a returning value, error 3614 “You cannot specify a String for Declare return data type.” will occur.

VarType

Following shows table of data type of EPSON RC+ 7.0 and variable type of C/C++.
Since there is no data for EPSON RC+ 7.0, byte type of C/C++ and structure cannot be used.

Table of data type for EPSON RC+ 7.0 and C/C++

EPSON RC+ 7.0	C/C++
Boolean	short
Byte	short
Short	short
Integer	short
Long	int
Real	float
Double	double
String	char [256] * includes Null

For example:

```
    Declare ReturnLong, "mystuff.dll", "ReturnLong", As Long

    Function main

        Print "ReturnLong = ", ReturnLong
    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

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 Statement

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. See ChDisk for the details.

Description

Deletes the specified file(s).

Del Statement Example

Example from the command window.

```
> Del TEST.PTS          ' Deletes the point file from the current directory.

> Del  c :TEST.PTS      ' NG
!! Error: 7213 The file specified by path does not exist.
> Del  c :\TEST.PTS     ' OK
```

DeleteDB Statement

Deletes data from the table in the opened database.

Syntax

DeleteDB *#databaseNum*,*tableNumber* [*, condition*]

Parameters

databaseNum	Specify the database number (integer from 501 to 508) specified in OpenDB.
tableNumber	Specify the table name whose data will be deleted.
condition	Specify the condition to delete the data. Compound condition can be specified by using AND and OR. If the condition is not specified, all data in the table will be deleted.

Description

Deletes the data matched to the delete condition from the specified table in the opened database.
If the database is an Excel book, this command cannot be executed.

Note

- Connection of PC with installed RC+ is required.

See Also

OpenDB, CloseDB, SelectDB, UpdateDB

DiffPoint Function

Returns the difference between two specified points.

Syntax

DiffPoint (*pointData1*, *pointData2*)

Parameters

pointData1	Specifies the first point data.
pointData2	Specifies the second point data.

Return Values

Returns the position and orientation of pointData2 as seen from pointData1.

Description

Returns the position and orientation of pointData2 in a coordinate system with pointData1 as the origin.
This function uses default values as returned the local number of the point data, and flag information such as Hand.
If point data is left undefined for either point data, the undefined values will be calculated as “0”.
For example, if pointData1 is specified as “XY (10,0,0,0,0,0): ST (10, 10)”, and pointData2 is specified as “XY (10,0,0,0,0,0)”, the S and T values are undefined for pointData2 while the values are defined for pointData1, and thus the values calculated with the S and T values for pointData2 as “0” will be returned.

Notes

About the Controllers to use

It cannot be used with T/VT series.

DiffPoint Function Example

```
'Display the position and orientation of P2 as seen from P1.

Print DiffPoint(P1, P2)

'Display the position and orientation of P1 as seen from the current position (Here).

Print DiffPoint(Here, P1)
```

DiffToolOrientation Function

Returns the angle between the coordinate axes of Tool coordinate systems in order to show difference between Tool orientations of two specified points.

Syntax

DiffToolOrientation (*pointData1*, *pointData2* , *axisNumber*)

Parameters

pointData1	Specify the first point data.
pointData2	Specify the second point data.
axisNumber	Specify the coordinate axis of Tool coordinate system.

Constant	Value	
COORD_X_PLUS	1:	+X axis
COORD_Y_PLUS	2:	+Y axis
COORD_Z_PLUS	3:	+Z axis
COORD_ALL	4:	Airbitrary axis

Return Values

Angle (real value, from 0 to 180 degrees)

Description

Returns the angle (real value, from 0 to 180 degrees) between the specified coordinate axes of the Tool coordinate systems which indicates the difference between Tool orientations of two specified points. The results are not affected by the order of parameters, *pointData1* and *pointData2*. The results are also not affected by positional relation (coordinate values of X, Y, and Z) between the origin points of the two points.
Returns a rotation amount around an arbitrary axis when COORD_ALL is specified. An arbitrary axis refers to a hypothetical axis (a straight line) around which the robot can move in a single rotation when two orientations are provided (U, V, W). This function is used to find the overall angle of rotation without limiting rotation to each axis.

Notes

About the Controllers to use

COORD_ALL cannot be specified as the axis number for T/VT series.

DiffToolOrientation Function Example

'Displays the angle between Tool coordinate Z axes of Point 1 and 2.

```
Print DiffToolOrientation(P1, P2, COORD_Z_PLUS)
```

DispDev Statement

Sets the current display device.

Syntax

DispDev (*deviceID*)

Parameters

deviceID	The device ID for the desired display device. 21 RC+ 24 TP (TP1 only) 20 TP3 The following parameters are also available. 21 DEVID_SELF 24 DEVID_TP 20 DEVID_TP3

See Also

DispDev Function

DispDev Statement Example

```
    DispDev DEVID_TP
```

DispDev Function

Returns the current display device.

Syntax

DispDev

Return Values

Integer value containing the deviceID.

21	RC+
24	TP (TP1 only)
20	TP3

See Also

DispDev Statement

DispDev Function Example

```
Print "The current display device is ", DispDev
```

Dist Function

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).

Description

Even if you are using the additional axis, only the robot travel distance is returned.
It doesn’t include the travel distance of additional axis while you use the additional axis as running axis.
For the Joint type robot, the return value of this function means nothing.

See Also

CU, CV, CW, CX, CY, CZ

Dist Function Example

```
Real distance

distance = Dist(P1, P2)
```

Do...Loop Statement

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.

Note

DO NOT use XQT command repeatedly in Loop statements.

Do not use XQT command repeatedly in Loop statements such as Do…Loop.
The controller may freeze up. If you use Loop statements repeatedly, make sure to add Wait command (Wait 0.1).

Avoid endless execution of empty Loop Statements and similar to them, use them with the Wait command instead

Do not use empty Do…Loop statements and similar commands to avoid effect on the system. The Controllers are detecting endless loop tasks. If the controller determines that the system will be affected, it will stop the program with error 2556 (An excessive loop was detected). When performing operations that require a loop or waiting for I/O, execute a Wait command (Wait 0.1) and more within the loop to avoid occupying the CPU.

When you exit the loop from the nested structure without using Exit Do

Error 2020 will occur when you repeatedly execute the program which exits the loop by the command other than the Exit For command (such as GoSub statement, Goto statement, and Call statement.) Be sure to use Exit Do command to exit the loop.

See Also

For...Next, Select...Send

Do...Loop Statement Example

```
Do While Not Lof(1)
    Line Input #1, tLine$
    Print tLine$
Loop
```

Double Statement

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 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

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.
Valid number of digits for Double is 14.

See Also

Boolean, Byte, Global, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Double Statement 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(5, 5, 5)    '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
```

ECP Statement

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

Syntax

(1) **ECP** *ECPNumber*
(2) **ECP**

Parameters

ECPNumber	Optional. Integer expression from 0 to 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 active.

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 active.

See Also

ECP Statement

ECP Function Example

```
Integer savECP

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

ECPClr Statement

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.)

Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Note

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

See Also

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

ECPClr Statement Example

```
ECPClr 1
```

ECPDef Function

Returns ECP definition status.

Syntax

ECPDef (*ECPNumber*)

Parameters

ECPNumber	Integer expression representing which ECP to return status for.

Return Values

True if the specified ECP has been defined, otherwise False.

See Also

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

ECPDef Statement Example

```
Function DisplayECPDef(ecpNum As Integer)

   If ECPDef(ecpNum) = False Then
      Print "ECP ", ecpNum, "is not defined"
   Else
      Print "ECP ", ecpNum, ": ",
      Print ECPSet(ecpNum)
   EndIf
Fend
```

ECPSet Statement

Defines or displays an external control point.

Syntax

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

Parameters

ECPNum	Integer number from 1 to 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.
When only the ECP number is specified, displays the specified ECPSet definitions.

Description

Defines an external control point.
Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Note

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

ECPSet Statement Example

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

ECPSet Function

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 active.

See Also

ECPSet Statement

ECPSet Function Example

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

ElapsedTime Function

Returns the elapsed time since the takt time measurement timer starts in seconds.

Syntax

ElapsedTime

Return Values

An actual value representing an elapsed time of a takt time measurement timer. (Unit: second)
Valid range is from 0 to approx. 1.7E+31. Timer resolution is 0.001 seconds.

Description

Returns an elapsed time since the takt time measurement timer starts. Unlike the Tmr function, the ElapsedTime function does not count the time while the program is in pause state.

The takt time measurement timer can be reset by using ResetElapsedTime statement.

```
Real overhead

ResetElapsedTime
overHead = ElapsedTime
```

See Also

ResetElapsedTime, Tmr Function

ElapsedTime Function Example

```
ResetElapsedTime        'Resets the takt time measurement timer
For i = 1 To 10         'Executes 10 times
    GoSub Cycle
Next
Print ElapsedTime / 10 'Measures a takt time and displays it
```

Elbow Statement

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

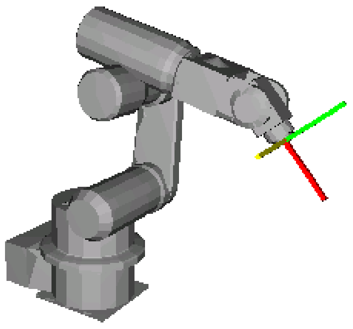
When both parameters are omitted, the elbow orientation is displayed for the current robot position.
If *value* is omitted, the elbow orientation for the specified point is displayed.

See Also

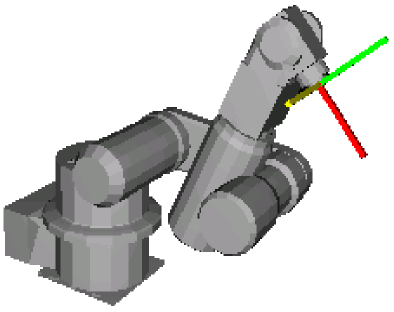
Elbow Function, Hand, J4Flag, J6Flag, Wrist

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

Returns the elbow orientation of a point.

Syntax

Elbow [(*point*)]

Parameters

point	Optional. Point expression. If *point* is omitted, 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)
```

Eof Function

Returns end of file status.

Syntax

Eof (*fileNumber*)

Parameters

fileNumber	Integer number from 30 to 60 or expression representing the file number to check.

Return Values

True if file pointer is at end of file, otherwise False.

Description

Eof is functional only if the file is opened for reading mode.
An error occurs if the file was opened with the AOpen or WOpen statements.

See Also

Lof

Eof Function Example

```
Integer fileNum
String data$

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
Do While Not Eof(fileNum)
    Line Input #fileNum, data$
    Print "data = ", data$
Loop
Close #fileNum
```

Era Function

Returns the joint number for which an error occurred.

Syntax

Era[(*taskNum*)]

Parameters

taskNum	Integer expression representing a task number from 0 to 32. Task number omission or "0" specifies the current task.

Return Values

The joint number that caused the error in the range 0 to 9 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
7 - The error was caused by joint number 7
8 - The error was caused by joint number 8 (additional S axis)
9 - The error was caused by joint number 9 (additional T axis)

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 "0".
When the event "Error during Auto Mode" occurs, normal task and NoPause task in AUTO mode stop execution and end the task.
If the target task has already ended when using this function for NoEmgAbort task or background task, "Error 2261" is occurred. Use OnErr to acquire information before the task ends.

See Also

Erl, Err, ErrMsg$, Ert, OnErr, Trap

Era Function Example

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  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 the error is ", Era(errTask)
    EndIf
  EndIf
Fend
```

EResume Statement

Resumes execution after an error-handling routine is finished.

Syntax

EResume [{ *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 { *label* }

If the error occurred in the same procedure as the error handler, execution resumes at the statement containing the label.

See Also

OnErr

EResume Statement Example

```
Function main
  Integer retry

  OnErr GoTo eHandler
  Do
    RunCycle
  Loop
  Exit Function

eHandler:
  Select Err
    Case MyError
      retry = retry + 1
      If retry < 3 Then
        EResume ' try again
      Else
        Print "MyError has occurred ", retry, " times
      EndIf
  Send
Fend
```

Erf$ Function

Returns the name of the function in which the error occurred.

Syntax

Erf$[(*taskNumber*)]

Parameters

taskNumber	Integer expression representing a task number from 0 to 32. Task number omission or "0" specifies the current task.

Return Values

The name of the function where the last error occurred.

Description

Erf$ is used with OnErr. Erf$ returns the function name 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.
When the event "Error during Auto Mode" occurs, normal task and NoPause task in AUTO mode stop execution and end the task.
If the target task has already ended when using this function for NoEmgAbort task or background task, "Error 2261" is occurred. Use OnErr to acquire information before the task ends.

See Also

Era, Erl, Err, ErrMsg$, Ert, OnErr

Erf$ Function Example

The Following items are returned in the program example below.
In which task the error occurred (Ert function)
In which function the error occurred (Erf$ function)
Where the error occurred (Erl function)
On which joint the error occurred (Era function)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "Function at which error occurred is ", Erf$(errTask)
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
  EndIf
Fend
```

Erl Function

Returns the line number in which the error occurred.

Syntax

Erl[(*taskNumber*)]

Parameters

taskNumber	Integer expression representing a task number from 0 to 32. Task number omission or "0" specifies the current task.

Return Values

The line number where the last error occurred.

Description

Erl is used with OnErr. 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.
When the event "Error during Auto Mode" occurs, normal task and NoPause task in AUTO mode stop execution and end the task.
If the target task has already ended when using this function for NoEmgAbort task or background task, "Error 2261" is occurred. Use OnErr to acquire information before the task ends.

See Also

Era, Erf$, Err, ErrMsg$, Ert, OnErr

Erl Function Example

The Following items are returned in the program example below.
  In which task the error occurred (Ert function)
  Where the error occurred (Erl function)
  What error occurred (Err function)
  On which joint the error occurred (Era function)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  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 the error is ", Era(errTask)
    EndIf
  EndIf
Fend
```

Err Function

Returns the most recent error status.

Syntax

Err [(*taskNumber*)]

Parameters

taskNumber	Optional. Integer expression representing a task number from 0 to 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.
To get the controller error, use SysErr function.
When the event "Error during Auto Mode" occurs, normal task and NoPause task in AUTO mode stop execution and end the task.
If the target task has already ended when using this function for NoEmgAbort task or background task, "Error 2261" is occurred. Use OnErr to acquire information before the task ends.

See Also

Era, Erf$, Erl, ErrMsg$, EResume, Ert, OnErr, Return, SysErr

Err Function 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."
   EndIf
   EResume Next
Fend
```

Errb Function

Returns the robot number which the error occurred.

Syntax

Errb

Return Values

Returns the robot number which the error occurred.

Description

Errb finds and returns the robot number where the error occurred. If the robot is not the cause of the error, "0" will be returned.

See Also

Era, Erl, Err, ErrMsg$, OnErr, Trap

Errb Function Example

The Following items are returned in the program example below.

In which task the error occurred (Ert function)
Where the error occurred (Erl function)
What error occurred (Err function)
On which joint the error occurred (Era function)
On which robot the error occurred (Errb function)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  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 the error is ", Era(errTask)
    EndIf
  Print "Robot number in which error occurred is ", errb
  EndIf
Fend
```

ErrMsg$ Function

Returns the error message which corresponds to the specified error number.

Syntax

ErrMsg$(*errNumber, langID*)

Parameters

errNumber	Integer expression containing the error number to get the message for.
langID	Optional. Integer expression containing the language ID based on the following values. 0 - English 1 - Japanese 2 - German 3 - French 4 – Simplified Chinese 5 – Traditional Chinese 6 – Spanish If omitted, English is used.

Return Values

Returns the error message which is described in the Error Codes table.

See Also

Era, Erl, Err, Ert, OnErr, Trap

ErrMsg$ Function Example

The Following items are returned in the program example below.

In which task the error occurred (Ert function)
Where the error occurred (Erl function)
On which joint the error occurred (Era function)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  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 the error is ", Era(errTask)
    EndIf
  EndIf
Fend
```

Error Statement

Generates a user error.

Syntax

(1) **Error** *task Number, errorNumber*
(2) **Error** *errorNumber*

Parameters

taskNumber	Optional. Integer expression representing a task number from 0 to 32. "0" specifies the current task.
errorNumber	Integer expression representing a valid error number. User error numbers range is from 8000 to 8999.

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

Era, Erl, Err, OnErr

Error Statement Example

```
#define ER_VAC 8000

If Sw(vacuum) = Off Then
    Error ER_VAC
EndIf
```

ErrorOn Function

Returns the error status of the controller.

Syntax

ErrorOn

Return Values

True if the controller is in error status, otherwise False.

Description

ErrorOn function is used only for NoEmgAbort task (special task using NoEmgAbort at Xqt) and background task.

See Also

ErrorOn, SafetyOn, SysErr, Wait, Xqt

ErrorOn Function Example

The following example shows a program that monitors the controller error and switches the I/O On/Off according to the error number when error occurs.

Notes

Forced Flag

This program example uses Forced flag for On/Off command.
Be sure that the I/O outputs change during error, or at Emergency Stop or Safety Door Open when designing the system.

After Error Occurrence

As this program, finish the task promptly after completing the error handling.

```
Function main

    Xqt ErrorMonitor, NoEmgAbort
    :
    :
Fend

Function ErrorMonitor
    Wait ErrorOn
    If 4000 < SysErr Then
          Print "Motion Error = ", SysErr
          Off 10, Forced
          On 12, Forced
    Else
          Print "Other Error = ", SysErr
          Off 11, Forced
          On 13, Forced
    EndIf

Fend
```

Ert Function

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.
Ert returns the number as follows:
No task with error (0), normal task (1 to 32), back ground task (65 to 80), TRAP task (257 to 267).

See Also

Era, Erl, Err, ErrMsg$, OnErr, Trap

Ert Function Example

The Following items are returned in the program example below.
In which task the error occurred (Ert function)
Where the error occurred (Erl function)
On which joint the error occurred (Era function)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  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 the error is ", Era(errTask)
    EndIf
  EndIf
Fend
```

EStopOn Function

Returns the Emergency Stop status.

Syntax

EstopOn

Return Values

True if the status is Emergency Stop, otherwise False.

Description

EStopOn function is used only for NoEmgAbort task (special task using NoEmgAbort at Xqt).

See Also

ErrorOn, SafetyOn, Wait, Xqt

EstopOn Function Example

The following example shows a program that monitors the Emergency Stop and switches the I/O On/Off when Emergency Stop occurs.

Notes

Forced Flag

This program example uses Forced flag for On/Off command.
Be sure that the I/O outputs change during error, or at Emergency Stop or Safeguard Open when designing the system.

Error Handling

As this program, finish the task promptly after completing the error handling.

Outputs OFF during Emergency Stop

As this program example, when the task executes I/O On/Off after the Emergency Stop, uncheck the [Setup]-[System Configuration]-[Controller]-[Preferences]-[Outputs off during emergency stop] check box. If this check box is checked, the execution order of turn Off by the controller and turn On using the task are not guaranteed.

```
Function main

    Xqt EStopMonitor, NoEmgAbort
    :
    :
Fend

Function EStopMonitor
    Wait EStopOn
    Print "EStop !!!"
    Off 10, Forced
    On 12, Forced
Fend
```

Eval Function

Executes a Command window statement from a program and returns the error status.

Syntax

Eval(*command* [, *reply$*])

Parameters

command	A string expression containing a command you want to execute.
reply$	Optional. A string variable that contains the reply from the command. If the command is in the error status, it will return "!Error: error code". If the reply is over 255 characters, the extra characters will be truncated.

Return Values

The error code returned from executing the command.
Even if the command execution results in an error, the function itself will not be an error. Also, the system log doesn't record it.
When the command is completed successfully, it returns "0".

Description

You can execute any command (executable commands from Command window) from communication port such as TCP/IP by using Eval. It takes more time to execute this function than by using a normal statement.

Use the *reply$* parameter to retrieve the reply from the command. For example, if the command was "Print Sw(1)", then *reply$* would be 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

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
```

ExportPoints Statement

Exports a point file to the specified path.

Syntax

ExportPoints *fileName, destination*

Parameters

fileName	String expression containing the specific file to be exported. The extension must be ".pts". You cannot specify a file path and *fileName* doesn't have any effect from ChDisk. See ChDisk for the details.
destination	Specify the path and file name to save the file. The extension must be ".pts". See ChDisk for the details.

Description

ExportPoints copies a specified point file to a folder on the PC.
If the file already exists in the folder, it will be overwritten.

Potential Errors

File Does Not Exist

If the specified path does not exist, an error will occur.

A Path Cannot be Specified

If *fileName* contains a path, an error will occur.

See Also

Dir, LoadPoints, SavePoints, FileExists, FolderExists

ExportPoints Statement Example

```
Function main
  LoadPoints "robot1.pts"
   :
  SavePoints "robot1.pts"
  If FolderExists("c:\mypoints\") Then
     ExportPoints "robot1.pts", "c:\mypoints\model1.pts"
  EndIf
Fend
```

FbusIO_GetBusStatus Function

Returns the status of the specified Fieldbus.

Syntax

FbusIO_GetBusStatus(*busNumber*)

Parameters

busNumber	Integer expression representing the Fieldbus system number. This number must be 16. This is the ID for the bus connected to the Fieldbus master board on the PC side of the controller.

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 Master option is active.

See Also

FbusIO_GetDeviceStatus, FbusIO_SendMsg

FbusIO_GetBusStatus Function Example

```
Long sts
sts = FbusIO_GetBusStatus(16)
```

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 number must be 16. This is the ID for the bus connected to the Fieldbus master board on the PC side of the controller.
deviceID	Integer expression representing the Fieldbus ID of the device.

Return Values

0 - OK
1 - Disconnected
2 - Power off
3 - Synchronization 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 Master option is active.

See Also

FbusIO_GetBusStatus, FbusIO_SendMsg

FbusIO_GetDeviceStatus Function Example

```
Long sts
sts = FbusIO_GetDeviceStatus(16, 10)
```

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 number must be 16. This is the ID for the bus connected to the Fieldbus master board on the PC side of the controller.
deviceID	Integer expression representing the Fieldbus ID of the device.
msgParam	Integer expression for the message parameter. Not used with DeviceNet.
sendData	Array of type Byte containing data that is sent 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 Master option is active.

See Also

FbusIO_GetBusStatus, FbusIO_GetDeviceStatus

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 16, 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 16, 1, 56, 0, recvData()
' Display the reply
For i = 0 to UBound(recvData)
  Print recvData(i)
Next i
```

FileDateTime$ Function

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. If only file name is specified, the file in the current directory is displayed. See ChDisk for the details.

Note

A network path is available.

Return Values

Returns the date and time of the last update in the following format:

m/d/yyyy hh:mm:ss

See Also

FileExists, FileLen

FileDateTime$ Function Example

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```

FileExists Function

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. If only the file name is specified, the file is checked in the current directory. See ChDisk for the details.

Note

A network path is available.

Return Values

True if the file exists, False if not.

See Also

FolderExists, FileLen, FileDateTime$

FileExists Function Example

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```

FileLen Function

Returns the size of a file.

Syntax

FileLen (*filename*)

Parameters

fileName	A string expression containing the file name to check. This includes both drive name and path name. If only the file name is specified, the file is checked in the current directory. See ChDisk for the details.

Note

A network path is available.

Return Values

Returns the number of bytes in the file.

See Also

FileDateTime$, FileExists

FileLen Function Example

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```

Find Statement

Specifies or displays the condition to store coordinates during motion.

Syntax

Find [*condition*]

Parameters

condition	Input status specified as a trigger

[*Event*] comparative operator (=, <>, >=, >, <, <=) [*Integer expression*]

The following functions and variables can be used in the *Event*:

Functions :	Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out, AIO_OutW Hand_On, Hand_Off, SF_GetStatus
Variables :	Byte, Inr32, Integer, Long, Short, UByte, UInt32, UShort global preserve variable, Global variable, module variable

In addition, using the following operators you can specify multiple event conditions.

Operator : And, Or, Xor

Example : Find Sw(5) = On
Find Sw(5) = On And Sw(6) = Off

Description

Find statement can be used by itself or as a modifier of a motion command.

The Find condition must include at least one of the functions above.

When variables are included in the Find condition, their values are computed when setting the Find condition. No use of variable is recommended. Otherwise, the condition may be an unintended condition. Multiple Find statements are permitted. The most recent Find condition remains current.

When parameters are omitted, the current Find definition is displayed.

Notes

Find Setting at Main Power On

At power on, the Find condition is:
Find Sw(0) = On 'Input bit 0 is on

Use of PosFound Function to Verify Find

Use PosFound function to verify if the Find condition has been satisfied after executing a motion command using Find modifier.

Use Variables in Event Condition Expression

- Available variables are Integer type (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)
- Array variables are not available
- Local variables are not available
- If a variable value cannot satisfy the event condition for more than 0.01 seconds, the system cannot retrieve the change in variables.
- Up to 64 can wait for variables in one system (including the ones used in the event condition expressions such as Wait). If it is over 64, an error occurs during the project build.
- If you try to transfer a variable waiting for variables as a reference with Byref, an error occurs.
- When a variable is included in the right side member of the event condition expression, the value is calculated when starting the motion command. We recommend not using variables in an integer expression to avoid making unintended conditions.

See Also

FindPos, Go, Jump, PosFound, SF_GetStatus

Find Statement Example

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
EndIf
```

FindPos Function

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."
EndIf
```

Fine Statement

Specifies and displays the positioning accuracy for target points.

Syntax

(1) **Fine** *axis1, axis2, axis3, axis4* [, *axis5*, *axis6*] [, *axis7*] [, *axis8*, *axis9*]
(2) **Fine**

Parameters

axis1	Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 1st joint.
axis2	Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 2nd joint.
axis3	Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 3rd joint.
axis4	Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 4th joint.
axis5	Optional. Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 5th joint. Only for the 6-axis robot (including N series).
axis6	Optional. Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 6th joint. Only for the 6-axis robot (including N series).
axis 7	Optional. Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 7th joint. Only for the Joint type 7-axis robot.
axis 8	Optional. Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 7th joint. Only for the additional S axis.
axis 9	Optional. Integer expression ranging from (0 to 65535) which represents the allowable positioning error for the 7th joint. Only for the additional T axis.

* For C8, C12 series Manipulators, the allowable positioning error is from 0 to 131070.

Return Values

When used without parameters, Fine displays the current fine values for each axis.

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 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 configuration. Positioning is considered complete when all axes have arrived within the specified ranges. Once positioning is complete program control is passed to the next statement, however, servo system keeps the control of the robot target position.

When relatively large ranges are used with the Fine instruction, the positioning will be confirmed relatively early in the move and executes the next statement.

The default Fine settings depend on the robot type. Refer to your Manipulator 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.

Initialization of Fine (by Motor On, SLock, SFree)

When any of the following commands is used, the Fine value will be initialized to the default:
SLock, SFree, Motor instructions.
Make sure that you reset Fine values after one of the above commands is executed.

Potential Error

If Fine positioning is not completed within about 2 seconds, Error 4024 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, FineDist, FineStatus

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          'reduces precision to +/- 5 Pulse
    Go P1
    Go P2
Fend

> Fine 10, 10, 10, 10
>
> Fine
10, 10, 10, 10
```

Fine Function

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. The additional S axis is 8 and T axis is 9.

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
```

FineDist Statement

Specifies and displays the positioning error limits. The unit of the setting value is “mm”.

Syntax

(1) **FineDist** *value*
(2) **FineDist**

Parameters

value	Positioning allowance ranges from 0.001[mm] to 10[mm].

Return Values

If the parameter is not specified, FineDist displays the current set value.

Notes

About the Controllers to use

It cannot be used with T/VT series.

Fine and FineDist

The difference between Fine and FineDist is the unit of the positioning check .
Fine statement sets the positioning check value in pulse, and the positioning check is performed on each axis.
FineDist statement sets the positioning check value in mm, and the positioning check is performed in the coordinate system of Tool number 0.
Fine and FineDist can be used at the same time. If Fine and FineDist are used in the program as shown below, the positioning check will be performed by FineDist. (If the order of Fine and FineDist is reversed, Fine will perform the positioning check.)

```
Function test
    Fine 5, 5, 5, 5
    FineDist 0.1

    Go P1
    Go P2
Fend
```

Note

Initialization of Fine (by Motor On, SLock, SFree)

When any of the following commands is used, the FineDist value will be initialized to the default and the positioning check will be performed by Fine:
SLock, SFree, Motor
Make sure to reset the FineDist value after any of the above commands is executed.

Potential Error

If FineDist positioning is not completed within about 2 seconds, Error 4024 will occur. This error normally means the servo system balance needs to be adjusted.

See Also

Accel, AccelR, AccelS, Arc, Go, Jump, Move, Speed, SpeedR, SpeedS, Pulse, Fine, FineStatus

FineDist Statement Example

The example below show the FineDist statement used in a program function, and used from the monitor window.

```
Function fineDisttest
    Fine 0.1   'Set precision to +/- 0.1 mm
    Go P1
    Go P2
Fend


> FineDist 0.1
>
> FineDist
0.1
```

FineStatus Function

Returns whether Fine or FineDist is used by an integer.

Syntax

FineStatus

Return Values

Returns whether Fine is used or FineDist is used by an integer.

0 = Fine is used
1 = FineDist is used

See Also

Fine, FineDist

FineStatus Function Example

```
    Print FineStatus
```

Fix Function

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
>
```

Flush Statement

Writes a file's buffer into the file.

Syntax

Flush *#fileNumber*

Parameters

#fileNumber	Integer value from 30 to 63 or expression

Description

Writes a file's buffer into the specified file.
Flush cannot be used if the file was opened with ROpen.

Flush Statement Example

```
Integer fileNum, i

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
For i = 0 To 100
   Print #fileNum, i
Next i
Flush #fileNum
Close #fileNum
```

FmtStr Statement

Formats a numeric expression or date/time expression.

Syntax

FmtStr *expFormat, strFormat, stringVar*

Parameters

expression	Numeric expression or date/time expression to be formatted. Specify date/time expression in “yyyy/mm/dd”.
strFormat	Format specification string.
stringVar	Output string variable.

Description

Returns the formatted string according to the *strFormat*.

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 “0” appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are “0” (on either side of the decimal) in the format expression, display leading or trailing “0”. If the number has more digits to the right of the decimal separator than there are “0” to the right of the decimal separator in the format expression, round the number to as many decimal places as there are “0”. If the number has more digits to the left of the decimal separator than there are “0” 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 “#” 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 “0” 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.
(,)	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.

Date/Time Expression Specifiers

Character	Description
(:)	Time separator. In some locals, other characters may be used. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in the formatted output depends on the Windows settings.
(/)	Date separator. In some locals, other characters may be used. The date separator separates day, month, and year when date values are formatted. The actual character used as the date separator in the formatted output depends on the Windows settings.
c	Display the date in "ddddd" and time in" ttttt", in this order. If the date serial number does not have a fraction, it only displays the date. If the timing information does not have the integer, it only displays the timing information.
d	Display the date with the day in the lead without "0". (1 to 31)
dd	Display the date with the day in the lead with "0". (01 to 31)
ddd	Displays the abbreviation of the day of the week. (Sun to Sat)
dddd	Displays the unabbreviated day of the week. (Sunday to Saturday)
ddddd	Displays the day, month, and year in the format of the short data display settings of the Windows. Default setting of the short data display format is m/d/yy.
dddddd	Displays the serial values of the date as day, month, and year in the long data display setting of the Windows. Default setting of the long data display is mmmm dd, yyyy.
w	Displays the day of the week with a number. (1: Sunday ~ 7: Saturday)
ww	Displays the number of weeks in a year with a number (1 to 54).
m	Display the month with the day in the lead without "0". (1 to 12) Even if this character is placed right after "h" or "hh", this does not display "minute". To display "minute", use "n" or "nn".
mm	Display the month with the day in the lead with "0".(01 to 12) Even if this character is placed right after "h" or "hh", this does not display "minute". To display "minute", use "n" or "nn".
mmm	Displays the abbreviated month name (Jan to Dec)
mmmm	Displays the unabbreviated month name (January to December) .
q	Displays the number of quarters in a year (1 to 4)
y	Displays the day of a year. (1 to 366)
yy	Displays the year in 2 digits.(00 to 99)
yyyy	Displays the year in 4 digits. (100 to 9999)
h	Displays the time in 24-hour clock without "0" at the beginning.(0 to 23)
hh	Displays the time in 24-hour clock with "0" at the beginning.(00 to 23)
n	Displays the minute without "0" at the beginning. (0 to 59)
nn	Displays the minute with "0" at the beginning.(00 to 59)
s	Displays the second without "0" at the beginning.(0 to 59)
ss	Displays the second with "0" at the beginning. (00 to 59)
t t t t t	Displays the time (hour, minute, second) with the time separator of Windows setting. If the "initial zero" option is used, the time before 10:00am/pm are displayed with "0" at the beginning. Default time format of the Windows is h:nn:ss.
AM/PM	Displays the time in 12-hour clock and displays morning and afternoon with AM/PM (uppercase).
am/pm	Displays the time in 12-hour clock and displays morning and afternoon with am/pm (lowercase).
A/P	Displays the time in 12-hour clock and displays morning and afternoon with A/P (uppercase).
a/p	Displays the time in 12-hour clock and displays morning and afternoon with a/p (lowercase).

AMPM	Displays the time in 12-hour clock. For the morning, displays AM with a string and for the afternoon, displays the PM with a string each with the Windows format setting. Both uppercases and lowercases can be used for AM/PM if the specified string matches the Windows setting. Default Windows setting is AM/PM.

Note

Mixture of numeric format specifiers and time/date specifiers

An error occurs if both numeric format specifier and time/date specifier are specified.

See Also

Left$, Right$, Str$

FmtStr Statement 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$, "mmddhhnn", f$
    f$ = f$ + ".dat"
    WOpen f$ as #30
    Print #30, "data"
    Close #30
Fend
```

FmtStr$ Function

Format a numeric expression.

Syntax

FmtStr$ (*expFormat, strFormat*)

Parameters

expFormat	Numeric expression or date/time expression to be formatted. Specify date/time expression in “yyyy/mm/dd”.
strFormat	Format specification string.

Return Values

A string containing the formatted expression.

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 “0” appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are “0” (on either side of the decimal) in the format expression, display leading or trailing “0”. If the number has more digits to the right of the decimal separator than there are “0” to the right of the decimal separator in the format expression, round the number to as many decimal places as there are “0”. If the number has more digits to the left of the decimal separator than there are “0” 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 “#” 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 “0” 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.
(,)	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.

Date/Time Expression Specifiers

Character	Description
(:)	Time separator. In some locals, other characters may be used. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in the formatted output depends on the Windows settings.
(/)	Date separator. In some locals, other characters may be used. The date separator separates day, month, and year when date values are formatted. The actual character used as the date separator in the formatted output depends on the Windows settings.
c	Display the date in "ddddd" and time in" ttttt", in this order. If the date serial number does not have a fraction, it only displays the date. If the timing information does not have the integer, it only displays the timing information.
d	Display the date with the day in the lead without "0". (1 to 31)
dd	Display the date with the day in the lead with "0". (01 to 31)
ddd	Displays the abbreviation of the day of the week. (Sun to Sat)
dddd	Displays the unabbreviated day of the week. (Sunday to Saturday)
ddddd	Displays the day, month, and year in the format of the short data display settings of the Windows. Default setting of the short data display format is m/d/yy.
dddddd	Displays the serial values of the date as day, month, and year in the long data display setting of the Windows. Default setting of the long data display is mmmm dd, yyyy.
w	Displays the day of the week with a number. (1: Sunday ~ 7: Saturday)
ww	Displays the number of weeks in a year with a number (1 to 54).
m	Display the month with the day in the lead without "0". (1 to 12) Even if this character is placed right after "h" or "hh", this does not display "minute". To display "minute", use "n" or "nn".
mm	Display the month with the day in the lead with "0".(01 to 12) Even if this character is placed right after "h" or "hh", this does not display "minute". To display "minute", use "n" or "nn".
mmm	Displays the abbreviated month name (Jan to Dec)
mmmm	Displays the unabbreviated month name (January to December) .
q	Displays the number of quarters in a year (1 to 4)
y	Displays the day of a year. (1 to 366)
yy	Displays the year in 2 digits.(00 to 99)
yyyy	Displays the year in 4 digits. (100 to 9999)
h	Displays the time in 24-hour clock without "0" at the beginning.(0 to 23)
hh	Displays the time in 24-hour clock with "0" at the beginning.(00 to 23)
n	Displays the minute without "0" at the beginning. (0 to 59)
nn	Displays the minute with "0" at the beginning.(00 to 59)
s	Displays the second without "0" at the beginning.(0 to 59)
ss	Displays the second with "0" at the beginning. (00 to 59)
t t t t t	Displays the time (hour, minute, second) with the time separator of Windows setting. If the "initial zero" option is used, the time before 10:00am/pm are displayed with "0" at the beginning. Default time format of the Windows is h:nn:ss.
AM/PM	Displays the time in 12-hour clock and displays morning and afternoon with AM/PM (uppercase).
am/pm	Displays the time in 12-hour clock and displays morning and afternoon with am/pm (lowercase).
A/P	Displays the time in 12-hour clock and displays morning and afternoon with A/P (uppercase).
a/p	Displays the time in 12-hour clock and displays morning and afternoon with a/p (lowercase).

AMPM	Displays the time in 12-hour clock. For the morning, displays AM with a string and for the afternoon, displays the PM with a string each with the Windows format setting. Both uppercases and lowercases can be used for AM/PM if the specified string matches the Windows setting. Default Windows setting is AM/PM.

Note

Mixture of numeric format specifiers and time/date specifiers

An error occurs if both numeric format specifier and time/date specifier are specified.

See Also

Left$, Right$, Str$

FmtStr$ Function Example

```
Function SendDateCode

    String d$, f$

    f$ = FmtStr$(10, "000.00")
    OpenCom #1
    Print #1, f$
    CloseCom #1
Fend
```

FolderExists Function

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. See ChDisk for the details.

Note

- This function is executable only with the PC disk.

Return Values

True if the folder exists, False if not.

See Also

FileExists, MkDir

FolderExists Function Example

```
If Not FolderExists("c:\TEST") Then
    MkDir "c:\TEST"
EndIf
```

For...Next Statement

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.

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.

When a variable exits the loop, the value is not a final value.

```
Function forsample
 Integer i
 For i = 0 To 3
 Next
 Print i   ' Displays 4
Fend
```

When you exit the loop from the nested structure without using Exit For

Error 2020 will occur when you repeatedly execute the program which exits the loop by the command other than the Exit For command (such as GoSub statement, Goto statement, and Call statement.) Be sure to use Exit For command to exit the loop.

Avoid endless execution of empty Loop Statements and similar to them, use them with the Wait command instead

Do not use empty For...Next statements and similar commands to avoid effect on the system. The Controllers are detecting endless loop tasks. If the controller determines that the system will be affected, it will stop the program with error 2556 (An excessive loop was detected). When performing operations that require a loop or waiting for I/O, execute a Wait command (Wait 0.1) and more within the loop to avoid occupying the CPU.

See Also

Do...Loop

For...Next Statement Example

```
Function fornext
    Integer counter
    For counter = 1 to 10
        Go Pctr
    Next counter

    For counter = 10 to 1 Step -1
        Go Pctr
    Next counter
Fend
```

Force_Calibrate Statement

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 (ATI Force Sensor) is active.
When using Force Sensor from EPSON, use each command of following manual.
EPSON RC+ 7.0 Option Force Guide 7.0 SPEL+ Language Reference

See Also

Force_Sensor

Force_Calibrate Statement Example

```
Force_Calibrate
```

Force_ClearTrigger Statement

Clears all trigger conditions for the current force sensor.

Syntax

Force_ClearTrigger

Description

Use Force_ClearTrigger to clear all conditions for the current force sensor's trigger.

Note

This command will only work if the Force Sensing option (ATI Force Sensor) is active.
When using Force Sensor from EPSON, use each command of following manual.
EPSON RC+ 7.0 Option Force Guide 7.0 SPEL+ Language Reference

See Also

Force_Sensor, Force_SetTrigger

Force_ClearTrigger Statement Example

```
Force_ClearTrigger
```

Force_GetForces Statement

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:

Index	Axis	Constant
1	X Force	FORCE_XFORCE
2	Y Force	FORCE_YFORCE
3	Z Force	FORCE_ZFORCE
4	X Torque	FORCE_XTORQUE
5	Y Torque	FORCE_YTORQUE
6	Z Torque	FORCE_ZTORQUE

Description

Use Force_GetForces to read all force and torque values at once.

Note

This command will only work if the Force Sensing option (ATI Force Sensor) is active.
When using Force Sensor from EPSON, use each command of following manual.
EPSON RC+ 7.0 Option Force Guide 7.0 SPEL+ Language Reference

See Also

Force_GetForce Function

Force_GetForces Statement Example

```
Real fValues(6)
Force_GetForces fValues()
```

Force_GetForce Function

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 (ATI Force Sensor) is active.
When using Force Sensor from EPSON, use each command of following manual.
EPSON RC+ 7.0 Option Force Guide 7.0 SPEL+ Language Reference

See Also

Force_GetForces

Force_GetForce Function Example

```
Print Force_GetForce(1)
```

Force_Sensor Statement

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 (ATI Force Sensor) is active.
When using Force Sensor from EPSON, use each command of following manual.
EPSON RC+ 7.0 Option Force Guide 7.0 SPEL+ Language Reference

See Also

Force_Sensor Function

Force_Sensor Statement Example

```
        Force_Sensor 1
```

Force_Sensor Function

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 (ATI Force Sensor) is active.
When using Force Sensor from EPSON, use each command of following manual.
EPSON RC+ 7.0 Option Force Guide 7.0 SPEL+ Language Reference

See Also

Force_Sensor

Force_Sensor Function Example

```
        var = Force_Sensor
```

Force_SetTrigger Statement

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

Comparison	Constant	Value
Less than or equal	FORCE_LESS	0
Greater than or equal	FORCE_GREATER	1

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 (ATI Force Sensor) is active.
When using Force Sensor from EPSON, use each command of following manual.
EPSON RC+ 7.0 Option Force Guide 7.0 SPEL+ Language Reference

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
```

FreeFile Function

Returns / reserves a file number that is currently not being used.

Syntax

FreeFile

Return Values

Integer between 30 and 63.

See Also

AOpen, BOpen, ROpen, UOpen, WOpen, Close

FreeFile Function Example

```
Integer fileNum, i, j

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

fileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print "data = ", j
Next i
Close #fileNum
```

Function...Fend Statement

A function is a group of program statements which includes a Function statement as the first statement and an Fend statement as the last statement.

Syntax

Function *funcName* [(*argList*)] [**As** *type(function)*]
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 64 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(argument)*

ByRef	Optional. Specify ByRef when you refer to the variable to be seen by the calling function. In this case, the argument change in a function can be reflected to the variable of the calling side.
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. If you use an array variable as argument, you should specify ByRef and add empty parentheses "()" representing the array after the variable name.
As *type (argument)*	Required. You must declare the type of argument.

As *type (function)*	Use this parameter if you want to obtain return values. You must declare the type of return values.

Return Values

Value whose data type is specified with the **As** clause at the end of the function declaration (**As** *type(function)*).

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.

If you want to use the return value, assign the value to the variable name which has the same name as the function and then terminate the function.

See Also

Call, Fend, Halt, Quit, Return, Xqt

Function...Fend Statement Example

<Example 1>

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
  Do
    On 1
    On 2
    Off 1
    Off 2
  Loop
Fend

Function task3
  Do
    On 10
    Wait 1
    Off 10
  Loop
Fend
```

<Example 2>

In the following example, the pressure control sequence for peripherals is supplied as an argument and the result sent to the external device is displayed as a return value.

```
Function main
  Integer iResult
   Real Sequence1(200)
  .
  .
   iResult = PressureControl(ByRef Sequence1())     'Argument is array
  .
  Print "Result:", iResult
  .
Fend

Function PressureControl(ByRef Array1() As Real) As Integer
  .
   (Control pressure for peripherals according to Array1)
  .
   PressureControl = 3               'Return value
  .
  .
Fend
```

GetCurrentUser$ Function

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 active.

See Also

LogIn Statement

GetCurrentUser$ Function Example

```
String currUser$

currUser$ = GetCurrentUser$
```

GetRobotInsideBox Function

Returns a robot which is in the approach check area.

Syntax

GetRobotInsideBox(*AreaNum*)

Parameters

AreaNum	Integer value (1 to 15) representing the approach check area you want to return the status for.

Return Values

Return the robot that is in the approach check area specified with *AreaNum* in bit.

Bit 0 : Robot 1 ……… Bit 15 : Robot 16

If the robot doesn't configure the approach check area, bit is always 0.

For example, Robot 1, Robot 3 are in the approach check area, bit 0, bit 2 will be On and 5 will be returned.

See Also

Box, InsideBox

GetRobotInsideBox Function Example

The following program uses the GetRobotInsideBox function.

Wait for the status that no robots are in the approach check area.

```
Function WaitNoBox

   Wait GetRobotInsideBox(1) = 0
```

Wait for the status that Robot 2 is only one in the approach check area.

```
Function WaitInBoxRobot2

   Wait GetRobotInsideBox(1) = &H2
```

The following program uses the GetRobotInsideBox function in the parallel processing of the motion command. When a robot is in the specific approach check area while it is running, it turns ON the I/O. One robot is connected to the controller in this case.

```
Function Main
   Motor On
   Power High
   Speed 30; Accel 30, 30

   Go P1 !D0; Wait GetRobotInsideBox(1) = 1; On 1!

Fend
```

Note

D0 must be described.

GetRobotInsidePlane Function

Returns a robot which is in the approach check plane.

Syntax

GetRobotInsidePlane (*PlaneNum*)

Parameters

PlaneNum	Integer value (1 to 15) representing the approach check plane you want to return the status for.

Return Values

Returns the number of the robot that is in the approach check plane specified with *PlaneNum* in bit.

Bit 0 : Robot 1 ……… Bit 15 : Robot 16

If the robot doesn't configure the approach check plane, it always returns bit 0.

For example, Robot 1, Robot 3 are in the approach check plane, bit 0, bit 2 will be On and 5 will be returned.

See Also

InsidePlane, Plane

GetRobotInsidePlane Function Example

The following program uses the GetRobotInsidePlane function.

Wait for the status that no robots are in the approach check plane.

```
Function WaitNoPlane

   Wait GetRobotInsidePlane(1) = 0
```

Wait for the status Robot 2 is only one in the approach check plane.

```
Function WaitInPlaneRobot2

   Wait GetRobotInsidePlane(1) = &H2
```

The following program uses the GetRobotInsidePlane function in the parallel processing of the motion command. When a robot is in the specific approach check plane while it is running, it turns ON the I/O. One robot is connected to the controller in this case.

```
Function Main
   Motor On
   Power High
   Speed 30; Accel 30, 30

   Go P1 !D0; Wait GetRobotInsidePlane(1) = 1; On 1!

Fend
```

Note

D0 must be described.

Global Statement

Declares variables with the global scope. Global variables can be accessed from anywhere.

Syntax

Global [**Preserve**] *dataType varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**] , ...]

Parameters

Preserve	If Preserve is specified, then the variable retains its values. The values are cleared by project changes. If Preserve is omitted, the variable doesn't retain its values.
dataType	Data type including Boolean, Byte, Double, Int32, Integer, Long, Real, Short, String, UByte, UInt32, or UShort.
varName	Variable name. Names may be up to 32 characters in length.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 to the upper bound value. The total available number of array elements for global variables is 10000 for strings and 100000 for all other types. The total available number of array elements for global preserve variables is 400 for strings and 4000 for all other types. To calculate the total elements used in an array, use the following formula. (If a dimension is not used, substitute 0 for the ubound value.) total elements = (ubound1 + 1) * (ubound2 + 1) * (ubound3 + 1)

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.

When declared in Preserve option, the variable retains the value at turning off the controller.

Global Preserve variables can be used with the RC+ Connectivity 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, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

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 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

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**] [LJM [*orientationFlag*]] [searchExpr] [!...!] [*SYNC*]

Parameters

destination	The target destination of the motion using a point expression.
CP	Optional. Specifies continuous path motion.
LJM	Optional. Convert the target destination using LJM function.
orientationFlag	Optional. Specifies a parameter that selects an orientation flag for LJM function.

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.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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 path is not predictable because each joint interpolates between the current point and the target point. Be careful of the interference with peripherals.

The Speed instruction determines the arm speed for motion initiated by the Go instruction. The Accel instruction defines the acceleration.

With CP parameter, the arm can accelerate for the next motion command while the arm starts decelerating to a stop. In this case, the arm is not positioned at the target point.

With LJM parameter, the arm moves to the point into where the target point is converted using LJM function, with the current point as reference point.

Go LJM (P1, Here, 1) can be Go P1 LJM 1.
At this point, the original point data P1 does not change.
LJM parameter is available for the 6-axis (including N series) and RS series robots.
When using *orientationFlag* with the default value, it can be omitted.

```
Go P1 LJM
```

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 whereas 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 Error

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, P#= (Point Assignment), Pulse, Speed, Sw, Till

Go Example

The example shown below shows a simple point to point move between points P0 and P10. 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 Go, 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."
    EndIf
  Fend
```

Some syntax examples from the command window are shown below:

```
>Go Here +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) = Off 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 Statement

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 { *label* }

{ *label*:}
statements
Return

Parameters

label	When the user specifies a label, the program execution will jump to the line on which this label resides. The label can be up to 32 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 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 label then an Error 3108 will be issued.

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 2383 will occur. A standalone 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

The GoTo instruction causes program control to branch unconditionally to a designated statement label.

Syntax

GoTo { *label* }

Parameters

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 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.

Note

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

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 16 for normal tasks and from 257 to 261 for trap tasks.

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.

Halt also stops the task when the specified task is NoPause task, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), trap tasks, or the background tasks.
However, stopping these tasks needs enough consideration. Normally, Halt is not recommended for the special task.

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 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

Sets the hand (arm) orientation of a point.

Syntax

(1) **Hand** *point* [, *Lefty* | *Righty*]
(2) **Hand**

Parameters

point	**P***number* or **P***(expr)* or point label.
Lefty | *Righty*	Hand (arm) orientation.

Return Values

When both parameters are omitted, the hand (arm) orientation is displayed for the current robot position.
If *Lefty* | *Righty* is omitted, the hand (arm) orientation for the specified point is displayed.

Note

Hand command is not a command to control hand (end-effecter).

Command	Function
Hand (This command)	Specify the arm of manipulator is righty or lefty.
Hand_On, Hand_Off, Hand_On Function, Hand_Off Function, Hand_TW Function, Hand_Def Function, Hand_Type Function, Hand_Label$ Function, Hand_Number Function	Control hand (end-effector) installed to the end of the manipulator.

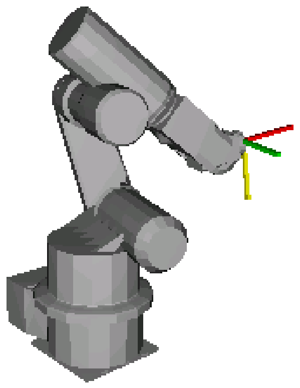
Please be careful not to confuse it.
For details of Hand control commands, refer to the Hand Function Manual.

See Also

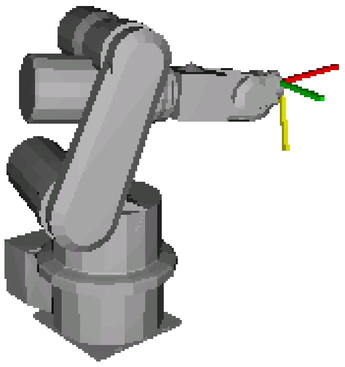
Elbow, Hand Function, J4Flag, J6Flag, Wrist, J1Flag, J2Flag

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

Returns the hand (arm) orientation of a point.

Syntax

Hand [**(***point***)**]

Parameters

point	Optional. Point expression. If *point* is omitted, then the hand (arm) orientation of the current robot position is returned.

Return Values

1	Righty (/R)
2	Lefty (/L)

Note

Hand command is not a command to control hand (end-effecter).

Command	Function
Hand (This command)	Specify the arm of manipulator is righty or lefty.
Hand_On, Hand_Off, Hand_On Function, Hand_Off Function, Hand_TW Function, Hand_Def Function, Hand_Type Function, Hand_Label$ Function, Hand_Number Function	Control hand (end-effector) installed to the end of the manipulator.

Please be careful not to confuse it.
For details of Hand control commands, refer to the Hand Function Manual.

See Also

Elbow, Wrist, J4Flag, J6Flag, J1Flag, J2Flag

Hand Function Example

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

HealthCalcPeriod Statement

Sets and displays a period calculating “remaining months” of parts consumption control information.

Syntax

(1) HealthCalcPeriod Period
(2) HealthCalcPeriod

Parameters

Period	Specify a period to calculate in integer (1~7).(Unit: day) Default value is “7”.

Return Values

Displays the current setting value of HealthCalcPeriod when omitting the parameter.

Description

Remaining months of parts consumption control information is automatically calculated based on the past operating condition. HealthCalcPeriod command sets and displays an operating period for this calculation.

When setting the period longer, “remaining month” that the influence of variations due to fluctuations within the period is suppressed.

Therefore, even when the motion or speed is changed, it takes a time until it is reflected to the “remaining month”.

Setting value of HealthCalcPeriod is applied to all robot, joint, and part controlled by executed controller.

Note

About the Controllers to use

It cannot be used with T/VT series. It is always using 1 (day) for T/VT series.

Setting Period

Period which is set in HealthCalcPeriod command is running hour of the controller.
The number of period is increased by 1 when 24 hours are accumulated.

Calculation of “remaining months” and “consumption rate” when clearing.

Calculates remaining months every day in spite of the setting value of HealthCalcPeriod until the setting period is reached for the first time after clearing the remaining months and parts consumption rate at the “parts consumption control information” in EPSON RC+ or by executing HealthCtrlReset or HealthRBReset. During this period, the number of remaining months will vary widely.

See Also

HealthCalcPeriod Function, HealthCtrlInfo, HealthRBInfo, HealthCtrlReset, HealthRBReset

HealthCalcPeriod on Functional Example

```
> HealthCalcPeriod 3
> HealthCalcPeriod
 3
```

HealthCalcPeriod Function

Returns "remaining months" calculating period of the parts consumption control information which is currently set.

Syntax

HealthCalcPeriod

Return Values

Returns calculating period in integer. (Unit: day)

Note

About the Controllers to use

It cannot be used with T/VT series.

See Also

HealthCalcPeriod

HealthCalcPeriod on Functional Example

Example to display the calculating period.

```
    Print "period is", HealthCalcPeriod
```

HealthCtrlAlarmOn Function

Returns the status of the parts consumption alarm for the specified Controller parts.

Syntax

HealthCtrlAlarmOn(*partType***)**

Parameters

partType Integer expression (1) or the following constant representing the parts you want to obtain the alarm status.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

Return Values

True if the parts consumption alarm is occurring for the specified parts, otherwise False.

The parts consumption alarm occurs when the parts consumption rate obtained by HealthRateCtrlInfo exceeds 100%.

See Also

HealthCtrlInfo, HealthRateCtrlInfo

HealthCtrlAlarmOn Function Example

The example below determines if the parts consumption alarm is occurring for the Controller batteries.

```
Function PrintAlarm
   If HealthCtrlAlarmOn(HEALTH_CONTROLLER_TYPE_BATTERY) = True Then
      Print "Controller Battery NG"
   Else
      Print "Controller Battery OK"
   EndIf
Fend
```

HealthCtrlInfo Statement

Displays the remaining months before the recommended replacement time for the specified Controller parts.

Syntax

HealthCtrlInfo *partType*

Parameters

partType Integer expression (1) or the following constant representing the parts you want to obtain the remaining months before the recommended replacement time.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

Description

Displays the remaining months before the recommended replacement time for the specified Controller parts.

The remaining months are calculated based on the parts consumption rate from the past usage and the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller.

Notes

Since the remaining months are calculated based on the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller, they cannot be calculated properly in the following cases:

- If the command is executed when the operating time is less than every operation of a period which is set in HealthCalcPeriod
- If the command is executed after the long-term operation stop period of the robot.
- If the command is executed after the parts consumption alarm is reset due to the parts replacement.
- If the time and date on the Controller is changed.

In above cases, execute the command after operating the Controller more than twice of setting period in HealthCalcPeriod to display the accurate value.

See Also

HealthCtrlAlarmOn, HealthRateCtrlInfo

HealthCtrlInfo Statement Example

The example below displays the remaining months before the recommended replacement time for the Controller batteries.

```
> HealthCtrlInfo HEALTH_CONTROLLER_TYPE_BATTERY
BATTERY             240.000
>
```

HealthCtrlInfo Function

Returns the remaining months before the recommended replacement time for the specified Controller parts.

Syntax

HealthCtrlInfo(*partType*)

Parameters

partType Integer expression (1) or the following constant representing the parts you want to obtain the remaining months before the recommended replacement time.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

Return Values

Real number representing the remaining months before the recommended replacement time. (Unit: month)

Description

The remaining months are calculated based on the parts consumption rate from the past usage and the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller.

Notes

Since the remaining months are calculated based on the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller, they cannot be calculated properly in the following cases:

- If the command is executed when the operating time is less than every operation of a period which is set in HealthCalcPeriod.
- If the command is executed after the long-term operation stop period of the robot.
- If the command is executed after the parts consumption alarm is reset due to the parts replacement.
- If the time and date on the Controller is changed.

In above cases, execute the command after operating the Controller more than twice of setting period in HealthCalcPeriod to display the accurate value.

See Also

HealthCtrlAlarmOn, HealthRateCtrlInfo

HealthCtrlInfo Function Example

The example below outputs the alarm when the recommended replacement time is in less than one month.

```
Function AlarmCheck
   Real month

   month = HealthCtrlInfo(HEALTH_CONTROLLER_TYPE_BATTERY)
   If month < 1 Then
      Print "Alarm ON"
   EndIf
Fend
```

HealthCtrlRateOffset Statement

Sets the offset for the consumption rate of the specified parts.

Syntax

HealthCtrlRateOffset *partType*, o*ffset*

Parameters

partType Integer expression (1) or the following constant representing the controller related parts.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

offset Integer expression that specifies the offset value added to the consumption rate. (Unit: %)

Description

Sets the offset for the consumption rate of the specified parts.

See Also

HealthRBAlarmOn, HealthRateRBInfo, HealthRBInfo

HealthCtrlRateOffset Statement Example

The following is the example to add 10% to the parts consumption rate of the Controller batteries.

```
> HealthCtrlRateOffset HEALTH_CONTROLLER_TYPE_BATTERY,10
>
```

HealthCtrlReset Statement

Clears the remaining months before the recommended replacement time and the consumption rate for the specified parts.

Syntax

HealthCtrlReset *partType*

Parameters

partType Integer expression (1) or the following constant representing the controller related parts.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

Description

Clears the remaining months before the recommended replacement time and the consumption rate for the specified parts.
The warnings are also canceled.

See Also

HealthCtrlAlarmOn, HealthRateCtrlInfo, HealthCtrlInfo

HealthCtrlReset Statement Example

```
> HealthCtrlReset HEALTH_CONTROLLER_TYPE_BATTERY
>
```

HealthCtrlWarningEnable Statement

Enable or disable the parts consumption alarm notification of specified part related to the Controller.

Syntax

HealthCtrlWarningEnable *partType* [, On/Off]

Parameters

partType Integer expression or the following constant representing the controller.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

On/Off On: Enable the parts consumption alarm notification.

Off: Disable the parts consumption alarm notification

Return Values

If On/Off parameters are omitted, the current On/Off settings are displayed.

Description

When the parts consumption alarm of the specified part occurs, set whether to notify the parts consumption alarm.

Notes

If the parts consumption alarm of the specified part is disabled, the parts consumption alarm will not be notified when the recommended replacement time is passed. Be careful to set when using this command.

See Also

HealthCtrlAlarmOn

HealthCtrlWarningEnable Example

Example to disable the parts consumption alarm of batteries of the controller.

```
> HealthCtrlWarningEnable HEALTH_CONTROLLER_TYPE_BATTERY, Off
```

Example to display the parts consumption alarm settings of batteries of the controller.

```
> HealthCtrlWarningEnable HEALTH_CONTROLLER_TYPE_BATTERY
BATTERY  Off
>
```

HealthCtrlWarningEnable Function

Returns the setting status of the parts consumption alarm notification of specified part related to the Controller.

Syntax

HealthCtrlWarningEnable(*partType*)

Parameters

partType Integer expression or the following constant representing the controller.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

Return Values

Returns the setting values of the parts consumption alarm in integer.
1: On
0: Off

See Also

HealthCtrlAlarmOn

HealthCtrlWarningEnable Function Example

Example to display the parts consumption alarm of batteries of the controller.

```
Print HealthCtrlWarningEnable(HEALTH_CONTROLLER_TYPE_BATTERY )
```

HealthRateCtrlInfo Function

Returns the consumption rate of the specified Controller parts.

Syntax

HealthRateCtrlInfo(*partType*)

Parameters

partType Integer expression (1) or the following constant representing the parts you want to obtain the remaining months before the recommended replacement time.

Constant	Value	Mode
HEALTH_CONTROLLER_TYPE_BATTERY	1	Specifies the batteries.

Return Values

Real number representing the current parts consumption rate, when regarding the recommended replacement time as 100%. (Unit: %)

Description

This command calculates the parts consumption rate based on the actual operating condition.

Notes

The recommended replacement time is the recommended time to replace the parts calculated based on statistics.
Replacement may be required before the consumption rate reaches 100%.
In addition, the parts will not become immediately unusable when the consumption rate reaches 100%.
However, it is recommended to replace the parts soon as the possibility of breakage increases after the consumption rate reaches 100%.

See Also

HealthCtrlAlarmOn, HealthCtrlInfo,

HealthRateCtrlInfo Function Example

The example below outputs the alarm when the consumption rate for the Controller batteries reaches 90%.

```
Function AlarmCheck
   Real HealthRate

   HealthRate = HealthRateCtrlInfo (HEALTH_CONTROLLER_TYPE_BATTERY)
   If HealthRate > 90 Then
      Print "Alarm ON"
   EndIf
Fend
```

HealthRateRBInfo Function

Returns the consumption rate for the specified robot parts.

Syntax

HealthRateRBInfo(*robotNumber*, *partType*, *jointNumber*)

Parameters

robotNumber Integer expression (1-16) representing the robot number you want to obtain the parts consumption rate.

partType Integer expression (1-6) or the following constants representing the parts you want to obtain the consumption rate.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber Integer expression (1-9) representing the joint you want to obtain the parts consumption rate.
This command is unavailable for the additional axes.

Return Values

Real number representing the current parts consumption rate, when regarding the recommended replacement time as 100%. (Unit: %)

Returns "-1" when the robot does not have the specified parts.

Description

This command calculates the parts consumption rate based on the actual operating condition.

Notes

The recommended replacement time is the recommended time to replace the parts calculated based on statistics.
Replacement may be required before the consumption rate reaches 100%.
In addition, the parts will not become immediately unusable when the consumption rate reaches 100%.
However, it is recommended to replace the parts soon as the possibility of breakage increases after the consumption rate reaches 100%.

See Also

HealthRBAlarmOn, HealthRBInfo

HealthRateRBInfo Function Example

The example below outputs the alarm when the consumption rate of the Joint #3 reduction gear unit on the robot 1 reaches 90%.

```
Function AlarmCheck
  Real HealthRate

  HealthRate = HealthRateRBInfo(1,HEALTH_ROBOT_TYPE_GEAR,3)
  If HealthRate > 90 Then
    Print "Alarm ON"
  EndIf
Fend
```

HealthRBAlarmOn Function

Returns the status of the parts consumption alarm for the specified robot parts.

Syntax

HealthRBAlarmOn(*robotNumber*, *partType*, *jointNumber*)

Parameters

robotNumber Integer expression (1-16) representing the robot number you want to obtain the alarm status.

partType Integer expression (1-6) or the following constants representing the parts you want to obtain the alarm status.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber Integer expression (1-9) representing the joint you want to obtain the alarm status. When the batteries are selected *for partType*, the same value will be returned when any joint is specified because the batteries are common to all joints. This command is unavailable for the additional axes.

Return Values

True if the parts consumption alarm is occurring for the specified parts, otherwise False.

The parts consumption alarm occurs when the parts consumption rate obtained by HealthRateRBInfo exceeds 100%.

Returns "-1" when the robot does not have the specified parts.

See Also

HealthRBInfo, HealthRateRBInfo

HealthRBAlarmOn Function Example

The example below determines if the parts consumption alarm is occurring for the grease on the Joint #3 of the robot 1.

```
Function PrintAlarm4
  If HealthRBAlarmOn(1, HEALTH_ROBOT_TYPE_GREASE, 3) = True Then
    Print "Robot1 Joint3 Grease NG"
  Else
    Print "Robot1 Joint3 Grease OK"
  EndIf
Fend
```

HealthRBAnalysis Statement

Simulates and displays the usable months for the specified parts in a particular robot operation cycle.

Syntax

HealthRBAnalysis *robotNumber*, *partType*[, *jointNumber*]

Parameters

robotNumber	Integer expression (1-16) representing the robot number.
partType	Integer expression or the following constants representing the robot parts.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_ALL	0	Specifies all parts.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber	Integer expression (1-6) representing the joint. If the joint number is not specified, returns values for all the joints. This command is unavailable for the additional axes.

Description

Simulates and displays the usable months for the specified parts in a particular robot operation cycle. This command calculates and displays how many months the parts can be used if they are new and used for 24 hours a day. The past usage is not considered.

Returns "-1" when the specified parts are not installed on the specified joint.

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode (including the virtual controller).

See Also

HealthRBStart, HealthRBStop

HealthRBAnalysis Statement Example

The example below displays the usable months for all parts of all joints on SCARA robot.

```
> HealthRBAnalysis 1, HEALTH_ROBOT_TYPE_ALL
BELT                -1.000,   -1.000,   38.689,   95.226
GREASE              -1.000,   -1.000,   21.130,   -1.000
MOTOR              240.000,  240.000,  240.000,  240.000
GEAR               240.000,  224.357,   -1.000,   -1.000
BALL_SCREW_SPLINE   -1.000,   -1.000,  240.000,   -1.000
>
```

The example below displays the usable months for the reduction gear units of all joints on SCARA robot.

```
> HealthRBAnalysis 1, HEALTH_ROBOT_TYPE_GEAR
GEAR               240.000,  224.357,   -1.000,   -1.000
>
```

The example below displays the usable months for the Joint #2 motor on 6-axis robot.

```
> HealthRBAnalysis 1, HEALTH_ROBOT_TYPE_MOTOR, 2
MOTOR              224.357
>
```

HealthRBAnalysis Function

Returns the usable months for the specified parts in a particular robot operation cycle.

Syntax

HealthRBAnalysis(*robotNumber*, *partType*, *jointNumber*)

Parameters

robotNumber	Integer expression (1-16) representing the robot number.
partType	Integer expression (2-6) or the following constants representing the robot parts.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber	Integer expression (1-6) representing the joint. This command is unavailable for the additional axes.

Return Values

Real number representing the usable months.

Returns "-1" when the specified parts are not installed on the specified joint.

Description

Simulates the usable months for the specified parts in a particular robot operation cycle. This command calculates how many months the parts can be used if they are new and used for 24 hours a day. The past usage is not considered.

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode.

See Also

HealthRBStart, HealthRBStop

HealthRBAnalysis Function Example

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

HealthRBDistance Statement

Displays the driving (rotation) amount of the motor of the specified joint.

Syntax

HealthRBDistance [*robotNumber*] **[,***jointNumber*]

Parameters

robotNumber	Optional. Integer expression (1-16) representing the robot number. If omitted, the current robot number will be used.
jointNumber	Integer expression (1-6) representing the joint. If the joint number is not specified, returns values for all the joints. This command is unavailable for the additional axes.

Description

Calculates and displays the driving (rotation) amount of the motor of the specified joint in robot operation from HealthRBStart to HealthRBStop. The past usage is not considered.

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode.

See Also

HealthRBStart, HealthRBStop

HealthRBDistance Statement Example

The example below displays the driving amount of the Joint #1 of SCARA robot.

```
> HealthRBDistance 1, 1
 1.000
>
```

HealthRBDistance Function

Returns the driving (rotation) amount of the motor of the specified joint.

Syntax

HealthRBDistance([*robotNumber*,] *jointNumber*)

Parameters

robotNumber	Optional. Integer expression (1-16) representing the robot number. If omitted, the current robot number will be used.
jointNumber	Integer expression (1-6) representing the joint. This command is unavailable for the additional axes.

Return Values

Real number representing the driving amount.

Description

Returns the driving (rotation) amount of the motor of the specified joint in robot operation from HealthRBStart to HealthRBStop. The past usage is not considered.

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode.

See Also

HealthRBStart, HealthRBStop

HealthRBDistance Function Example

```
Function RobotPartAnalysis
   Real healthDistance

   Robot 1

   HealthRBStart 1
   Motor On
   Go P0
   Go P1
   Motor Off
   HealthRBStop 1

   healthDistance = HealthRBDistance(1,1)
   Print "Distance =", Str$(healthDistance)
Fend
```

HealthRBInfo Statement

Displays the remaining months before the recommended replacement time for the specified robot parts.

Syntax

HealthRBInfo *robotNumber*, *partType*[, *jointNumber*]

Parameters

robotNumber Integer expression (1-16) representing the robot number you want to obtain the remaining months before the recommended replacement time.

partType Integer expression (0-6) or the following constants representing the parts you want to obtain the remaining months before the recommended replacement time.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_ALL	0	Specifies all parts.
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber Integer expression (1-9) representing the joint you want to obtain the remaining months before the recommended replacement time. When the batteries are selected for *partType*, the same value will be returned when any joint is specified because the batteries are common to all joints. If the joint number is not specified, returns values for all the joints. This command is unavailable for the additional axes.

Description

Displays the remaining months before the recommended replacement time for the specified robot parts.

The remaining months are calculated based on the parts consumption rate from the past usage and the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller.

Returns "-1" when the robot joint does not have the specified parts.

Notes

Since the remaining months are calculated based on the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller, they cannot be calculated properly in the following cases:

- If this command is executed when the operating time is less than every operation of a period which is set in HealthCalcPeriod.
- If this command is executed after the long-term operation stop period of the robot.
- If this command is executed after the parts consumption alarm is reset after the parts replacement.
- If the time and date on the Controller is changed.

In above cases, execute the command after operating the Controller more than twice of setting period in HealthCalcPeriod to display the accurate value.

See Also

HealthRBAlarmOn, HealthRateRBInfo

HealthRBInfo Statement Example

The example below displays the remaining months for all parts of all joints on the robot 1.

```
> HealthRBInfo 1, HEALTH_ROBOT_TYPE_ALL
BATTERY            240.000
BELT                -1.000,   -1.000,   38.689,   95.226
GREASE              -1.000,   -1.000,   21.130,   -1.000
MOTOR              240.000,  240.000,  240.000,  240.000
GEAR               240.000,  224.357,   -1.000,   -1.000
BALL_SCREW_SPLINE   -1.000,   -1.000,  240.000,   -1.000
>
```

The example below displays the remaining months for the reduction gear units of all joints on the robot 1.

```
> HealthRBInfo 1, HEALTH_ROBOT_TYPE_GEAR
GEAR               240.000,  224.357,   -1.000,   -1.000
>
```

The example below displays the remaining months for the Joint #2 motor of the robot 1.

```
> HealthRBInfo 1, HEALTH_ROBOT_TYPE_MOTOR, 2
MOTOR              224.357
>
```

HealthRBInfo Function

Returns the remaining months before the recommended replacement time for the specified robot parts.

Syntax

HealthRBInfo(*robotNumber*, *partType*, *jointNumber*)

Parameters

robotNumber Integer expression (1-16) representing the robot number you want to obtain the remaining months before the recommended replacement time.

partType Integer expression (1-6) or the following constants representing the parts you want to obtain the remaining months before the recommended replacement time.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber Integer expression (1-9) representing the joint you want to obtain the remaining months before the recommended replacement time. When the batteries are selected for *partType*, the same value will be returned when any joint is specified because the batteries are common to all joints. This command is unavailable for the additional axes.

Return Values

Real number representing the remaining months before the recommended replacement time. (Unit: month)
Returns "-1" when the robot does not have the specified parts.

Description

The remaining months are calculated based on the parts consumption rate from the past usage and the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller.

Notes

Since the remaining months are calculated based on the amount of change in the consumption rate obtained every operation of a period which is set in HealthCalcPeriod of the Controller, they cannot be calculated properly in the following cases:

- If this command is executed when the operating time is less than every operation of a period which is set in HealthCalcPeriod.
- If this command is executed after the long-term operation stop period of the robot.
- If this command is executed after the parts consumption alarm is reset after the parts replacement.
- If the time and date on the Controller is changed.

In above cases, execute the command after operating the Controller more than twice of setting period in HealthCalcPeriod to display the accurate value.

See Also

HealthRBAlarmOn, HealthRateRBInfo

HealthRBInfo Function Example

The example below outputs the alarm when the recommended replacement time for the Joint #3 ball screw spline on the robot 1 is in less than one month.

```
   Function AlarmCheck
   Real month

   month = HealthRBInfo(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
   If month < 1 Then
      Print "Alarm ON"
   EndIf
Fend
```

HealthRBRateOffset Statement

Sets the offset for the consumption rate of the specified parts.

Syntax

HealthRBRateOffset *robotNumber*, *partType*, *jointNumber*, *offset*

Parameters

robotNumber Integer expression (1-16) representing the robot number.

partType Integer expression (1-6) or the following constants representing the robot part.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber Integer expression (1-6) representing the joint. When the batteries are selected for partType, the offset will be set when any joint is specified because the batteries are common to all joints. This command is unavailable for the additional axes.

offset Integer expression that specifies the offset value added to the consumption rate. (Unit: %)

Description

Sets the offset for the consumption rate of the specified parts and joints.

See Also

HealthRBAlarmOn, HealthRateRBInfo, HealthRBInfo

HealthRBRateOffset Example

The example below adds 10% to the consumption rate of the Joint #1 reduction gear unit on the robot 1.

```
> HealthRBRateOffset 1,HEALTH_ROBOT_TYPE_GEAR,1,10
>
```

HealthRBReset Statement

Clears the remaining months before the recommended replacement time and the consumption rate for the specified parts.

Syntax

HealthRBReset *robotNumber*, *partType*, *jointNumber*

Parameters

robotNumber Integer expression (1-16) representing the robot number.

partType Integer expression (1-6) or the following constants representing the robot parts.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

jointNumber Integer expression (1-6) representing the joint you want to obtain the remaining months before the recommended replacement time. When the batteries are selected for *partType*, the remaining months will be cleared when any joint is specified because the batteries are common to all joints. This command is unavailable for the additional axes.

Description

Clears the remaining months before the recommended replacement time and the consumption rate for the specified parts and joints.

The warnings are also canceled.

See Also

HealthRBAlarmOn, HealthRateRBInfo, HealthRBInfo

HealthRBReset Statement Example

```
> HealthRBReset 1,HEALTH_ROBOT_TYPE_GEAR,1
>
```

HealthRBSpeed Statement

Displays the average speed of the specified joint.

Syntax

HealthRBSpeed [*robotNumber*] [, *jointNumber*]

Parameters

robotNumber	Optional. Integer expression (1-16) representing the robot number. If omitted, the current robot number will be used.
jointNumber	Integer expression (1-6) representing the joint. If the joint number is not specified, returns values for all the joints. This command is unavailable for the additional axes.

Description

Returns the average of the absolute values for speed of the specified joint in robot operation from HealthRBStart to HealthRBStop. The result is a real number from 0 to 1.
The maximum average speed is "1".
The value is "0" when the average value is 0.001 or less.

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode.

See Also

HealthRBStart, HealthRBStop, AveSpeed

HealthRBSpeed Statement Example

The example below displays the speed of the Joint #1 of SCARA robot.

```
> HealthRBSpeed 1, 1
 0.100
>
```

HealthRBSpeed Function

Returns the average of the absolute values for speed of the specified joint.

Syntax

HealthRBSpeed ([*robotNumber*,] *jointNumber*)

Parameters

robotNumber	Optional. Integer expression (1-16) representing the robot number. If omitted, the current robot number will be used.
jointNumber	Integer expression (1-6) representing the joint. This command is unavailable for the additional axes.

Return Values

The result is a real number from 0 to 1.

Description

Returns the average of the absolute values for speed of the specified joint in robot operation from HealthRBStart to HealthRBStop. The result is a real number from 0 to 1. The maximum average speed is "1".

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode (including the virtual controller).

See Also

HealthRBStart, HealthRBStop, AveSpeed

HealthRBSpeed Function Example

```
Function RobotPartAnalysis
   Real healthSpeed

   Robot 1

   HealthRBStart 1
   Motor On
   Go P0
   Go P1
   Motor Off
   HealthRBStop 1

   healthSpeed = HealthRBSpeed(1,1)
   Print "AveSpeed =", Str$(healthSpeed)
Fend
```

HealthRBStart Statement

Starts calculation of the usable months and elements for the parts in a particular robot operation cycle.

Syntax

HealthRBStart *robotNumber*

Parameters

robotNumber	Integer expression (1-16) representing the robot number.

Description

Starts calculation of the usable months and elements (torque, speed, and driving amount) for the parts on the specified robot in a particular robot operation cycle.

If this command is executed again when the calculation is already started, the previous calculation result will be initialized.

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode (including the virtual controller).

See Also

HealthRBAnalysis, HealthRBStop, HealthRBTRQ, HealthRBSpeed, HealthRBDistance

HealthRBStart Statement Example

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

HealthRBStop Statement

Stops calculation of the usable months and elements for the parts in a particular robot operation cycle.

Syntax

HealthRBStop *robotNumber*

Parameters

robotNumber	Integer expression (1-16) representing the robot number.

Description

Stops calculation for the usable months and elements (torque, speed, and driving amount) of the parts on the specified robot in a particular robot operation cycle.

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode (including the virtual controller).
- Calculation automatically ends when one hour passes since calculation starts.
- If the command is executed after the automatic termination, an error will occur.
- If the command is executed without executing the HealthRBStart command, an error will occur.
- If the command is executed again without executing the HealthRBStart command after the previous HealthRBStop command, an error will occur.

See Also

HealthRBAnalysis, HealthRBStart, HealthRBTRQ, HealthRBSpeed, HealthRBDistance

HealthRBStop Statement Example

```
Function RobotPartAnalysis
   Real month

   Robot 1

   HealthRBStart 1
   Motor On
   Go P0
   Go P1
   Motor Off
   HealthRBStop 1

   month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
   Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

HealthRBTRQ Statement

Displays the torque value which affects the life of the parts on the specified joint.

Syntax

HealthRBTRQ [*robotNumber*] [, *jointNumber*]

Parameters

robotNumber	Optional. Integer expression (1-16) representing the robot number. If omitted, the current robot number will be used.
jointNumber	Integer expression (1-6) representing the joint. If the joint number is not specified, returns values for all the joints. This command is unavailable for the additional axes.

Description

Displays the torque value which affects the life of the parts on the specified joint in robot operation from HealthRBStart to HealthRBStop. The result is a real number from 0 to 1. The maximum torque value is "1".

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode (including the virtual controller).

See Also

HealthRBStart, HealthRBStop, ATRQ

HealthRBTRQ Statement Example

The example below displays the torque value which affects the life of the parts on the Joint #1 of SCARA robot.

```
> HealthRBTRQ 1, 1
 0.020
>
```

HealthRBTRQ Function

Returns the torque value which affects the life of the parts on the specified joint.

Syntax

HealthRBTRQ ([*robotNumber*,] *jointNumber*)

Parameters

robotNumber	Optional. Integer expression (1-16) representing the robot number. If omitted, the current robot number will be used.
jointNumber	Integer expression (1-6) representing the joint. This command is unavailable for the additional axes.

Return Values

The result is a real number from 0 to 1.

Description

Returns the torque value which affects the life of the parts on the specified joint in robot operation from HealthRBStart to HealthRBStop. The result is a real value from 0 to 1. The maximum torque value is "1".

Notes

- This command does not function in Auto mode.
- This command does not function in dry run mode (including the virtual controller).

See Also

HealthRBStart, HealthRBStop, ATRQ

HealthRBTRQ Function Example

```
Function RobotPartAnalysis
   Real healthTRQ

   Robot 1

   HealthRBStart 1
   Motor On
   Go P0
   Go P1
   Motor Off
   HealthRBStop 1

   healthTRQ = HealthRBTRQ(1,1)
   Print "Torque =", Str$(healthTRQ)
Fend
```

HealthRBWarningEnable Statement

Enable or disable the parts consumption alarm notification of specified part related to the robot.

Syntax

HealthRBWarningEnable *robotNumber, partType* [, On/Off]

Parameters

robotNumber Integer expression (1-16) representing the robot number.

partType Integer expression (1-6) or the following constants representing the robot parts.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

On/Off On: Enable the parts consumption alarm notification.

Off: Disable the parts consumption alarm notification

Return Values

If On/Off parameters are omitted, the current On/Off settings are displayed.

Description

When the parts consumption alarm of the specified part occurs, set whether to notify the parts consumption alarm.

Notes

If the parts consumption alarm of the specified part is disabled, the parts consumption alarm will not be notified when the recommended replacement time is passed. Be careful to set when using this command.

See Also

HealthRBAlarmOn

HealthRBWarningEnable Example

Example to disable the parts consumption alarm of the grease part of SCARA robot 1.

```
> HealthRBWarningEnable 1, HEALTH_ROBOT_TYPE_GREASE, Off
```

Example to display the parts consumption alarm settings of the grease part of SCARA robot 1.

```
> HealthRBWarningEnable 1, HEALTH_ROBOT_TYPE_GREASE
GREASE  Off
>
```

HealthRBWarningEnable Function

Returns the setting status of the parts consumption alarm notification of specified part related to the robot.

Syntax

HealthRBWarningEnable(*robotNumber*, *partType*)

Parameters

robotNumber	Integer expression (1-16) representing the robot number you want to obtain the remaining months before the recommended replacement time.
partType	Integer expression (1-6) or the following constants representing the parts you want to obtain the remaining months before the recommended replacement time.

Constant	Value	Mode
HEALTH_ROBOT_TYPE_BATTERY	1	Specifies the batteries.
HEALTH_ROBOT_TYPE_BELT	2	Specifies the timing belts.
HEALTH_ROBOT_TYPE_GREASE	3	Specifies the grease.
HEALTH_ROBOT_TYPE_MOTOR	4	Specifies the motors.
HEALTH_ROBOT_TYPE_GEAR	5	Specifies the reduction gear units.
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	Specifies the ball screw spline.

Return Values

Returns the setting values of the parts consumption alarm in integer.
1: On
0: Off

See Also

HealthRBAlarmOn

HealthRBWarningEnable Function Example

Example to display the parts consumption alarm settings of the grease part of SCARA robot 1.

```
Print HealthRBWarningEnable(1, HEALTH_ROBOT_TYPE_GREASE )
```

Here Statement

Teach a robot point at the current position.

Syntax

Here *point*

Parameters

point	**P***number* or **P**(*expr*) or point label.

Notes

The Here statement and Parallel Processing

You cannot use both of the Here statement and parallel processing in one motion command like this:

```
Go Here :Z(0) ! D10; MemOn 1 !
```

Be sure to change the program like this:

```
P999 = Here
Go P999 Here :Z(0) ! D10; MemOn 1 !
```

The Here statement and Multitask

If the Here statement is executed in a multitask function executed by Xqt while the robot is moved by Move, Go, etc., in the main task, the task will be stopped due to an error.
Current robot position can be retrieved by CurPos.

Example

```
Function Xqt_PrintHere
         Do
             Print CurPOS
             Wait 0.1
         Loop
Fend
Function main
         Xqt 10, Xqt_PrintHere
         Go P0
Fend
```

See Also

Here Function, CurPos

Here Statement Example

```
Here P1
Here pick
```

Here Function

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.

See Also

Here Statement

Here Function Example

```
P1 = Here
```

Hex$ Function

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 to 9 or A to F. Hex$ is especially useful for examining the results of the Stat function.

See Also

Str$, Stat, Val

Hex$ Function Example

```
> print hex$(stat(0))
A00000
> print hex$(255)
FF
```

Hofs Statement

Displays or sets the offset pulses between the encoder origin and the home sensor.

Syntax

(1) **Hofs** *j1Pulses, j2Pulses, j3Pulses, j4Pulses [, j5pulses, j6pulses] [, j7pulses] [, j8pulses, j9pulses]*
(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.
j5Pulses	For 6 axis robots (including N series). Integer expression representing joint 5 offset pulses.
j6Pulses	For 6 axis robots (including N series). Integer expression representing joint 6 offset pulses.
j7Pulses	For 7 axis robots. Integer expression representing joint 7 offset pulses.
j8Pulses	For additional S axis. Integer expression representing joint 8 (additional S axis) offset pulses.
j9Pulses	For additional T axis. Integer expression representing joint 9 (additional T axis) offset pulses.

Return Values

Displays current Hofs values when used without parameters.

Description

Hofs displays or sets the home position offset pulses. Hofs specifies the offset from the encoder 0 point (Z phase) 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.

Notes

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.

Reset JointAccuracy (only the supported products of joint accuracy offset)

For the supported products of joint accuracy offset, when sets the home position offset pulses by Hofs, offset value of joint accuracy offset set in JointAccuracy is reset to "0" for the changed axis. When not want to reset the accuracy value that set in the JointAccuracy, use HofsJointAccuracy.

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 Save and Load commands in the [System Configuration] dialog-[Robot]-[Calibration] from the System Configuration menu.

After executing this command, start the Safety Function Manager (only for the Controllers with Safety Board)

For the Controllers with Safety Board, the Hofs value of the Controller and the Hofs value of the Safety Board that implements the safety function must match.
If this command executed with these Controllers, a warning occurs because only the Hofs value of the Controller is changed and there is a difference with the Safety Board setting.
Therefore, after executing this command, start the Safety Function Manager to refresh the Safety board settings.
For more details, refer to the following manual.
Robot Controller Safety Function Manual

See Also

Calib, CalPls, JointAccuracy, HofsJointAccuracy, Home, Hordr, MCal, SysConfig

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

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. The additional S axis is 8 and T axis is 9.

Return Values

The offset pulse value (integer value, in pulses).

See Also

Calib, CalPls, Home, Hordr, MCal, SysConfig

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
```

HofsJointAccuracy Statement

Sets and displays the offset pulses between the encoder origin and the software origin without changing the joint accuracy offset values.

Syntax

(1) HofsJointAccuracy *j1Pulses, j2Pulses, j3Pulses, j4Pulses*
[, *j5pulses, j6pulses*] [, *j7pulses*] [, *j8pulses, j9pulses*]
(2) HofsJointAccuracy

Parameters

j1Pulses	An expression or a numerical value representing Joint #1 offset pulses (integer).
j2Pulses	An expression or a numerical value representing Joint #2 offset pulses (integer).
j3Pulses	An expression or a numerical value representing Joint #3 offset pulses (integer).
j4Pulses	An expression or a numerical value representing Joint #4 offset pulses (integer).
j5Pulses	Optional for 6-axis robots (including N series). An expression or a numerical value representing Joint #5 offset pulses (integer).
j6Pulses	Optional for 6-axis robots (including N series). An expression or a numerical value representing Joint #6 offset pulses (integer).
j7Pulses	Optional for Joint type 7-axis robots. An expression or a numerical value representing Joint #7 offset pulses (integer).
j8Pulses	Optional for additional S axis. An expression or a numerical value representing Joint #8 (additional S axis) offset pulses (integer).
j9Pulses	Optional for additional T axis. An expression or a numerical value representing Joint #9 (additional T axis) offset pulses (integer).

Return Values

Displays current Hofs values when used without parameters.

Description

HofsJointAccuracy displays or sets the home position offset pulses without changing the joint accuracy offset values. For more information about joint accuracy offset-compatible models, refer to the Manipulator manual.

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 is used to set the encoder position matching the mechanical zero point as the zero point in the software application.

When values change at any joint, the Hofs statement resets the corresponding joint accuracy offset value set with JointAccuracy to 0, but the HofsJointAccuracy statement does not reset the value. Use HofsJointAccuracy if you do not wish to reset the joint accuracy offset value to 0.

Note

Do not use Hofs values unless absolutely necessary.

Hofs values have been precisely set at the factory. Changing these values unnecessarily may lead to positioning errors and other hazardous unexpected behavior. Do not change Hofs values unless absolutely necessary.

See Also

JointAccuracy, Hofs

HofsJointAccuracy Statement Example

The following is a simple example using the Command window.
This example 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 both be "0". It then displays the current home offset values. Joint accuracy offset values do not change when using HofsJointAccuracy.

```
> JointAccuracy 1
1000, 420, 100, 240
> HofsJointAccuracy -545, 514, 0, 0

> HofsJointAccuracy
-545, 514, 0, 0
> JointAccuracy 1
1000, 420, 100, 240
>
```

Home Statement

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.

Normally, for SCARA robots (including RS series), 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 Error

Attempting to Home without HomeSet Values Defined

Attempting to Home the robot without setting the HomeSet values will result in an Error 2228 being issued.

See Also

HomeClr, HomeDef, HomeSet, Hordr

Home Statement 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 Command window like this:

```
> home
>
```

HomeClr Function

Clears the home position definition.

Syntax

HomeClr

Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

HomeDef, HomeSet

HomeClr Function Example

This example uses the HomeClr function in a program:

```
Function ClearHome

   If HomeDef = True Then
      HomeClr
   EndIf
Fend
```

HomeDef Function

Returns whether home position has been defined or not.

Syntax

HomeDef

Return Values

True if home position has been defined, otherwise False.

See Also

HomeClr, HomeSet

HomeDef Function Example

This example uses the HomeDef function in a program:

```
Function DisplayHomeSet

   Integer i

   If HomeDef = False Then
      Print "Home is not defined"
   Else
      Print "Home values:"
      For i = 1 To 4
         Print "J", i, " = ", HomeSet(i)
      Next i
   EndIf
Fend
```

HomeSet Statement

Specifies and displays the Home position.

Syntax

(1) **HomeSet** *j1Pulses*, *j2Pulses*, *j3Pulses*, *j4Pulses*
[, *j5Pulses*, *j6Pulses*] [, *j7Pulses*] [, *j8Pulses*, *j9Pulses*]
(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 (including N series). The home position encoder pulse value for joint 5.
j6Pulses	Optional for 6-axis robots (including N series). The home position encoder pulse value for joint 6.
j7Pulses	Optional for Joint type 7-axis robots. The home position encoder pulse value for joint 7.
j8Pulses	Optional for additional S axis. The home position encoder pulse value for joint 8 (additional S axis).
j9Pulses	Optional for additional T axis. The home position encoder pulse value for joint 9 (additional T axis).

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.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Potential Errors

Attempting to Home without HomeSet Values Defined:

Attempting to Home the robot without setting the HomeSet values will result in an Error 2228 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 2228.

See Also

Home, HomeClr, HomeDef, Hordr, Pls

HomeSet Statement Example

The following examples are done from the monitor window:

```
> 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

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. The additional S axis is 8 and T axis is 9.

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) = 0 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

Specifies or displays the order of the axes returning to their Home positions.

Syntax

(1) **Hordr** *step1, step2, step3, step4* [, *step5*] [, *step6*] [, *step7*] [, *step8*] [, *step9*]
(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	Bit pattern that defines which joints should home during the 5th step of the homing process.
step6	Bit pattern that defines which joints should home during the 6th step of the homing process.
step7	Bit pattern that defines which joints should home during the 7th step of the homing process.
step8	Bit pattern that defines which joints should home during the 8th step of the homing process.
step9	Bit pattern that defines which joints should home during the 9th 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, 6, or 9 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 SCARA robots (including RS series, 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.

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	7	8	9
Bit Number:	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8
Binary Code:	&B0001	&B0010	&B0100	&B1000	&B10000	&B100000	&B1000000	&B10000000	&B100000000

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

Home, HomeSet

Hordr Statement Example

Following are some command window examples for SCARA robots (including RS series, 4 axis robots):

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

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

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 controller has been turned on and running SPEL. (Accumulated Operating Time) Time is always displayed in units of hours.

See Also

Time

Hour Statement Example

The following example is done from the Command window:

```
> hour
2560
>
```

Hour Function

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
```

If…Then…Else…EndIf Statement

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.

(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.

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.

True in the Conditions:

Constant True is –1 and the type is Boolean, so you need to be careful when using it in a comparing condition with other type variable.

```
Function main
   Integer i
   i = 3
   If i = True Then
      Print "i=TRUE"
   EndIf
Fend
```

When you execute the program above, “i=TRUE” is displayed.
The judgement of condition including the Boolean type is done with “0” or “non-0”.
If the value of “i” is not “0”, it is considered that the condition is established and “i=TRUE” is displayed.

See Also

Else, Select...Case, Do...Loop

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

Imports a point file into the current project for the specified robot.

Syntax

ImportPoints *sourcePath, filename* [*, robotNumber*]

Parameters

sourcePath	String expression containing the specific path and file to import into the current project. The extension can be ".pts" or ".pnt" (EPSON RC+ 3.x and 4.x format). See ChDisk for the details.
fileName	String expression containing the specific file to be imported to in the current project for the current robot. The extension must be ".pts". You cannot specify a file path and fileName doesn't have any effect from ChDisk. See ChDisk for the details.
robotNumber	Optional. Integer expression that specifies which robot the point file should be associated with. If robotNumber = 0, then the point file is imported as a common point file. If robotNumber is omitted, the current robot number is used.

Description

ImportPoints copies a point file into the current project and adds it to the project files for the specified 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.

The point data is stored in the Compact Flash inside of the controller. Therefore, ImportPoints starts writing into the Compact Flash. Frequent writing into the Compact Flash will shorten the Compact Flash lifetime. We recommend using ImportPoints only for saving the point data.

Potential Errors

File Does Not Exist

If *sourcePath* does not exist, an error will occur.

A Path Cannot be Specified

If *fileName* contains a path, an error will occur.

Point file for another robot.

If *fileName* is a point file for another robot, an error will occur.

See Also

LoadPoints, Robot, SavePoints

ImportPoints Statement Example

```
Function main
  Robot 1
  ImportPoints "c:\mypoints\model1.pts", "robot1.pts"
  LoadPoints "robot1.pts"
Fend
```

In Function

Returns the status of the specified Byte port. Each port contains 8 input channels.

Syntax

In(byte*portNumber*)

Parameters

byteportNumber	Integer number representing one eight bit port (one byte).

Return Values

Returns an integer value between 0 and 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 to 255. Please review the chart below to see how the integer return values correspond to individual input channels.

Input Channel Result (Using Byte port #0)

Return Values	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 Byte port #2)

Return Values	23	22	21	20	19	18	17	16
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 let's assume that input channels 20, 21, 22, and 23 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 8 input channels status of byte port 2 and makes sure that channels 20, 21, 22, and 23 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 20, 21, 22, and 23 to all be On, then the result of In(2) will be 240 or larger. (We don't care about Inputs 16, 17, 18, and 19 in this case so any values between 240-255 will allow the program to proceed.)

```
Function main
    Integer var1
    var1 = In(2)      'Get 8 input channels status of byte port 2
    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 20,21,22, and 23 for"
        Print "proper state for cycle start and then"
        Print "start program again"
    EndIf
Fend
```

We cannot set inputs from the command window but we can check them. For the examples shown below, we will assume that the Input channels 1, 5, and 15 are On. All other inputs are Off.

```
> print In(0)
34
> print In(1)
128
> print In(2)
0
```

InBCD Function

Returns the input status of 8 inputs using BCD format. (Binary Coded Decimal)

Syntax

InBCD(*portNumber***)**

Parameters

portNumber	Integer number representing one eight bit port (one byte).

Return Values

Returns as a Binary Coded Decimal (0-9), the input status of the input port (0 to 99).

Description

InBCD simultaneously reads 8 input lines using the BCD format. The *portNumber* parameter for the InBCD instruction defines which group of 8 inputs to read where *portNumber* = 0 means inputs 0 to 7, *portNumber* = 1 means inputs 8 to 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 to 9 for each digit, every I/O combination cannot be met. The able below shows some of the possible I/O combinations and their associated return values assuming that *portNumber* is 0.

Input Settings (Input number)

Return Values	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.

Note

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 to 255 vs. 0 to 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 Function Example

Some simple examples from the Command 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 load inertia and eccentricity for current robot.

Syntax

Inertia [*loadInertia* [, *eccentricity*]]
Inertia

Parameters

loadInertia	Optional (It is not possible to omit only *loadInertia*). Real expression that specifies total moment of inertia in kgm2 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.
When [eccentricity] is omitted, the entered [loadInertia] will be set and the default value [eccentricity] will be set.
It is not possible to omit only [loadInertia].

Note that when you specify a value smaller than the actual value to the inertia or eccentricity, excessive acceleration and deceleration values will be set and may damage the manipulator. In addition, it may cause an error or impact. It may cause the function will not work best, the life of the parts be shortened, or the belt be misaligned due to teeth losing.

Description

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.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.
You can also set by following “Weight, Inertia, and Eccentricity/offset Measurement Utility”.
The following manual describes the details.
EPSON RC+ 7.0 User’s Guide 6.18.12 Weight, Inertia, and Eccentricity/offset Measurement Utility

Notes

Inertia Values Are Not Changed by Turning Main Power Off

The Inertia values are not changed by turning power off. Once the value is set, the value is memorized in the controller.
When nothing is changed, it will remain at the previously set value.

See Also

Inertia Function

For details of Hand, refer to the *Hand Function Manual.*

Inertia Statement Example

```
Inertia 0.02, 1
```

Inertia Function

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 kgm^2.
	2: Causes function to return eccentricity in mm.

Return Values

Real value of the specified setting.

See Also

Inertia Statement

For details of Hand, refer to the *Hand Function Manual.*

Inertia Function Example

```
Real loadInertia, eccentricity

loadInertia = Inertia(1)
eccentricity = Inertia(2)
```

InPos Function

Returns the position status of the specified robot.

Syntax

InPos

Return Values

True if position has been completed successfully, otherwise False.

See Also

CurPos, FindPos, WaitPos

InPos Function Example

```
Function main

  P0 = XY(0, -100, 0, 0)
  P1 = XY(0, 100, 0, 0)

  Xqt MonitorPosition
  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

Receives input data from the display device 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 data from the display device and assigns the data to the variable(s) used with the Input instruction.

When executing the Input instruction, a "?" prompt appears at the display 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 Error

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 2505 will occur.

See Also

Input #, Line Input, Line Input #, Print, String

Input Statement Example

This is a simple program example using Input statement.

```
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, -10000
A = -10000
B = 25 C = -10000
```

Input # Statement

Allows string or numeric data to be received from a file, communications port, or database and stored in one or more variables.

Syntax

Input **#***portNumber, varName* [*, varName, varName,...*]

Parameters

#portNumber	The ID number that specifies a file, communication port, database, or device. The File number can be specified in ROpen, WOpen, and AOpen statements. Communication port number can be specified in OpenCom (RS-232C) and OpenNet (TCP/IP) statements. The database number can be specified in OpenDB statement. Device ID is: 21 RC+ 24 TP (TP1 only) 20 TP3
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

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

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.

Maximum data length

This command can handle up to 256 bytes.
However, the target is the database, it can handle up to 4096 bytes.
If the target is the communication port (TCP/IP), it can handle up to 1024 bytes.

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 or blank (“ ”).
- When more than one string variable or both of numeric variable and string variable is specified, the numeric data has to be separated by a comma (“,”) character or blank (“ “).
- The input data type must match the variable type.

The following programs are examples to exchange the string variable and numeric variable between the controllers using a communication port.

Sending end (Either pattern is OK.)
```
Print #PortNum, "$Status,", InData, OutData
Print #PortNum, "$Status", ",",InData, OutData
```
Receiving end
```
Input #PortNum, Response$, InData, OutData
```

Potential Error

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 2505 will occur.

See Also

Input, Line Input, Line Input #, Print #, Read, ReadBin

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     'Get a numeric value from Port#1
    Input #1, B$    'Get a string from Port#1
    CloseCom #1
Fend
```

InputBox Statement

Displays a prompt in a dialog box, waits for the operator to input text or choose a button, and returns the contents of the box.

Syntax

InputBox *prompt, title, default, data$*

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 (“”).
data$	A string variable which will contain what the operator entered. If the operator clicks Cancel, this string will be “@”.

Description

InputBox displays the dialog and waits for the operator to click OK or Cancel. *data* is a string that contains what the operator typed in.

See Also

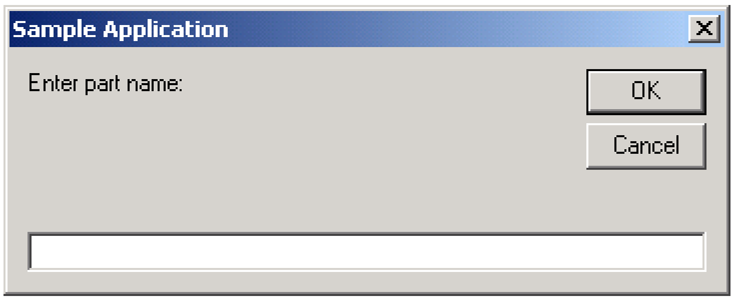
MsgBox

InputBox Statement Example

This function shows an InputBox example.

```
Function GetPartName$ As String
    String prompt$, title$, data$

    prompt$ = "Enter " + Chr$(34) + "part name" + Chr$(34) + ":"
    title$ = "Sample Application"
    InputBox prompt$, title$, "", data$
    If data$ <> "@" Then
        GetPartName$ = data$
    EndIf
Fend
```

The following picture shows the example output from the InputBox example code shown above.



Restriction

If the prompt, title, and default of parameter contain a half-width comma ",", the string cannot be displayed correctly.
Use a string that does not contain a half-width comma.

InReal Function

Returns the input data of 2 words (32 bits) as the floating-point data (IEEE754 compliant) of 32 bits.

Syntax

InReal(*WordPortNumber*)

Parameters

WordPortNumber	Integer expression representing the I/O Input Word.

Return Values

Returns the input port status in Real type number.

Description

From the input word port specified by the word port number, retrieve the 2 input word values as IEEE754 Real type value. Input word label can be used for the word port number parameter.
InReal Function cannot be used for the Wait command, or the condition of Till, Find, Sense.

See Also

In, InW, InBCD, Out, OutW, OpBCD, OutReal

InW Function Example

```
Real realVal

realVal = InReal(32)
```

InsideBox Function

Returns the check status of the approach check area.

Syntax

InsideBox(*AreaNum [, robotNumber | All]*)

Parameters

AreaNum	Integer expression from 1 to 15 representing which approach check area to return status for.
robotNumber	Integer value that contains the robot number you want to search. If omitted, the current robot will be specified. If you specify All, True is returned if one robot is in the check area.

Return Values

True if the robot end effector approaches the specified approach check area, otherwise False.

See Also

Box, BoxClr, BoxDef, GetRobotInsideBox, InsidePlane

Note

You can use the Wait statement with InsideBox to wait for the result of the InsideBox function in EPSON RC+ 5.0. However, you cannot use it in EPSON RC+ 6.0 and 7.0.
In this case, use the GetRobotInsideBox function instead of the InsideBox function.

Correspondence table

RC+ version	Robot Controller	Wait	Till, Find, Sense, Trap	Other commands (such as Print)/ branch decision processing	Use of GetRobotInsideBox Function
RC+ 7.0	RC700 series	Not available	Not available	Available	All available
RC+ 7.0	RC90 series	Not available	Not available	Available	All available
RC+ 6.0	RC620	Not available	Not available	Available	All available
RC+ 5.0	RC90 series	Available	Not available	Available	Not available

Not available: Unavailable combination
Available: Available combination
All available: Available for Wait, Till, Find, Sense, Trap, Print, and branch decision processing.

InsideBox Function Example

The following program checks Robot 1 is in the check area (Box 3) or not.

```
Function PrintInsideBox
    If InsideBox(3,1) = True Then
         Print "Inside Box3"
    Else
         Print "Outside Box3"
    EndIf
Fend
```

InsidePlane Function

Returns the check status of the approach check plane.

Syntax

InsidePlane(*PlaneNum* [, *robotNumber* | *All*])

Parameters

PlaneNum	Integer expression from 1 to 15 representing which approach check plane to return status for.
robotNumber	Integer value that contains the robot number you want to search. If omitted, the current robot will be specified. If you specify All, True is returned if one robot is in the check area.

Return Values

True if the robot end effector approaches the specified approach check plane, otherwise False.

See Also

InsideBox, GetRobotInsidePlane, Plane, PlaneClr, PlaneDef

Note

You can use the Wait statement with InsidePlane to wait for the result of the InsidePlane function in EPSON RC+ 5.0. However, you cannot use it in EPSON RC+ 6.0 and 7.0.
In this case, use the GetRobotInsidePlane function instead of the InsidePlane function.

Correspondence table

RC+ version	Robot Controller	Wait	Till, Find, Sense, Trap	Other commands (such as Print)/ branch decision processing	Use of GetRobotInsidePlane Function
RC+ 7.0	RC700 series	Not available	Not available	Available	All available
RC+ 7.0	RC90 series	Not available	Not available	Available	All available
RC+ 6.0	RC620	Not available	Not available	Available	All available
RC+ 5.0	RC90 series	Available	Not available	Available	Not available

Not available: Unavailable combination
Available: Available combination
All available: Available for Wait, Till, Find, Sense, Trap, Print, and branch decision processing.

InsidePlane Function Example

This is an example to check Robot 1 is in the check plane (Plane 3).

```
Function PrintInsidePlane
      If InsidePlane(3,1) = True Then
            Print "Inside Plane3"
      Else
            Print "Outside Plane3"
      EndIf
Fend
```

InStr Function

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

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 Command window are as follows:

```
> Print Int(5.1)
5
> Print Int(0.2)
0
> Print Int(-5.1)
-6
>
```

Int32 Statement

Declares variables of Int32 type. (4 byte integer type variable).

Syntax

Int32 *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameters

varName	Variable name which the user wants to declare.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

Int32 is used to declare variables as type integer. Integer variables can contain values from -2147483648 to 2147483647. 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, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Int32 Statement Example

The following example shows a simple program that declares some variables using Int32.

```
Function int32test
    Int32 A(10)         'Single dimension array of Int32
    Int32 B(10, 10)     'Two dimension array of Int32
    Int32 C(5, 5, 5)    'Three dimension array of Int32
    Int32 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
```

Int64 Statement

Declares variables of Int64 type. (8 byte integer type variable).

Syntax

Int64 *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameters

varName	Variable name which the user wants to declare.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows

(ubound1, [ubound2], [ubound3])

ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension.

The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1.

When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

Int64 is used to declare variables as type integer. Integer variables can contain values from -9223372036854775808 to 9223372036854775807. 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, Int32, Integer, Long, Real, Short, String, UByte, UInt32, UShort UInt64

Int64 Statement Example

The following example shows a simple program that declares some variables using Int64.

```
Function int64test
    Int64 A(10)          'Single dimension array of Int64
    Int64 B(10, 10)      'Two dimension array of Int64
    Int64 C(5, 5, 5)     'Three dimension array of Int64
    Int64 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
```

Integer Statement

Declares variables of Integer type. (2 byte integer type variable).

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 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

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, Int32, Int64, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

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(5, 5, 5)     '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

Returns the status of the specified input word port. Each word port contains 16 input bits.

Syntax

InW(*WordPortNum*)

Parameters

WordPortNum Integer expression representing the I/O Input Word.

Return Values

Returns the current status of inputs (long integers from 0 to 65535).

Note

Rule of word port which contains the input bit of Real Time I/O

Word ports =1, 3, 17, 19 return the state of the input port with an integer from 0 to 255.
The input bit of the Real Time I/O is not reflected.

See Also

In, Out, OutW

InW Function Example

```
Long word0

word0 = InW(0)
```

IODef Function

Returns whether the specified input or output bit, byte, word, or I/O label are defined.

Syntax

IODef (*IOType, IOWidth, portNumber*)
IODef (*IOlabel*)

Parameters

IOType	Integer expression representing the type of I/O. 0 - Input 1 - Output 2 - Memory
IOWidth	Integer expression representing the width of the port: 1(bit), 8 (byte), or 16 (word).
portNumber	Integer expression representing the bit, byte, or word port number to return the label for.
IOlabel	String expression that specifies the standard I/O or memory I/O label.

Return Values

True if the specified input or output bit, byte, word or the I/O label are defined, otherwise False.

See Also

IOLabel$, IONumber

IODef Function Example

```
Integer i

For i = 0 To 15
  If IODef( 0, 1, i) = TRUE Then
    Print "Port " , i, " is defined"
  Else
    Print "Port " , i, " is undefined"
  EndIf
Next i
```

IOLabel$ Function

Returns the I/O label for a specified input or output bit, byte, or word.

Syntax

IOLabel$(*IOType, IOWidth, portNumber*)

Parameters

IOType	Integer expression representing the type of I/O. 0 - Input 1 - Output 2 - Memory
IOWidth	Integer expression representing the width of the port: 1(bit), 8 (byte), or 16 (word).
portNumber	Integer expression representing the bit, byte, or word port number to return the label for.

Return Values

String containing the label.

See Also

PLabel$, IONumber, IODef

IOLabel$ Function Example

```
Integer i

For i = 0 To 15
  Print "Input ", i, ": ", IOLabel$(0, 1, i)
Next i
```

IONumber Function

Returns the I/O number of the specified 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 I/O port number (bit, byte, word) of the specified I/O label. If there is no such I/O label, an error will be generated.

See Also

IOLabel$, IODef

IONumber Function Example

```
Integer IObit

IObit = IONumber("myIO")

IObit = IONumber("Station" + Str$(station) + "InCycle")
```

J1Angle Statement

Sets the J1Angle attribute of a point.

Syntax

(1) **J1Angle** *point* [, *Step*]
(2) **J1Angle**

Parameters

point	**P***number* or **P**(*expr*) or point label.
Step	Optional. Real value that specifies the set value.

Result

The J1Angle attribute can be used for the RS and N robot series.
If *Step* is omitted, the J1Angle value for the specified point will be displayed.
If both parameters are omitted, the J1Angle value of the current robot position will be displayed.

RS series:	Specify the angle of the Joint #1 when both X and Y coordinate values of a point are "0" (singularity). For other robot series points, J1Angle has no meaning.
N series:	Specify the angle of the Joint #1 when the axis centers of "Joint #1, #4, and #6", "Joint #1 and #6", or "Joint #1 and #4" are on the straight line. For other robot series points (not singularity), J1Angle has no meaning.

See Also

Hand, J1Angle Function, J1Flag, J2Flag, J4Angle, J4Angle Function

J1Angle Statement Example

```
J1Angle P0, 10.0
J1Angle P(mypoint), 0.0
```

J1Angle Function

Returns the J1Angle attribute of a point.

Syntax

J1Angle [(*point*)]

Parameters

point	Point expression Optional. If omitted, returns the J1Angle setting of the current robot position.

Return Values

The J1Angle attribute can be used for the RS and N robot series.
Returns the angle of Joint 1 when both X and Y coordinate values of a point are “0” (singularity) in a real value. The J1Angle attribute can be used for the RS series.

RS series:	Returns an integer value representing the angle of the Joint #1 when both X and Y coordinate values of a point are “0” (singularity).
N series:	Returns an integer value representing the angle of the Joint #1 when the axis centers of “Joint #1, #4, and #6”, “Joint #1 and #6”, or “Joint #1 and #4” are on the straight line.

See Also

Hand, J1Angle, J1Flag, J2Flag, J4Angle, J4Angle Function

J1Angle Function Example

```
Print J1Angle(pick)
Print J1Angle(P1)
Print J1Angle
```

J1Flag Statement

Specifies the J1Flag attribute of a point.

Syntax

(1) **J1Flag** *point* [, *value*]
(2) **J1Flag**

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Optional. Integer expression. For RS series Manipulator: 0 (/J1F0) J1 range is -90 to +270 degrees 1 (/J1F1) J1 range is from -270 to -90 or +270 to +450 degrees For C8, C12 series Manipulator: 0 (/J1F0) J1 range is 0 to -180 or 0 to +180 degrees 1 (/J1F1) J1 range is -180 to -240 or +180 to -240 degrees

Return Values

The J1Flag attribute specifies the range of values for joint 1 for one point. If *value* is omitted, the J1Flag value for the specified point is displayed. When both parameters are omitted, the J1Flag value is displayed for the current robot position.

See Also

Hand, J1Flag Function, J2Flag

J1Flag Statement Example

```
J1Flag P0, 1
J1Flag P(mypoint), 0
```

J1Flag Function

Returns the J1Flag attribute of a point.

Syntax

J1Flag [**(***point***)**]

Parameters

point	Optional. Point expression. If *point* is omitted, then the J1Flag setting of the current robot position is returned.

Return Values

0	/J1F0
1	/J1F1

See Also

Hand, J1Flag Statement, J2Flag

J1Flag Function Example

```
Print J1Flag(pick)
Print J1Flag(P1)
Print J1Flag
Print J1Flag(Pallet(1, 1))
```

J2Flag Statement

Sets the J2Flag attribute of a point.

Syntax

(1) **J2Flag** *point* [, *value*]
(2) **J2Flag**

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Optional. Integer expression. 0 (/J1F0) J1 range is -180 to +180 degrees 1 (/J1F1) J1 range is -240 to -180 or +180 to +240 degrees

Return Values

The J2Flag attribute specifies the range of values for joint 2 for one point. If *value* is omitted, the J2Flag value for the specified point is displayed. When both parameters are omitted, the J2Flag value is displayed for the current robot position.

See Also

Hand, J1Flag, J2Flag Function

J2Flag Statement Example

```
J2Flag P0, 1
J2Flag P(mypoint), 0
```

J2Flag Function

Returns the J2Flag attribute of a point.

Syntax

J2Flag [**(***point***)**]

Parameters

point	Optional. Point expression. If *point* is omitted, then the J2Flag setting of the current robot position is returned.

Return Values

0	/J2F0
1	/J2F1

See Also

Hand, J1Flag, J2Flag Statement

J2Flag Function Example

```
Print J2Flag(pick)
Print J2Flag(P1)
Print J2Flag
Print J2Flag(P1 + P2)
```

J4Angle Statement

Sets the J4Angle attribute of a point.

Syntax

(1) J4Angle *point* [, *value*]
(2) J4Angle

Parameters

point	**P***number* or **P**(*expr*) or point label.
value	Optional. Integer expression.

Result

The J4Angle attribute is used only for N robot series.
It specifies the angle of the Joint #4 when the axis centers of the Joint #4 and #6 are on the straight line.
If the point is not singularity, J4Angle has no meaning.
If *value* is omitted, the J4Angle value for the specified point is displayed. When both parameters are omitted, the J4Angle value is displayed for the current robot position.

See Also

Hand, J1Angle, J1Angle Function, J4Angle Function

Note

When both J4Flag and J4Angle are used, J4Angle is prioritized as follows:

J4Angle P0,0
J4Flag P0,1

J4Angle Example

```
J4Angle P0, 10.0
J4Angle P(mypoint), 0.0
```

J4Angle Function

Returns the J4Angle attribute of a point.

Syntax

J4Angle [(*point*)]

Parameters

point	Optional. Point expression. If *point* is omitted, then the J4Angle setting of the current robot position is returned.

Return Values

Returns an integer value representing the angle of the Joint #4 when the axis centers of the Joint #4 and #6 are on the straight line.
The J4Angle attribute is used only for N robot series.

See Also

Hand, J1Angle, J1Angle Function, J4Angle

J4Angle Function Example

```
Print J4Angle(pick)
Print J4Angle(P1)
Print J4Angle
```

J4Flag Statement

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 omitted, 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

Returns the J4Flag attribute of a point.

Syntax

J4Flag [(*point*)]

Parameters

point	Optional. Point expression. If *point* is omitted, 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

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 to /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 omitted, 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

Note

Range of J6Flag differs depending on manipulator models

C4	:	0 - 127 (/J6F0 to /J6F127)
C8, C12	:	0 - 81 (/J6F0 to /J6F81)
N2	:	0 - 40 (/J6F0 to /J6F40)
N6	:	0 - 61 (/J6F0 to /J6F61)

J6Flag Statement Example

```
J6Flag P0, 1
J6Flag P(mypoint), 0
```

J6Flag Function

Returns the J6Flag attribute of a point.

Syntax

J6Flag [(*point*)]

Parameters

point	Optional. Point expression. If *point* is omitted, then the J6Flag setting of the current robot position is returned.

Return Values

0 to 127	/J6F0 to /J6F127

See Also

Elbow, Hand, Wrist, J4Flag, J6Flag

J6Flag Function Example

```
Print J6Flag(pick)
Print J6Flag(P1)
Print J6Flag
Print J6Flag(P1 + P2)
```

JA Function

Returns a robot point specified in joint angles.

Syntax

JA (*j1*, *j2*, *j3*, *j4* [, *j5*, *j6*] [, *j7*] [, *j8*, *j9*] **)**

Parameters

j1 – j9	Real expressions representing joint angles. For linear joints, specifies in units of mm. j5 and j6 are for the 6-axis robot (including N series) and Joint type 6-axis robot. j7 is for the Joint type 7-axis robot. j8 and j9 are for the additional ST axis.

Note

If the angle exceeding the motion range is specified, an out of range error occurs.

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)
```

Joint Statement

Displays the current position for the robot in joint coordinates.

Syntax

Joint

See Also

Pulse, Where

Joint Statement Example

```
>joint
JOINT:  1: -6.905 deg 2:  23.437 deg 3:  -1.999 mm  4: -16.529 deg
>
```

JointAccuracy Statement

Sets and displays the joint accuracy offset values.

Syntax

(1) JointAccuracy *jointNumber, settingValue1, settingValue2, settingValue3, settingValue4*
(2) JointAccuracy *jointNumber*

Parameters

jointNumber	Specifies the joint number.
settingValue1	Specifies the first setting value (integer) as a numerical value. The value is within a range of 0 to 2000.
settingValue2	Specifies the second setting value (integer) as a numerical value. The value is within a range of 0 to 999.
settingValue3	Specifies the third setting value (integer) as a numerical value. The value is within a range of 0 to 2000.
settingValue4	Specifies the fourth setting value (integer) as a numerical value. The value is within a range of 0 to 999.

Joints compatible with joint accuracy offset settings vary from manipulator to manipulator. Setting joint accuracy offset settings on a joint that does not support it will result in an error. For more information about compatible joints, refer to the Manipulator manual.

Return Values

In the case of syntax (2), it displays the current joint accuracy offset corresponding to the joint number.

Description

JointAccuracy sets the offset value for the specified joint. Setting the offset value properly will improve robot trajectory accuracy.

Note

Do not change JointAccuracy unless absolutely necessary.

JointAccuracy contains factory settings that have been precisely set. Changing this value unnecessarily may adversely impact trajectory accuracy. JointAccuracy is set automatically when running the calibration wizard. Do not change JointAccuracy unless absolutely necessary.

Using Calib and Hofs

Running the Calib and Hofs commands when JointAccuracy is set will return the joint accuracy offset value for the altered joint to “0”. To change the Hofs value without changing the JointAccuracy offset value, run HofsJointAccuracy.

See Also

HofsJointAccuracy, Calib, Hofs

JointAccuracy Statement Example

The following is a simple example using the Command window. In this example, the joint accuracy offset value for Joint #1 is set to “1000” for the first setting value, “420” for the second setting value, “100” for the third setting value, and “240” for the fourth setting value. Once set, the current joint accuracy offset value is displayed for Joint #1.

```
> JointAccuracy 1, 1000, 420, 100, 240

> JointAccuracy 1
1000, 420, 100, 240
>
```

JointAccuracy Function

Displays the joint accuracy offset values.

Syntax

JointAccuracy(*jointNumber, paramNumber*)

Parameters

jointNumber	Specifies the joint number.
paramNumber	Specifies the joint number to display with following constants or integer (1 to 4).

Constant	Value	Description
JAC_PARAM1	1:	First Parameter Value
JAC_PARAM2	2:	Second Parameter Value
JAC_PARAM3	3:	Third Parameter Value
JAC_PARAM4	4:	Forth Parameter Value

Return Values

Returns offset values (integer) of the joint accuracy corresponding to the specified joint parameter number.

Description

JointAccuracy

JointAccuracy Statement Example

The example below displays the program using JointAccuracy function.

```
Function DisplayJointAccuracy(joint As Integer)
   Integer i

   Print "Joint ", joint, ", JointAccuracy settings:"
   For i = 1 To 4
      Print "Param ", i, " = ", JointAccuracy(joint,  i)
   Next i
Fend
```

JRange Statement

Defines the permissible working range of the specified joint in pulses.

Syntax

JRange *jointNumber, lowerLimit, upperLmit*

Parameters

jointNumber	Integer expression from 1 to 9 representing the joint for which JRange will be specified. The additional S axis is 8 and T axis is 9.
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.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

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. Turning controller power off will not change the JRange joint limit values.

Maximum and Minimum Working Ranges:

Refer to the specifications in the Manipulator 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 Command window:

```
> JRange 2, -6000, 7000        'Define the 2nd joint range

> JRange 1, 0, 7000            'Define the 1st joint range
```

JRange Function

Returns the permissible working range of the specified joint in pulses.

Syntax

JRange(*jointNumber, paramNumber*)

Parameters

jointNumber	Specifies reference joint number (integer from 1 to 9) by an expression or numeric value. The additional S axis is 8 and T axis is 9.
paramNumber	Integer expression containing one of two values: 1: Specifies lower limit value. 2: Specifies upper limit value.

Return Values

Range configuration (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

Jump Sense detects whether the arm stopped prior to completing a Jump, Jump3, JumpTLZ , or Jump3CP instruction which used a Sense input or if the arm completed the move.

Syntax

JS

Return Values

Returns a True or a False.

True	: When the arm was stopped prior to reaching its target destination because a Sense Input condition was met JS returns a True.
False	: When the arm completes the normal move and reaches the target destination as defined in the Jump instruction JS returns a False.

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 True. When the input condition is not met and the arm reaches the target position, JS returns a False.

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, JumpTLZ, 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, JumpTLZ, Sense

JS Function Example

```
Function SearchSensor As Boolean
    Sense Sw(5) = On

    Jump P0
    Jump P1 Sense
    If JS = TRUE Then
        Print "Sensor was found"
        SearchSensor = TRUE
    EndIf
Fend
```

JT Function

Returns the status of the most recent Jump, Jump3, JumpTLZ, 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, JumpTLZ, 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

Perform a relative move of one joint.

Syntax

JTran *jointNumber, distance*

Parameters

jointNumber	Integer expression representing which joint to move. The additional S axis is 8 and T axis is 9.
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

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*] [*!...!*] [*SYNC*]

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 from C0 to 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.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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.

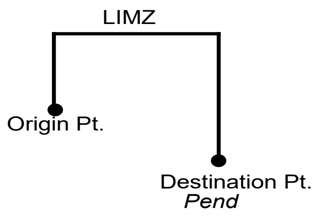
The Jump instruction cannot be executed for the vertical 6-axis robots (including N series). Use the Jump3 instruction.

CP Details

The CP parameter causes acceleration of the next motion command to start when the deceleration starts for the current motion command. In this case the robot will not stop at the destination coordinate and will continue to move to the next point.

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 and C7. The Arch table entries for C0 to 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:



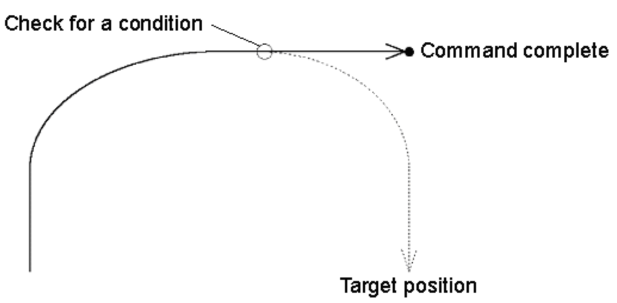
LimZ Details

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 I/O 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 (including N series)

Use Jump3 or Jump3CP for 6-axis robots.

Omitting *archNumber* Parameters

If the archNumber 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 (including N series). On the other hand the Jump instruction cannot be used for 6-axis robots. For SCARA robots (including RS series), 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.

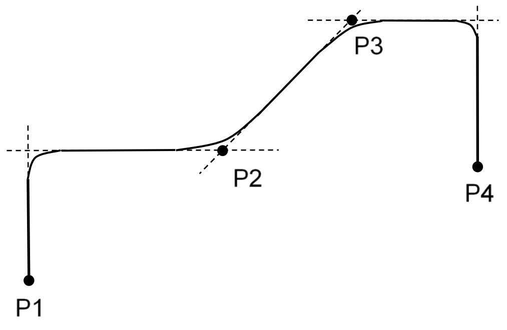
Proper Speed and Acceleration Instructions with Jump:

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

When the CP parameter is specified for 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 the CP parameter is specified for a PTP motion command right before a 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) with several PTP motions.

(Example)
```
Go P1
Jump P2 :Z(-50) C0 LimZ -50 CP
Go P3 :Z(0) CP
Jump P4 C0 LimZ 0
```



Caution for Arch motion

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.

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 4005 will occur.

See Also

Accel, Arc, Arch, Go, JS, JT, LimZ, P#= (Point Assignment), 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. If input #4 never goes high then the arm starts the approach motion and moves to P1. If input #4 goes high then the arm does not execute the approach motion.

```
Function jumptest
    Home
    Go P0
    Go P1
    Sense Sw(4) = On
    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!
```

Here's an example from Command window.

```
> Jump P0                          'Jump to P0
> Jump P0 C0                       'Jump to P0 with arch motion specified as
                                    archNumber C0.
> Jump P0 LimZ -10                 'Jump to P0 of Z -10mm point
> Jump P0  !D0; On 1; D50; Off 1!  'Jump to P0. Turn on the first bit of the output
                                    until the amount of the movement reaches 50%,
                                    and turn off the first bit after 50%.
```

If the first bit of the input is turned On, stop the Jump and go on to the next process.

```
Function main
    (omitted)
  Till Sw(1) = On
  Jump P0 C0 CP Till
    (omitted)
Fend
```

Jump3, Jump3CP Statements

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**] [, LJM [, *orientationFlag*]] [, *searchExpr*] [, *!...!*] [, *SYNC*]
(2) **Jump3CP** *depart, approach, destination* [, **ROT**] [, **C***archNumber*] [, **CP**] [, LJM [, *orientationFlag*]] [, *searchExpr*] [, *!...!*] [, *SYNC*]

Parameters

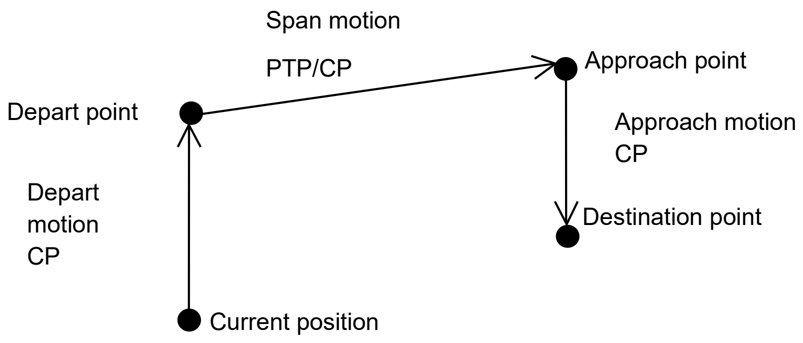
depart	The departure point above the current position using a point expression.
approach	The approach point above the destination position a point expression.
destination	The target destination of the motion using a point expression.
ROT	Optional. :Decides the speed/acceleration/deceleration in favor 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 to C7.)
CP	Optional. Specifies continuous path motion.
LJM	Optional. Convert the target destination using LJM function.
orientationFlag	Optional. Specifies a parameter that selects an orientation flag for LJM function.

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**}

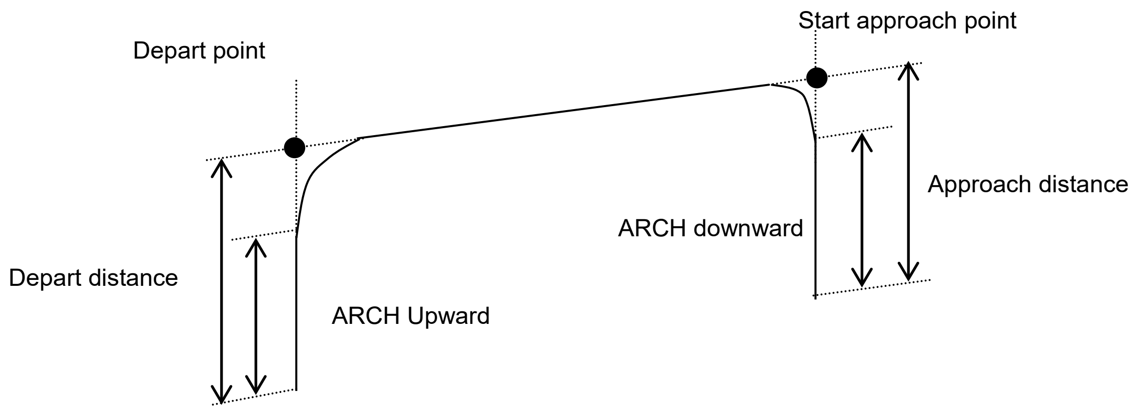
!...!	Optional. Parallel Processing statements can be added to the Jump instruction to cause I/O and other commands to execute during motion.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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

LimZ does not affect Jump3 and Jump3CP

LimZ has no effect on Jump3 or Jump3CP since the span motion is not necessarily perpendicular to the Z axis of the coordinate 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.

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 *destination*.

Pass function of Jump3

When the CP parameter is specified for 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 the CP parameter is specified for a 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) with several PTP motions.

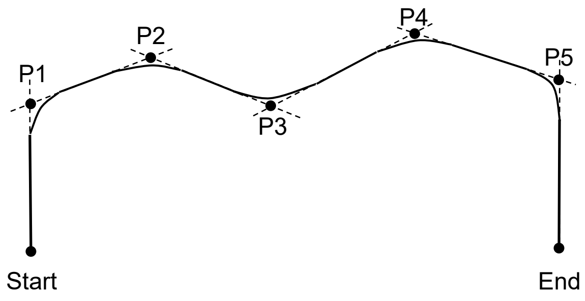
Pass function of Jump3CP

When the CP parameter is specified for 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 the CP parameter is specified for a 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) with several CP motions.
(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)
```



Using Jump3, Jump3CP with LJM

With LJM parameter, the program using LJM function can be more simple.
For example, the following four-line program

```
P11 = LJM(P1, Here, 2)
P12 = LJM(P2, P11, 2)
P13 = LJM(P3, P12, 2)
Jump3 P11, P12, P13
```

can be… the one-line program.

```
Jump3 P1, P2, P3 LJM 2
```

LJM parameter is available for 6-axis (including N series) and RS series robots.
Jump3CP span motion is straight line (CP) motion and it cannot switch the wrist orientation along the way. Therefore, do not use the *orientationFlag* (LJM 1) of LJM function which is able to switch the wrist orientation.

Caution for Arch motion

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.

Potential acceleration errors

When the majority of depart (approach) motion uses the same joint as the span motion

An acceleration error may occur during an arch motion execution by the Jump3 and Jump3CP 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.

See Also

Accel, Arc, Arch, Go, JS, JT, Point Expression, Pulse, Sense, Speed, Stat, Till

Jump3 Statement Example

```
' 6 axis robot (including N series) motion which works like Jump of SCARA robot
Jump3  Here :Z(100), P3 :Z(100), P3

' Depart and approach use Z tool coordinates
Jump3  Here -TLZ(100), P3 -TLZ(100), P3

' Depart uses base Z and approach uses tool Z
Jump3  Here +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
```

JumpTLZ Statement

3D gate motion.
JumpTLZ is a combination of two CP motions and one PTP motion.

Syntax

JumpTLZ *destination, TLZ movement* [, **C***archNumber*] [, **CP**] [, LJM [, *orientationFlag*]] [, *searchExpr*] [, *!...!*] [, *SYNC*]

Parameters

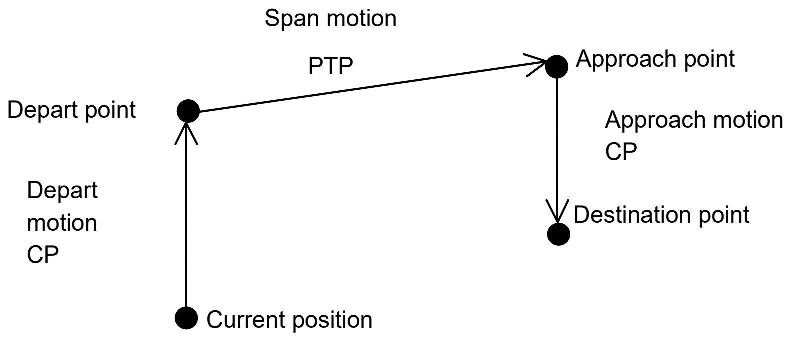
destination	The target destination of the motion using a point expression.
TLZ movement	The amount of movement in Z direction in Tool coordinate system. The is unit is [mm]. The Tool coordinate system for the currently used Tool number is used.
archNumber	Optional. The arch number (*archNumber*) specifies which Arch Table entry to use for the Arch type motion caused by the JumpTLZ instruction. *archNumber* must always be proceeded by the letter C. (Valid entries are C0 to C7.)
CP	Optional. Specifies continuous path motion.
LJM	Optional. Convert the target destination using LJM function.
orientationFlag	Optional. Specifies a parameter that selects an orientation flag for LJM function.

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**}

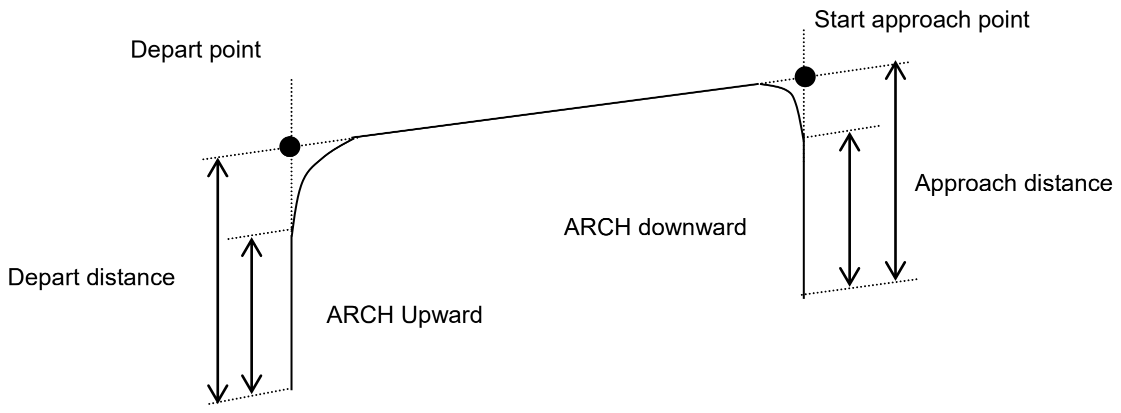
!...!	Optional. Parallel Processing statements can be added to the Jump3 and Jump3CP instruction to cause I/O and other commands to execute during motion.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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.
The depart point is a point moved from the current position with *TLZ amount* in the Z direction.
The robot posture at the depart point is same as the current position. (Posture may change if the robot passes the singularity or singularity neighborhood.)
The approach point is a point moved from the depart point in X and Y direction of the Tool coordinate system with move amount to approach to the destination point. The U, V, and W coordinates and the robot posture at the depart point and are the same as the current position. (Posture may change if the robot passes the singularity or singularity neighborhood)



Arch motion is achieved by specifying the arch number. 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.



Notes

LimZ does not affect JumpTLZ

LimZ has no effect on JumpTLZ since the span motion is not necessarily perpendicular to the Z axis of the coordinate system.

JumpTLZ span motion is PTP (point to point)

It is difficult to predict JumpTLZ 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.

Difference between JumpTLZ and Jump3

JumpTLZ and Jump3 are different in the following points.

JumpTLZ:

The depart point must be in the Z direction from the current position.

The approach point must be in the Z direction from the destination point. Also, the approach distance cannot be specified.

Different Tool coordinate systems cannot be selected for the depart, approach, and destination points.

(It is not possible to execute the depart motion in Tool1, and execute the approach motion in Tool2.)

Jump3:

The depart point can be anywhere.

The approach point can be anywhere.

Different Tool coordinate systems can be selected for the depart, approach, and destination points.

(It is possible to execute the depart motion in Tool1, and execute the approach motion in Tool2.)

Applicable manipulators

JumpTLZ is only available for N series.

Caution for Arch motion

JumpTLZ motion trajectory is comprised of depart, span, and approach motions. It is not a continuous path trajectory. The actual JumpTLZ trajectory of arch motion is not determined by Arch parameters alone. It also depends on motion and speed.

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

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

In a JumpTLZ 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 JumpTLZ commands with the same distance and speed are executed, the trajectory is affected by motion of the robot arms.

Potential acceleration errors

When the majority of depart (approach) motion uses the same joint as the span motion

An acceleration error may occur during an arch motion execution by the JumpTLZ command. 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 JumpTLZ. 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.

See Also

Accel, Arc, Arch, Go, JS, JT, Point Expression, Pulse, Sense, Speed, Stat, Till

JumpTLZ Example

Move 100 mm upward from the current point in Z direction of the Tool coordinate system. Then, move to the target point (P0):

```
JumpTLZ  P0, -100
```

LatchEnable Statement

Enables / Disables the latch function for the robot position by the R-I/O input.

Syntax

LatchEnable { On | Off }

Parameters

On | Off	On : Enables the latch function of the robot position. Off : Disables the latch function of the robot position.

Result

When the parameter is omitted, displays that the current latch function is ON or OFF.

Description

Enables / Disables the latch function for the robot position using the trigger input signals connected to the R-I/O. It latches the robot position with the first trigger input after you enable the latch function.
To repeatedly latch the robot position, execute LatchEnable Off and then execute LatchEnable On again. To use the command repeatedly, it needs at least 60 ms interval for the each command processing time but you do not need to consider the command executing time.

Note

Before enabling the latch function, set the trigger input port and trigger signal logic using SetLatch.

See Also

LatchPos Function, LatchState Function, SetLatch

LatchEnable Statement Example

```
Function main
      SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
      LatchEnable On            'Enables the latch function
      Go P1
      Wait LatchState = True    'Waits a trigger
      Print LatchPos            'Displays the latched position
      LatchEnable Off           'Disables the latch function
Fend
```

LatchState Function

Returns the latch state of robot position using the R-I/O.

Syntax

LatchState

Return Values

Returns True when the robot position has been latched, False when the latch is not finished.
When confirmed the latch completion, acquires the latched position information by LatchPos Function.
When specified continuous latch times with SetLatch, returns “True” if specified latch times all complete.

See Also

LatchEnable, LatchPos Function, SetLatch, Wait

LatchState Function Example

```
Function main
      SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
      LatchEnable On              'Enables the latch function
      Go P1
      Wait LatchState = True      'Wait a trigger
      Print LatchPos              'Display the latched position
      LatchEnable Off             'Disable the latch function
Fend
```

LatchPos Function

Returns the robot position latched using the R-I/O input signal.

Syntax

LatchPos ([WithToolArm | WithoutToolArm], Latch Number)

Parameters

WithToolArm | WithoutToolArm	Specifies whether to return positional data based on Tool and Arm settings, or to return the Tool 0, Arm 0 position. While optional, set this parameter when specifying a latch number. If omitted, WithToolArm is used.

Constant	Value
WithToolArm	0
WithoutToolArm	1

WithToolArm	Constant value of 0. Returns the position according to the Tool and Arm settings at function call.
WithoutToolArm	Constant value of 0.1 Returns the position of Tool 0 and Arm 0, regardless of the Tool and Arm settings.
Latch number	Specifies the R-I/O input signal used to return latched point data after setting LatchEnable On. Specify either 1, 2, 3, or 4. Specify the continuous latch count with SetLatch to enable a latch with point data produced from up to four R-I/O input signals after setting LatchEnable On. This parameter is optional. If omitted, point data latched using the first R-I/O input signal will be returned.

Return Values

Returns the robot position latched by the R-I/O input signal in point data.
Executing this function needs approx. 15 msec for processing.
When WithToolArm is specified, returns the position according to the Tool and Arm settings at function call.
When WithoutToolArm is specified, returns the position of Tool 0 and Arm 0, regardless of the Tool and Arm settings.

See Also

LatchEnable, LatchState Function, SetLatch

LatchPos Function Example

```
Function main
      SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
      LatchEnable On             'Enables the latch function
      Go P1
      Wait LatchState = True     'Wait a trigger
      Print LatchPos(WithoutToolArm, 1)      'Display the latched position 1
      Print LatchPos(WithoutToolArm, 2)      'Display the latched position 2
      Print LatchPos(WithoutToolArm, 3)      'Display the latched position 3
      Print LatchPos(WithoutToolArm, 4)      'Display the latched position 4
      LatchEnable Off            'Disable the latch function
Fend
```

To omit parameter:

```
Function main
     SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
     LatchEnable On                 ' Enables the latch function
     Go P1
     Wait LatchState = True         ' Wait a trigger
     Print LatchPos                 ' Display the latched position 1
     LatchEnable Off                ' Disable the latch function
Fend
```

To assign the return value of LatchPos to the point data:

```
P2 = LatchPos                      ' Assign the latched position 1. Omit parameter.

P2 = LatchPos(WithoutToolArm, 1)    'Assign the latched position 1
P3 = LatchPos(WithoutToolArm, 2)    'Assign the latched position 2
P4 = LatchPos(WithoutToolArm, 3)    'Assign the latched position 3
P5 = LatchPos(WithoutToolArm, 4)    'Assign the latched position 4
```

LCase$ Function

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

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 Command window.

```
> Print Left$("ABCDEFG", 2)
 AB

> Print Left$("ABC", 3)
 ABC
```

Len Function

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* 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 and 255 (since a string can contain between 0 and 255 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 command window.

```
> ? len("ABCDEFG")
7

> ? len("ABC")
3

> ? len("")
0
>
```

LimitTorque Statement

Sets / returns the upper limit torque value in High power mode.

Syntax

(1) **LimitTorque** *AllMax*
(2) **LimitTorque** *j1Max, j2Max, j3Max, j4Max*
(3) **LimitTorque** *j1Max, j2Max, j3Max, j4Max, j5Max, j6Max*
(4) **LimitTorque**

Parameters

AllMax	Specify the upper limit torque value for all axes in high power mode by an integer number representing the percentage of the maximum momentary torque of each axis
j #n Max	Specify the upper limit torque value for axis #n in high power mode by an integer number representing the percentage of the maximum momentary torque of axis #n

Result

Returns the current LimitTorque value when the parameters are omitted.

Description

Sets the upper limit value of torque in high power mode. Normally, the maximum torque is set and there is no need to change this setting value. This statement is useful to restrict the torque not to exceed which is necessary for the specific motion in order to reduce damage to the manipulator and equipment caused by collision with peripherals.
The upper limit value is a peak torque in specific motion measured by PTRQ with allowance considering the variation added (approximately 10%).

The torque lower than the upper limit value in Low power mode cannot be set for this command. The minimum values vary for models and joints. Obtain the setting value and confirm the actual upper limit value after setting the value.

In any of the following cases, LimitTorque becomes the default value.

Controller startup
Motor On
SFree, SLock, or Brake is executed
Reset or Reset Error is executed
Task end by STOP switch or Quit All

Note

Too low LimitTorque setting

LimitTorque restricts the torque for the specific motion with the set torque restriction value as the upper limit value, regardless of the size of torque necessary for the motion to be executed with the set acceleration/deceleration. Therefore, if the motion requires larger torque than the set upper limit value, the robot may not move properly and cause vibrational motion, noise, or position deviation error and overrun. Make sure to measure PTRQ before using the torque restriction function. If the above problems occur, set the upper limit value larger and adjust the value so that the manipulator can operate properly.

See Also

LimitTorque Function, Power, PTrq, RealTorque

LimitTorque Statement Example

Following is the example which operates the manipulator with the maximum torque of the Joint #1 at 80%.

```
Function main
    Motor On
    Power high
    Speed 100; Accel 100,100
    LimitTorque 80,100,100,100      'Restricts the maximum torque of Joint #1 to 80%
    Jump P1                         'Executes the Jump motion
Fend
```

LimitTorque Function

Returns the setting value of LimitTorque command.

Syntax

LimitTorque*(jointNumber)*

Parameters

jointNumber	Integer expression ranging from 1 to 9. Additional S axis is 8, and T axis is 9.

Return Values

Returns an integer number representing the setting value of LimitTorque command.

See Also

LimitTorque

LimitTorque Function Example

```
Print LimitTorque(1)  'Displays the LimitTorque value of the Joint #1.
```

LimitTorqueLP Statement

Sets / returns the upper limit torque value in Low power mode.

Syntax

(1) **LimitTorqueLP** *AllMax*
(2) **LimitTorqueLP** *j1Max, j2Max, j3Max, j4Max*
(3) **LimitTorqueLP** *j1Max, j2Max, j3Max, j4Max, j5Max, j6Max*
(4) **LimitTorqueLP**

Parameters

AllMax	Specify the upper limit torque value for all axes in low power mode by an integer number representing the percentage of the maximum momentary torque of each axis
j #n Max	Specify the upper limit torque value for axis #n in low power mode by an integer number representing the percentage of the maximum momentary torque of axis #n

Result

Returns the current LimitTorqueLP value when the parameters are omitted.
Displays the default value when the values are not changed by this command.

Description

Sets the upper limit value of torque in low power mode. Normally, the maximum torque is set and there is no need to change this setting value (the values vary depending on the robot models and axes. Approx. 15-60%). This command is useful to restrict the torque not to exceed which is necessary for the normal motion in order to reduce damage to the manipulator and equipment caused by collision with peripherals.
The upper limit value is a peak torque in the motion measured by PTRQ with allowance considering the variation added (40% is recommended). To apply the same value to a different robot, add a further 10-20% allowance.
The PTRQ value considers the default maximum torque in low power mode as 1.0. For example, when the default value before change is 27% and the value measured by PTRQ is 0.43, the upper limit value is as follows: 27% × 0.43 × 1.4 = 16.25. Then, round up the value and set 17.
The value lower than 5% or larger than the default value cannot be set for this command. If these values are set, the setting values lower than 5 will be rounded up to 5, and the values exceeding the default value will be rounded down to the default. For instance, when "LimitTorqueLP 100", the values are returned to the default for all joints because the default value is always less than 100.
Obtain the setting value and confirm the actual upper limit value after setting the value.
The LimitTorqueLP setting value is effective until the Controller is restarted.

Note

Too low LimitTorqueLP setting

LimitTorqueLP restricts the torque for the specific motion with the set torque restriction value as the upper limit value, regardless of the size of torque necessary for the motion to be executed with the set acceleration/deceleration. Therefore, if the motion requires larger torque than the set upper limit value, the robot may not move properly and cause position deviation error. Make sure to measure PTRQ before using the torque restriction function. If the above problem occurs, set the upper limit value larger and adjust the value so that the manipulator can operate properly.

See Also

LimitTorqueLP Function, PTRQ

LimitTorqueLP Example

Following is the example which operates the manipulator with the maximum torque of the Joint #1 at 10%.

```
Function main
Motor On
Power low
LimitTorqueLP 10,27,31,42 ' Restricts the maximum torque of the Joint #1 to 10%
                          ' Set the default value for other axes
Go P1                     ' Executes the Go motion
Fend
```

LimitTorqueLP Function

Returns the setting value of LimitTorqueLP command.

Syntax

LimitTorqueLP*(jointNumber)*

Parameters

jointNumber	Integer expression ranging from 1 to 9. Additional S axis is 8, and T axis is 9.

Return Values

Returns an integer number representing the setting value of LimitTorqueLP command.

See Also

LimitTorqueLP

Len Function Example

```
Print LimitTorqueLP(1) 'Displays the LimitTorqueLP value of the Joint #1.
```

LimitTorqueStop Statement

Enables or disables the function to stop the robot when the upper limit torque is reached in High power mode.

Syntax

(1) LimitTorqueStop *status*
(2) LimitTorqueStop *status*, *jointNumber*
(3) LimitTorqueStop

Parameters

status	On: Enables the function to stop the robot at the upper limit torque
	Off: Disables the function to stop the robot at the upper limit torque
jointNumber	The joint number from 1 to 6. (For SCARA robots, joint numbers are from 1 to 4)

Result

Returns the current LimitTorqueStop status if the parameter is omitted.

Description

LimitTorqueStop enables the function to stop the robot at the upper limit torque value in High power mode. The robot immediately stops when it reaches the upper limit torque (default is 100%). Using this command together with the torque restriction function of LimitTorque provides the effect to reduce damage on the robot and peripherals at a collision in High power mode.
This function can be enabled or disabled for each joint or all joints. The default is “all joints off”.
The setting returns to the default at the Controller startup. In other cases, the setting does not change unless otherwise configured by this command explicitly.
When the upper limit torque is reached, Error 5040 “Motor torque output failure in high power state.” will be output and the robot will stop.

See Also

LimitTorque, LimitTorque Function

LimitTorqueStop Example

Following is the example which restricts the maximum torque of the Joint #1 at 30% and stops the robot immediately.

```
Function main
Motor On
Power high
Speed 20
Accel 20,20
LimitTorque 30,100,100,100    ' Restricts the maximum torque of the Joint #1 to 30%
LimitTorqueStop On, 1         ' Joint #1 immediately stops at the maximum torque
Go P1                         ' Executes the Go motion
Fend
```

LimitTorqueStop Function

Returns the setting value of LimitTorqueStop command.

Syntax

LimitTorqueStop*(jointNumber)*

Parameters

jointNumber	Integer expression ranging from 1 to 6.

Return Values

Returns an integer number representing the setting value of LimitTorqueStop command.
0 = Off
1 = On

See Also

LimitTorqueStop

LimitTorqueStop Function Example

```
Print LimitTorqueStop(1) 'Displays the LimitTorqueStop value of the Joint #1.
```

LimitTorqueStopLP Statement

Enables or disables the function to stop the robot when the upper limit torque is reached in Low power mode.

Syntax

(1) LimitTorqueStopLP *status*
(2) LimitTorqueStopLP *status*, *jointNumber*
(3) LimitTorqueStopLP

Parameters

status	On: Enables the function to stop the robot at the upper limit torque Off: Disables the function to stop the robot at the upper limit torque
jointNumber	The joint number from 1 to 6. (For SCARA robots, joint numbers are from 1 to 4)

Result

Returns the current LimitTorqueStopLP status if the parameter is omitted.

Description

LimitTorqueStopLP enables the function to stop the robot at the upper limit torque value in Low power mode. The robot immediately stops when it reaches the upper limit torque. Using this command together with the torque restriction function of LimitTorqueLP provides the effect to reduce damage on the robot and peripherals at a collision in Low power mode.
This function can be enabled or disabled for each joint or all joints. The default is “all joints off”.
The setting returns to the default at the Controller startup. In other cases, the setting does not change unless otherwise configured by this command explicitly.
When the upper limit torque is reached, Error 5041 “Motor torque output failure in low power state.” will be output and the robot will stop.

See Also

LimitTorqueLP, LimitTorqueLP Function

LimitTorqueStopLP Example

Following is the example which restricts the maximum torque of the Joint #3 at 15% and stops the robot immediately.

```
Function main
Motor On
Power low
LimitTorqueLP 20,27,15,42       ' Restricts the maximum torque of the Joint #3 to 15%
                                ' Set the default value for other axes
LimitTorqueStopLP On, 3         ' Joint #3 immediately stops at the maximum torque
Go P1                           ' Executes the Go motion
Fend
```

LimitTorqueStopLP Function

Returns the setting value of LimitTorqueStopLP command.

Syntax

LimitTorqueStopLP*(jointNumber)*

Parameters

jointNumber	Integer expression ranging from 1 to 6.

Return Values

Returns an integer number representing the setting value of LimitTorqueStopLP command.
0 = Off
1 = On

See Also

LimitTorqueStopLP

LimitTorqueStopLP Function Example

```
Print LimitTorqueStopLP(3)  'Displays the LimitTorqueStopLP value of the Joint #3.
```

LimZ Statement

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.

Notes

Resetting LimZ to 0

Restarting the controller, or executing the SFree, SLock, Motor On commands will 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 effect 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

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
```

LimZMargin Statement

Sets and returns the setting value for error detection when operation starts at higher than the LimZ value.

Syntax

(1) **LimZMargin** *LimZmargin*
(2) **LimZMargin**

Parameters

LimZmargin	a margin value for LimZ error detection

Return Values

If the parameter is omitted, current LimZMargin value will be returned.

Description

When Jump command is executed, Joint #3 lifts up to the position set by LimZ. However, if the start position of the joint is above the LimZ position, an error will occur. LimZMargin sets a margin value for the error detection. Default is 0.02 mm.

Note

Resetting LimZ to default

Restarting the controller, or executing the SFree, SLock, Motor On commands will initialize LimZ to the default value.

See Also

LimZMargin Function, LimZ

LimZ Statement Example

Following is a usage example of LimZMargin in Jump operation.

```
Function main
    LimZ -10             'sets LimZ default value
    LimZMargin 0.03      'sets 0.03 mm for a margin of LimZ error detection
    Jump P1              'horizontal movement with -10 at Jump execution
    Jump P2 LimZ -20     'horizontal movement with -20 at Jump execution
    Jump P3              'horizontal movement with -10 at Jump execution
Fend
```

LimZMargin Function

Returns the current LimZMargin setting.

Syntax

LimZMargin

Return Values

Real number containing the current LimZMargin setting.

See Also

LimZMargin Statement, LimZ Statement

LimZ Function Example

```
Real savLimzMargin

savLimzZMargin = LimZMargin
LimZMargin 0.03
Jump pick
LimZ savLimZMargin
```

Line Input Statement

Reads input data of one line and assigns the data to a string variable.

Syntax

Line Input *stringVar$*

Parameters

stringVar$	A string variable name. (the string variable must end with the $ character.)

Description

Line Input reads input data of one line from the display 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 display 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 Statement 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

Reads data of one line from a file, communication port, database, or the device.

Syntax

Line Input *#portNumber, stringVar$*

Parameters

portNumber	ID number representing a file, communications port, database, or device. File number can be specified in ROpen, WOpen, and AOpen statements. Communications port number can be specified in OpenCom (RS232) and OpenNet (TCP/IP) statements. Database number can be specified in OpenDB statement. Device ID integers are as follows. 21 RC+ 23 OP 24 TP (TP1 only) 20 TP3
stringVar$	A string variable. (string variables must end with a $ character.)

Description

Line Input # reads string data of one line from the device specified with the portNumber parameter, and assigns the data to the string variable *stringVar$*.

Notes

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

See Also

Input, Input #, Line Input

Line Input # Statement 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
```

LJM Function

Returns the point data with the orientation flags converted to enable least joint motion when moving to a specified point based on the reference point.

Syntax

LJM (*Point* [, *refPoint* [, *orientationFlag*]])

Parameters

Point	Specifies point data.
refPoint	Specifies the reference point data. When this is omitted, the reference point is the current position (Here).

orientationFlag

6-axis robot

1: Converts the wrist orientation (Wrist Flag), J4Flag, J6Flag or J1Flag so that Joint #4 will be the shortest movement. This is the default setting when "*orientationFlag"* is omitted.

2: Converts the J4Flag or J6Flag.

3: Converts the wrist orientation (Wrist Flag), J4Flag, J6Flag or J1Flag so that Joint #5 will be the shortest movement.

4: Converts the wrist orientation (Wrist Flag), J4Flag, J6Flag or J1Flag so that Joint #6 will be the shortest movement.

"orientation Flag"	Hand orientation	Elbow orientation	Wrist orientation	J1Flag	J4Flag	J6Flag	Priority order of axis with the shortest movement
1	-	-	✓	✓	✓	✓	J4
2	-	-	-	✓	✓	✓	-
3	-	-	✓	✓	✓	✓	J5
4	-	-	✓	✓	✓	✓	J6

Note: Orientation of "-" is the same as the orientation specified by "*refPoint*".

RS series

1: Converts the hand orientation (Hand Flag), J1Flag or J2Flag. This is the default setting when "*orientationFlag"* is omitted.

2: Converts the hand orientation (Hand Flag), J1Flag or J2Flag. Prevents the U axis from moving out of motion range at "*orientationFlag*" convert.

N2 series

1: Converts to the posture with minimum joint movement in priority order of Joint #1 and Joint #5. The target postures are hand orientation (Hand Flag), elbow orientation (Elbow Flag), wrist orientation (Wrist Flag), J4Flag, and J6Flag. The elbow orientation (Elbow Flag) is always above elbow orientation. This is the default setting when "*orientationFlag"* is omitted.

2: Converts to the posture with minimum joint movement in priority order of Joint #1 and Joint #4. The target postures are hand orientation (Hand Flag), elbow orientation (Elbow Flag), wrist orientation (Wrist Flag), J4Flag, and J6Flag. The elbow orientation (Elbow Flag) is always above elbow orientation.

3: Converts the wrist orientation (Wrist Flag), J4Flag, and J6Flag so that Joint #4 will be the shortest movement.

4: Converts the J4Flag and J6Flag.

5: Change the hand orientation specified by "*refPoint*" to different hand orientation (Hand Flag). Converts the wrist orientation (Wrist Flag), J4Flag, J6Flag or J1Flag so that Joint #5 will be the shortest movement. The target postures are hand orientation (Hand Flag), elbow orientation (Elbow Flag), wrist orientation (Wrist Flag), J4Flag, and J6Flag. The elbow orientation (Elbow Flag) is always above elbow orientation.

6: Change the hand orientation specified by "*refPoint*" to different hand orientation (Hand Flag). Converts the wrist orientation (Wrist Flag), J4Flag, J6Flag or J1Flag so that Joint #4 will be the shortest movement. The target postures are hand orientation (Hand Flag), elbow orientation (Elbow Flag), wrist orientation (Wrist Flag), J4Flag, and J6Flag. The elbow orientation (Elbow Flag) is always above elbow orientation.

7: Change the elbow orientation to the below elbow orientation (Elbow Flag). To be the shortest movement, converts the wrist orientation (Wrist Flag), J4Flag, and J6Flag in priority order of Joint #1 and Joint #5. The target postures are hand orientation (Hand Flag), elbow orientation (Elbow Flag), wrist orientation (Wrist Flag), J4Flag, and J6Flag.

8: Change the elbow orientation to the below elbow orientation (Elbow Flag). To be the shortest movement, converts the wrist orientation (Wrist Flag), J4Flag, and J6Flag in priority order of Joint #1 and Joint #4. The target postures are hand orientation (Hand Flag), elbow orientation (Elbow Flag), wrist orientation (Wrist Flag), J4Flag, and J6Flag.

"orientation Flag"	Hand orientation	Elbow orientation	Wrist orientation	J4Flag	J6Flag	Priority order of axis with the shortest movement
1	✓	*1	✓	✓	✓	J1>J5
2	✓	*1	✓	✓	✓	J1>J4
3	-	-	✓	✓	✓	J4
4	-	-	-	✓	✓	-
5	*2	*1	✓	✓	✓	J5
6	*2	*1	✓	✓	✓	J4
7	✓	*3	✓	✓	✓	J1>J5
8	✓	*3	✓	✓	✓	J1>J4

Note: Orientation of "-" is the same as the orientation specified by "*refPoint*".

*1: Above elbow orientation

*2: Hand orientation is different from the orientation specified by "*refPoint*".

*3: Below elbow orientation

N6 series

1: Converts the wrist orientation (Wrist Flag), J4Flag, and J6Flag so that Joint #4 will be the shortest movement. This is the default setting when "*orientationFlag*" is omitted.

2: Converts the J4Flag and J6Flag.

3: Converts the wrist orientation (Wrist Flag), J4Flag, J6Flag or J1Flag so that Joint #5 will be the shortest movement.

4: Converts the wrist orientation (Wrist Flag), J4Flag, J6Flag or J1Flag so that Joint #6 will be the shortest movement.

"orientation Flag"	Hand orientation	Elbow orientation	Wrist orientation	J1Flag	J4Flag	J6Flag	Priority order of axis with the shortest movement
1	-	-	✓	✓	✓	✓	J4
2	-	-	-	✓	✓	✓	-
3	-	-	✓	✓	✓	✓	J5
4	-	-	✓	✓	✓	✓	J6

Note: Orientation of "-" is the same as the orientation specified by "*refPoint*".

Description

When the 6-axis or N series robot moves to a point calculated by such as pallet or relative offsets, the wrist part may rotate to an unintended direction. The point calculation above does not depend on robot models and results in motion without converting the required point flag.
LJM function can be used to convert the point flag to prevent the unintended wrist rotation.

For the N series robots, it is also possible to reduce the cycle time and omit teaching of the avoidance point, which is necessary for the 6-axis robots, by changing the Hand Flag and Elbow Flag.

When the RS series robot moves to a point calculated by such as pallet or relative offsets, Arm #1 may rotate to an unintended direction. LJM function can be used to convert the point flag to prevent the unintended rotation of Arm #1.

In addition, the U axis of an RS series robot may go out of motion range when the orientation flag is converted, which will cause an error.
To prevent this error, the LJM function adjusts the U axis target angle so that it is inside the motion range. This is available when “2” is selected for *orientationFlag*.

Returns the specified point for all robots except the 6-axis, N series, and RS series robot.

Note

The reference point omission and Parallel Processing

You cannot use both of the parallel point omission and parallel processing in one motion command like this:

```
    Go LJM(P10)  ! D10; MemOn 1 !
```

Be sure to change the program like this:

```
    P999 = Here
    Go LJM(P10,P999) ! D10; MemOn 1 !
```

orientationFlag for N2 series

- *orientationFlag* 1, 2:
  To shorten the cycle time, select *orientationFlag* 1 or 2.
  Since the posture has minimum Joint #1 movement, the cycle time can be shortest in most motion.
  To reduce the Joint #5 movement, select *orientationFlag* 1.
  To reduce the Joint #4 movement, select *orientationFlag* 2.
- *orientationFlag* 3, 4:
  Use these flags if you do not want to change the reference orientation, hand orientation, and elbow orientation.
  Use these flags if you want to use them in a same manner as the flags for vertical 6-axis robots.
  orientationFlag 3 is same as *orientationFlag* 1 of the vertical 6-axis robots.
  orientationFlag 4 is same as *orientationFlag* 2 of the vertical 6-axis robots.
- *orientationFlag* 5, 6:
  If the hand collides with peripheral walls during the operation, select *orientationFlag* 5 or 6.
  Since the hand passes the neighborhood of the robot’s origin point, the robot can move with less possibility to collide with the obstacles.
  To reduce the Joint #5 movement, select *orientationFlag* 5.
  To reduce the Joint #4 movement, select *orientationFlag* 6.
- *orientationFlag* 7, 8:
  To have a below elbow orientation, select *orientationFlag* 7 or 8.
  Depending on motion, the robot passes the neighborhood of the origin like *orientationFlag* 5 and *orientationFlag* 6. Therefore, the robot can move with less possibility to collide with the obstacles, if these are located around the robot.
  To reduce the Joint #5 movement, select *orientationFlag* 7.
  To reduce the Joint #4 movement, select *orientationFlag* 8.

localNumber

Local numbers of the points returned by LJM function are the same as that of “Point Expression”.

See Also

Pallet

LJM Function Example

```
Function main
   Integer i, j

   P0 = XY(300, 300, 300, 90, 0, 180)
   P1 = XY(200, 280, 150, 90, 0, 180)
   P2 = XY(200, 330, 150, 90, 0, 180)
   P3 = XY(-200, 280, 150, 90, 0, 180)

   Pallet 1, P1, P2, P3, 10, 10

   Motor On
   Power High
   Speed 50; Accel 50, 50
   SpeedS 1000; AccelS 5000

   Go P0
   P11 = P0 -TLZ(50)

   For i = 1 To 10
         For j = 1 To 10
            'Specify points
            P10 = P11                   'Depart point
         P12 = Pallet(1, i, j)          'Target point
         P11 = P12 -TLZ(50)             'Start approach point
         'Converting each point to LJM
         P10 = LJM(P10)
         P11 = LJM(P11, P10)
         P12 = LJM(P12, P11)
         'Execute motion
            Jump3 P10, P11, P12 C0
      Next
   Next
Fend

Function main2
   P0 = XY(300, 300, 300, 90, 0, 180)
   P1 = XY(400, 0, 150, 90, 0, 180)
   P2 = XY(400, 500, 150, 90, 0, 180)
   P3 = XY(-400, 0, 150, 90, 0, 180)
   Pallet 1, P1, P2, P3, 10, 10

   Motor On
   Power High
   Speed 50; Accel 50, 50
   SpeedS 1000; AccelS 5000

   Go P0

   Do
      ' Specify points
      P10 = Here -TLZ(50)                                    'Depart point
      P12 = Pallet(1, Int(Rnd(9)) + 1, Int(Rnd(9)) + 1)      'Target point
      P11 = P12 -TLZ(50)                                     'Start approach point

      If TargetOK(P11) And TargetOK(P12) Then                'Point check
      ' Converting each point to LJM
      P10 = LJM(P10)
      P11 = LJM(P11, P10)
      P12 = LJM(P12, P11)
      'Execute motion
         Jump3 P10, P11, P12 C0
      EndIf
    Loop
Fend
```

LoadPoints Statement

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 .PTS. The file must exist in the current project for the current robot. You cannot specify a file path and fileName doesn't have any effect from ChDisk. See ChDisk for the details.
Merge	Optional. If supplied, then the current points are not cleared before loading the new points. Points in the file are added 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 for the current robot.

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 from 0 to 100. Then use Merge to load other point files for each part being run without clearing the common points. The range could be from 101 to 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 occur.

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

ImportPoints, Robot, SavePoints

LoadPoints Statement Example

```
Function main
  ' Load common points for the current robot
  LoadPoints "R1Common.pts"

  ' Merge points for part model 1
  LoadPoints "R1Model1.pts", Merge

  Robot 2
  ' Load point file for the robot 2
  LoadPoints "R2Model1.pts"

Fend
```

Local Statement

Defines and displays local coordinate systems.

Syntax

(1) **Local** *localNumber,* (*pLocal1 : pBase1*), (*pLocal2 : pBase2*) [, { **L** | **R** }] [, **BaseU**]
(2) **Local** *localNumber, pCoordinateData*
(3) **Local** *localNumber, pOrigin,* [*pXaxis*], [*pYaxis*], [{ **X** | **Y** }]
(4) **Local** *localNumber*

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.
pCoordinateData	Point data representing the coordinate data of the origin and direction. When it's SCARA Robots (include RS series), specify V coordinate and W coordinate to "0".
pOrigin	Integer expression representing the origin point using robot coordinate system.
pXaxis	Optional. Integer expression representing a point along the X axis using robot coordinate system if X alignment is specified.
pYaxis	Optional. Integer expression representing a point along the Y axis using robot coordinate system if Y alignment is specified.
X | Y	If X alignment is specified, then pXaxis lies on the X axis of the local. The Y axis and Z axis are calculated to be orthogonal to X in the plane that is created by the 3 local points. If Y alignment is specified, then pYaxis lies on the Y axis of the local. The X axis and Z axis are calculated to be orthogonal to Y in the plane that is created by the 3 local points.

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
```

(4) Displays the specified local settings.

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.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Note

When it's SCARA Robots, do not set for V and W.

When you use SCARA Robot, not to set value to V and W coordinate in base coordinate or input "0". If you set them, an error may occur out of the J4.

See Also

ArmSet, Base, ECPSet, LocalClr, TLSet, Where

Local Statement Examples

Here are some examples from the command 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, 0
```

Local defined with only the origin point rotated 45 degrees about the X axis:

```
> local 2, 50, 200, 0, 0, 45, 0
```

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

Returns the specified local coordinate system data as a point.

Syntax

Local(*localNumber*)

Parameters

localNumber	local coordinate system number (integer from 1 to 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

Clears (undefines) a local coordinate system.

Syntax

LocalClr *localNumber*

Parameters

localNumber	Integer expression representing which of 15 locals (integer from 1 to 15) to clear (undefine).

Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

Arm, ArmSet, ECPSet, Local, Tool, TLClr, TLSet

LocalClr Statement Example

```
    LocalClr 1
```

LocalDef Function

Returns local definition status.

Syntax

LocalDef (*localCoordinateNumber*)

Parameters

localCoordinateNumber Integer expression representing which local coordinate to return status for.

Return Values

True if the specified local has been defined, otherwise False.

See Also

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

LocalDef Function Example

```
Function DisplayLocalDef(localNum As Integer)

   If LocalDef(localNum) = False Then
      Print "Local ", localNum, "is not defined"
   Else
      Print "Local 1: ",
      Print Local(localNum)
   EndIf
Fend
```

Lof Function

Checks whether the specified RS-232 or TCP/IP port has any lines of data in its buffer.

Syntax

Lof (*fileNumber* As Integer)

Parameters

fileNumber	A number specified with OpenCom (RS-232C) or OpenNet (TCP/IP) statement.

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.
You can wait for the return value of Lof function by executing Wait.

Note

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

When using PC COM port (1001 to 1008), you cannot use Lof function with Wait command.

See Also

ChkCom, ChkNet, Input#, Wait

Lof Function Example

This Command window example prints out the number of lines of data received through the communication port number 1.

```
>print lof(1)
 5
>
```

LogIn Statement

Log into EPSON RC+ 6.0 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 in Auto Mode, 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 active.

See Also

GetCurrentUser$ Function

LogIn Statement Example

```
Integer errCode
errCode = LogIn("operator", "oprpass")
```

Long Statement

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 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

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, Int32, Int64, Integer, Real, Short, String, UByte, UInt32, UInt64, UShort

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(5, 5, 5)     '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
```

LSet$ Function

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

```
temp$ = "123"
temp$ = LSet$(temp$, 10)  ' temp$ = "123       "
```

LShift Function

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 (integer from 0 to 31) 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, Double, Int32, Integer, Long, Real, Short, UByte, UInt32, and UShort.

See Also

And, LShift64, Not, Or, RShift, RShift64, 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 command window.

```
> Print LShift(2,2)
8
> Print LShift(5,1)
10
> Print LShift(3,2)
12
>
```

LShift64 Function

Shifts numeric data to the left by a user specified number of bits.

Syntax

LShift64*(number, shiftBits)*

Parameters

number	Integer expression to be shifted.
shiftBits	The number of bits (integer from 0 to 63) 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

LShift64 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 LShift64 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. LShift64 works with data types: Int64 and UInt64.

See Also

And, LShift, Not, Or, RShift, RShift64, Xor

LShift64 Function Example

```
Function lshiftst
    Int64 i
    Int64 num, snum
    num = 1
    For i = 1 to 10
         Print "i =", i
         snum = LShift64(num, i)
         Print "The shifted num is ", snum
    Next i
Fend
```

Some other example results from the LShift64 instruction from the command window.

```
> Print LShift64(2,2)
8
> Print LShift64(5,1)
10
> Print LShift64(3,2)
12
>
```

LTrim$ Function

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

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

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 manipulator manual for details.

Potential Errors

Attempt to Execution a Motion command 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

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, Home, Hordr, Mcorg, MCordr

Mcal Statement Example

The following example is done from the monitor window:

```
> Motor On
> Mcal
>
```

MCalComplete Function

Returns status of MCal.

Syntax

MCalComplete

Return Values

True if MCal has been completed, otherwise False.

See Also

MCal

MCalComplete Function Example

```
If Not MCalComplete Then
    MCal
EndIf
```

MCordr Statement

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 [, Step5] [, Step6] [, Step7] [, Step8] [, Step9]*
(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)
Step5	Bit pattern that tells which axes should be calibrated during the 5th step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 5th step. (see below for bit pattern definitions)
Step6	Bit pattern that tells which axes should be calibrated during the 6th step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 6th step. (see below for bit pattern definitions)
Step7	Bit pattern that tells which axes should be calibrated during the 7th step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 7th step. (see below for bit pattern definitions)
Step8	Bit pattern that tells which axes should be calibrated during the 8th step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 8th step. (see below for bit pattern definitions)
Step9	Bit pattern that tells which axes should be calibrated during the 9th step of the Mcal process. Any number of axes between 0 to all 4 axes may calibrate during the 9th 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, 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 9 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 9 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:	&B000001	&B000010	&B000100	&B001000	

Joint:	5	6	7	8	9
Bit Number:	bit 4	bit 5	bit 6	bit 7	bit 8
Binary Code:	&B010000	&B100000	&B1000000	&B10000000	&B100000000

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

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 the robot is redefined.

See Also

Mcal

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

Returns an MCordr parameter setting.

Syntax

MCordr (*paramNumber*)

Parameters

paramNumber	Specifies reference setting numbers (integers from 1 to 9) 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

MCordr Function Example

This example uses the MCordr function in a program:

```
Integer a
a = MCordr(1)
```

MemIn Function

Returns the status of the specified memory I/O port. Each port contains 8 memory bits.

Syntax

MemIn(*portNumber***)**

Parameters

portNumber	Integer expression representing memory I/O bytes.

Return Values

Returns an integer value between 0 and 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 and 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 Values	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 Values	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

Note

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 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 and 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 Function 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 the 1st 8 memory I/O bit value
    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 Command 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
>
```

MemInW Function

Returns the status of the specified memory I/O word port. Each word port contains 16 memory I/O bits.

Syntax

MemInW(*WordPortNum*)

Parameters

WordPortNum	Integer expression representing the I/O word port.

Return Values

Returns the current status of the memory I/O (long integers from 0 to 65535).

See Also

MemIn, MemOut, MemOutW

MemInW Function Example

```
Long word0

word0 = MemInW(0)
```

MemOff Statement

Turns Off the specified bit of the memory I/O.

Syntax

MemOff { *bitNumber* | *memIOLabel* }

Parameters

bitNumber	Integer expression representing memory I/O bits.
memIOLabel	Memory I/O label.

Description

MemOff turns Off the specified bit of memory I/O. The 256 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 controller 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 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 command 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

Turns On the specified bit of the memory I/O.

Syntax

MemOn { *bitNumber* | *memIOLabel* }

Parameters

bitNumber	Integer expression representing memory I/O bits.
memIOLabel	Memory I/O label.

Description

MemOn turns on the specified bit of the robot memory I/O. The 256 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 controller 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

MemOn 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 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 command window are as follows:

```
> memon 1
> print memsw(1)
1
> memoff 1
> print memsw(1)
0
```

MemOut Statement

Simultaneously sets 8 memory I/O bits.

Syntax

MemOut *portNumber, outData*

Parameters

portNumber Integer expression representing memory I/O bit port number. The *portNumber* selection corresponds to the following:

Port number	Outputs
0	0-7
1	8-15
.	.

outData Integer expression between 0 and 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 *portNumber* parameter specifies which group of 8 outputs to use where *portNumber* = 0 means outputs 0 to 7, *portNumber* = 1 means outputs 8 to 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 and 255 and may be represented in hexadecimal or integer format. (i.e. &H0 to &HFF or 0 to 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 Statement 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 from 0 to 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
    MemOut 0, &H

    Wait 1
    MemOut 0, &H0
    Wait 1
  Loop
Fend
```

Other simple examples from the command 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
```

MemOutW Statement

Simultaneously sets 16 memory I/O bits.

Syntax

MemOutW *wordPortNum, outputData*

Parameters

wordPortNum	Integer expression representing memory I/O words.
outputData	Specifies output data (integers from 0 to 65535) using an expression or numeric value.

Description

Changes the current status of memory I/O port group specified by the word port number to the specified output data.

See Also

MemIn, MemInW, MemOut

MemOutW Statement Example

```
    MemOutW 0, 25
```

MemSw Function

Returns the status of the specified memory I/O bit.

Syntax

MemSw(*bitNumber***)**

Parameters

bitNumber Integer expression representing the memory I/O bit number.

Return Values

Returns "1" when the specified bit is On and "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. 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 Function 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 Command window are as follows:

```
> memon 1
> print memsw(1)
1
> memoff 1
> print memsw(1)
0
```

MHour Function

Returns the accumulated MOTOR ON time of the robot motors.

Syntax

MHour ([*robotNumber*])

Parameters

robotNumber	Specify the robot number to check the MOTOR ON time by an integer value. If omitted, currently selected robot will be used.

Return Values

Returns the accumulated MOTOR ON time of the motors by an integer value.

See Also

Time, Hour

MHour Function Example

```
Robot 2
Print MHour
Print MHour(1)
```

Mid$ Function

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 omitted, 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

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. See ChDisk for the details.

Description

Creates a subdirectory in the specified path. If omitted, a subdirectory is created in the current directory.

Note

- This statement is executable only with PC disk.

See Also

ChDir, ChDrive, RenDir, RmDir

MkDir Statement Example

The following examples are done from the command window:

```
> MkDir \Data

> MkDir \Data\PTS

> MkDir \TEST1 \TEST2
```

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 “0”. When the number is odd, the result of the Mod instruction will return “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 Command window.

```
> Print  36 Mod 6
> 0

> Print  25 Mod 10
> 5
>
```

Motor Statement

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.

In order to move the robot, motor power must be turned on.

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 sets the robot control parameter as below:

Speed, SpeedR, SpeedS	Default values
Accel, AccelR, AccelS	Default values
QPDecelR, QPDecelS	Default values
LimZ	0
CP	Off
SoftCP	Off
Fine	Default values
Power Low	Low
PTPBoost	Default values
TCLim, TCSpeed	Default values
PgLSpeed	Default values
PerformMode	Standard mode

See Also

Brake, Power, Reset, SFree, SLock

Motor Statement Example

The following examples are done from the command window:

```
> Motor On

> Motor Off
```

Motor Function

Returns status of motor power for the specified robot.

Syntax

Motor [(*robotNumber*)]

Parameters

robotNumber	Specify the robot number to check the status by an integer value. If omitted, currently selected robot will be used.

Return Values

0 = Motors off, 1 = Motors on.

See Also

Motor

Motor Function Example

```
If Motor = Off Then
    Motor On
EndIf
```

Move Statement

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*] [*!...!*] [*SYNC*]

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.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

Description

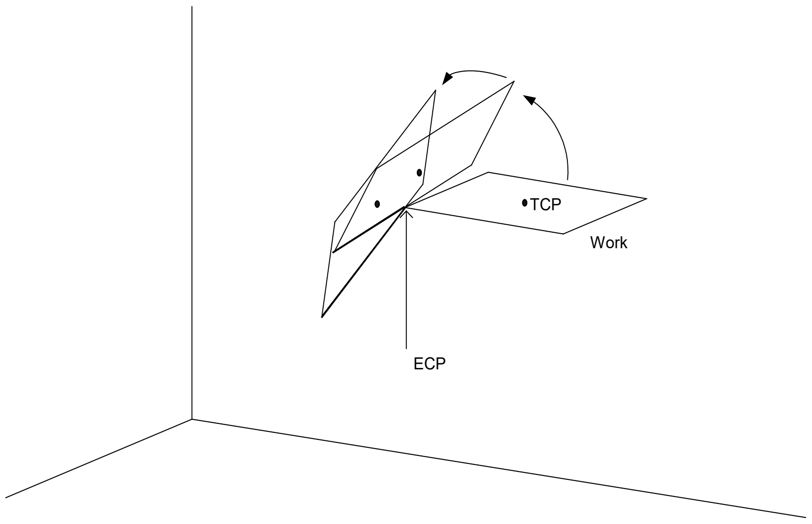
Move moves the arm from the current position to *destination* in a straight line. Move coordinates all axes to start and stop at the same time. The coordinates of *destination* must be taught previously before executing the Move instruction. Acceleration and deceleration for the Move is controlled by 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

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 Moves 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.*

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. In this case, use the ROT parameter.

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.

Attempt to Pass the Original Point (RS series)

It is impossible to operate the arm of RS series to pass near an original point. If attempted this, an overspeed error will occur. For the operation near an original point, take the following actions.

- Lower the speed of SpeedS
- Find a different path to prevent an original point
- Use PTP motion such as Go command instead of Move command.

See Also

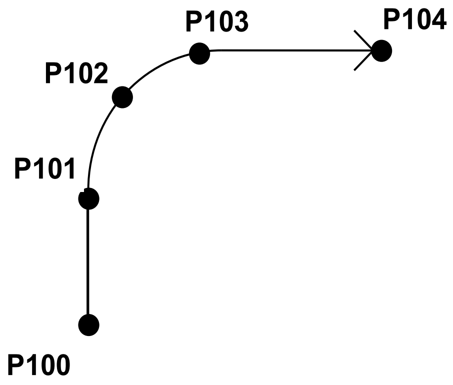
AccelS, Arc, CP, Go, Jump, Jump3, Jump3CP, SpeedS, Sw, Till

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 P0
  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 P101 CP        'Do not stop at P101
   Arc P102, P103 CP   'Do not stop at P103
   Move P104           'Decelerate to stop at P104
Fend
```

MsgBox Statement

Displays a message in a dialog box and waits for the operator to choose a button.

Syntax

MsgBox *message$* [, *type*] [, *title$*] [, *answer*]

Parameters

message$ 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+ 7.0 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 message box.

answer Optional. An integer variable that receives a value indicating the action taken by the operator. EPSON RC+ 6.0 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

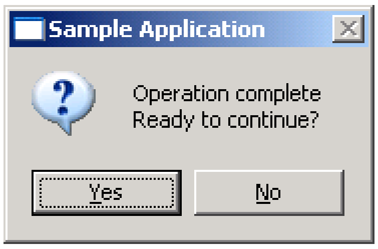
InputBox

MsgBox Statement 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 operator 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$ = Chr$(34) + "Operation complete" + Chr$(34) + 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.


Sample Application

Operation complete
Ready to continue?

Restriction

If the message$ and title$ of parameter contain a half-width comma ",", the string cannot be displayed correctly.
Use a string that does not contain a half-width comma.

MyTask Function

Returns the task number of the current program.

Syntax

MyTask

Return Values

The task number of the current task. Valid entries are as below:

Normal task	1 to 32
Background tasks	65 to 80
Trap tasks	257 to 267

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 Statement

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.

Note

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

Next Statement 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

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 Command window example on the usage of the Not instruction.

```
>print not(1)
 -2
>
```

Off Statement

Turns Off the specified output and after a specified time can turn it back on.

Syntax

Off { *bitNumber* | *outputLabel* }, [*time*], [*parallel*] [,*Forced*]

Parameters

bitNumber	Integer expression representing 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)
Forced	Optional. Usually omitted.

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. If you want to keep the settings even in case of the emergency stop, this feature can be reconfigured from the [Outputs Off during emergency stop] checkbox in the [Setup]-[System Configuration]-[Controller]-[Preferences].

Forced Flag

This flag is used to turn Off the I/O output at Emergency Stop and Safety Door Open from NoPause task or NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt).
Be sure that the I/O outputs change by Emergency Stop and Safety Door Open when designing the system.

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 Command window are as follows:

```
> on 1
> off 1, 10       'Turn Output 1 off, wait 10 seconds, turn on again
> on 2
> off 2
```

OLAccel Statement

Sets up the automatic adjustment of acceleration/deceleration that is adjusted according to the overload rating.

Syntax

OLAccel {On | Off}

Parameters

On | Off	On: Enables the automatic adjustment of acceleration/deceleration that is adjusted according to the overload rating.
	Off: Disables the automatic adjustment of acceleration/deceleration that is adjusted according to the overload rating.

Description

OLAccel can be used to enable the automatic adjustment function of acceleration and deceleration that is adjusted according to the robot loading rate (OLRate). When OLAccel is On, the acceleration and deceleration are automatically adjusted in accordance with the robot loading rate at PTP motion commands. This is done to prevent the over load error by reducing the acceleration/deceleration automatically when the loading rate is exceeding a certain value at PTP motion. Heretofore, when users were executing motion with heavy duty that may cause over load error, users had to stop the robot by the program or adjust the speed and acceleration to prevent the error. OLAccel statement lessens these measures. However, this statement do not prevent over load error at all types of cycles. When the cycle has very heavy duty and load, the over load error may occur. In this case, users need to stop the robot or adjust the speed and acceleration. In some operation environment, the motor temperature may rise by operating the robot without over load error and result in over heat error.

This statement is unnecessary at proper load operation.
Use OLRate in the test cycle to check whether the over load error may occur or not.

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

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

Note

If OLAccel On is executed to a robot that does not support the automatic adjustment function of acceleration and deceleration, an error occurs.

See Also

OLAccel Function, OLRate

OLAccel Statement Example

```
>olaccel on
>olaccel
OLACCEL is ON

Function main
  Motor On
  Power High
  Speed 100
  Accel 100, 100
  OLAccel On
  Xqt 2, MonitorOLRate
    Do
      Jump P0
      Jump P1
    Loop
Fend

Function MonitorOLRate
  Do
    'Displays OLRate
    OLRate
    Wait 1
  Loop
Fend
```

OLAccel Function

Returns the automatic adjustment setting.

Syntax

OLAccel

Return Values

Off =	Automatic adjustment of acceleration/deceleration that is adjusted according to the overload rating is disabled.
On =	Automatic adjustment of acceleration/deceleration that is adjusted according to the overload rating is enabled.

See Also

OLAccel, OLRate

OLAccel Function Example

```
If OLAccel = Off Then
   Print “OLAccel is off”
EndIf
```

OLRate Statement

Display overload rating for one or all joints for the current robot.

Syntax

OLRate [*jointNumber*]

Parameters

jointNumber	Integer expression from 1 to 9. The additional S axis is 8 and T axis is 9.

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 command 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
    OLRate      ' Display OLRate
    Wait 1
  Loop
Fend
```

OLRate Function

Returns overload rating for one joint for the current robot.

Syntax

OLRate(*jointNumber***)**

Parameters

jointNumber	Integer expression from 1 to 9. The additional S axis is 8 and T axis is 9.

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 command 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

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

Turns on the specified output and after a specified time can turn it back off**.**

Syntax

On { *bitNumber* | *outputLabel* }, [*time*], [*parallel*] [,*Forced*]

Parameters

bitNumber	Integer expression representing 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)
Forced	Optional. Usually omitted.

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 *EPSON RC+ User's Guide*. The individual bits for the remote connector can be set as remote or I/O from [Setup]-[System Configuration]-[Controller]-[Remote Control] panel.

Outputs and When an Emergency Stop Occurs

The Controller has a feature which causes all outputs to go off when an E-Stop occurs. If you want to keep the settings even in case of the emergency stop, this feature can be reconfigured from the [Outputs Off during emergency stop] checkbox in [Setup]-[System Configuration]-[Controller]-[Preferences].

Forced Flag

This flag is used to turn On the I/O output at Emergency Stop and Safety Door Open from NoPause task, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), or background tasks.
Be sure that the I/O outputs change by Emergency Stop and Safety Door Open when designing the system.

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 command window are as follows:

```
> on 1
> off 1, 10       'Turn Output 1 off, wait 10 seconds, turn on again
> on 2
> off 2
```

OnErr Statement

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* | **0**}

Parameters

label	Statement label to jump to when an error occurs.
0	Parameters 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 label specified in the EResume 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

Err, EResume

OnErr Statement 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 = 7007 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
```

OpBCD Statement

Simultaneously sets 8 output lines using BCD format. (Binary Coded Decimal)

Syntax

OpBCD *portNumber*, *outData* [, *Forced*]

Parameters

portNumber Integer expression representing I/O output bytes. Where the *portNumber* selection corresponds to the following outputs:

Portnum	Outputs
0	0-7
1	8-15
2	16-23
3	24-31
...	...

outData Integer expression between 0 and 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.

Forced Optional. Usually omitted.

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 to 7, *portNumber* = 1 means outputs 8 to 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 to 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

Note 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 and 255 (0 to 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 *EPSON RC+ User's Guide*. The individual bits for the remote connector can be set as remote or I/O from [Setup]-[System Configuration]-[Controller]-[Remote Control] panel.

Outputs and When an Emergency Stop Occurs:

The Controller has a feature which causes all outputs to go off when an E-Stop occurs. This feature is set or disabled from the [Outputs Off during emergency stop] checkbox in the [Setup]-[System Configuration]-[Controller]-[Preferences].

Forced Flag

This flag is used to turn On the I/O output at Emergency Stop and Safety Door Open from NoPause task, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), or background tasks.
Be sure that the I/O outputs change by Emergency Stop and Safety Door Open when designing the system.

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 command 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
```

OpenDB Statement

Opens a database or Excel workbook.

Syntax

OpenDB *#fileNumber*, { *SQL* | *Accel* | *Eccel* } [, *DBserverName* As String],
{*DBname* As String | *filename* As String }

Parameters

fileNumber	Integer number from 501 to 508
SQL | *Accel* | *Eccel*	Selects a database type you want to open from [SQL], [Access], and [Excel].
DBserverName	If you select [SQL], the SQL server name is specified. If omitted, LOCAL server is specified. The SQL server on the network cannot be specified. If you select [Access] or [Excel], the SQL server name is not specified.
DBname | *filename*	If you select [SQL] as a database, a database name on the SQL server is specified. If you select [Access], Access file name is specified. If omitted the path of Access file name, it searches in the current folder. See ChDisk for the details. If you select [Excel], Excel file name is specified. You can specify Excel 2007 book or Excel 97-2003 book file as Excel file. If you omitted Excel file name, it searches in the current folder. See ChDisk for the details.

Description

Opens the specified database using the specified file number.
The specified database must exist on the disk of PC with installed RC+. Otherwise, it causes an error. The specified file number can be used to identify the database while it is open, but cannot be used to refer to the different database until you close the database with the CloseDB command. The file number is used with the database operation commands (SelectDB, Print#, Input#, CloseDB).
Access and Excel files of Microsoft office 2010 64-bit cannot be used.

Note

- Connection of PC with installed RC+ is required.

See Also

SelectDB, CloseDB, UpdateDB, DeleteDB, Input #, Print #

OpenDB Statement Example

Using the SQL database
The following example uses the SQL server 2000 sample database, Northwind and loads the data from a table.

```
Integer count, i, eid
String Lastname$, Firstname$, Title$

OpenDB #501, SQL, "(LOCAL)", "Northwind"
count = SelectDB(#501, "Employees")
For i = 0 To count - 1
   Input #501, eid, Lastname$, Firstname$, Title$
   Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
Next
CloseDB #501
```

Using Access database

The following example uses Microsoft Access 2007 sample database “Students” and loads the data from a table.

```
Integer count, i, eid
String Lastname$, Firstname$, dummy$

OpenDB #502, Access, "c:\MyDataBase\Students.accdb"
count = SelectDB(#502, "Students")
For i = 0 To count - 1
      Input #502, eid, dummy$, Lastname$, Firstname$
      Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #502
```

Using Excel workbook

The following example uses Microsoft Excel workbook “StudentsList“ and loads the data from a sheet.

```
Integer count, i, eid
String Lastname$, Firstname$

OpenDB #503, Excel, "c:\MyDataBase\Students.xls"
count = SelectDB(#503, "[Students$]")
For i = 0 To count - 1
      Input #503, eid, Lastname$, Firstname$
      Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #503
```

OpenCom Statement

Open an RS-232 communication port.

Syntax

OpenCom *#portNumber*

Parameters

portNumber	Integer expression for RS-232C port number to open. The range of port number is: Real Part 1 to 8 Windows Part 1001 to 1008

Description

You need to connect the specified RS-232C port to the controller.

To use the SPEL$^+$ real part ports, option I/O board must be installed to the Controller.
To use Windows part ports, RC+ setting must be done. For details, refer to the description about RC-232C in the *EPSON RC+ User's Guide 5.13 [Setup] Menu*.

See Also

ChkCom, CloseCom, SetCom

OpenCom Statement Example

```
Integer PortNo

PortNo = 1001
OpenCom #PortNo
Print #PortNo, "Data from COM1"
CloseCom #PortNo
```

OpenCom Function

Acquires the task number that executes OpenCom.

Syntax

OpenCom (*portNumber*)

Parameters

portNumber	Integer expression for RS-232C port number. The range of port number is:	
	Real Part	1 to 8
	Windows Part	1001 to 1008

Description

Acquires the task number that executes OpenCom.

See Also

ChkCom, CloseCom, OpenCom, SetCom

OpenCom Function Example

```
Print OpenCom(PortNo)
```

OpenNet Statement

Open a TCP/IP network port.

Syntax

OpenNet *#portNumber* **As** { **Client | Server** }

Parameters

portNumber	Integer expression for TCP/IP port number to open. Range is from 201 to 216.

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 controllers have their TCP/IP settings configured as follows:

Controller #1:
Port: #201
Host Name: 192.168.0.2
TCP/IP Port: 1000

```
Function tcpip
  OpenNet #201 As Server
  WaitNet #201
  Print #201, "Data from host 1"
Fend
```

Controller #2:
Port: #201
Host Name: 192.168.0.1
TCP/IP Port: 1000

```
Function tcpip
  String data$
  OpenNet #201 As Client
  WaitNet #201
  Input #201, data$
  Print "received '", data$, "' from host 1"
Fend
```

OpenNet Function

Acquires the task number that executes OpenNet.

Syntax

OpenNet (*portNumber*)

Parameters

portNumber	Integer expression for TCP/IP port number. Range is from 201 to 216.

Description

Acquires the task number that executes OpenNet.

See Also

ChkNet, CloseNet, OpenNet, SetNet

OpenNet Function Example

```
    Print OpenNet(PortNo)
```

Oport Function

Returns the state of the specified output.

Syntax

Oport(*outnum*)

Parameters

outnum	Integer expression representing I/O output bits.

Return Values

Returns the specified output bit status as either 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.

Note

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 command 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

Simultaneously sets 8 output bits.

Syntax

Out *portNumber, outData* [, *Forced*]

Parameters

portNumber Integer expression representing I/O output bytes. The portnum selection corresponds to the following outputs:

Portnum	Outputs
0	0-7
1	8-15
...	...

outData Integer number between 0 and 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).

Forced Optional. Usually omitted.

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 *portNumber* parameter defines which group of 8 outputs to use where *portNumber* = 0 means outputs 0 to 7, *portNumber* = 1 means outputs 8 to 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 to 255 and may be represented in Hexadecimal or Integer format. (i.e. &H0 to &HFF or 0 to 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 and 255 (0 to 99 for OpBCD). This allows all possible combinations for the 8 bit output groups to be initiated according to the users specifications.

Forced Flag

This flag is used to turn On the I/O output at Emergency Stop and Safety Door Open from NoPause task, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), or background tasks.
Be sure that the I/O outputs change by Emergency Stop and Safety Door Open when designing the system.

See Also

In, InBCD, MemOff, MemOn, MemOut, MemSw, Off, On, Oport, Sw, Wait

Out Statement 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 to 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 command 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

Returns the status of one byte of outputs.

Syntax

Out(*portNumber***)**

Parameters

portNumber	Integer expression representing I/O output bytes. Where the *portNumber* selection corresponds to the following outputs:

Portnum	Outputs
0	0-7
1	8-15
...	...

Return Values

The output status 8 bit value for the specified port.

See Also

Out Statement

Out Function Example

```
Print Out(0)
```

OutReal Statement

The output data of real value is the floating-point data (IEEE754 compliant) of 32 bits.
Set the status of output port 2 word (32 bits).

Syntax

OutReal *WordPortNumber*, *OutputData* [,*Forced*]

Parameters

WordPortNumber	Integer expression representing I/O output words.
OutputData	Specifies the integer expression representing the output data (Real type value).
Forced	Optional. Normally omitted.

Description

Outputs the specified IEEE754 Real value to the output word port specified by word port number and the following output word port.
Output word label can be used for the word port number parameter.

Note

Forced Flag

This flag is used to turn On the I/O output at Emergency Stop and Safety Door Open from NoPause task or NoEmgAbort task (special task initiated by specifying NoPause or NoEmgAbort at Xqt).

Carefully design the system because the I/O output changes by Emergency Stop and Safety Door Open.

See Also

In, InW, InBCD, InReal, Out, OutW, OpBCD, OutReal Function

OutReal Statement Example

```
OutReal 32, 2.543
```

OutReal Function

Retrieve the output port status as the 32 bits floating-point data (IEEE754 compliant).

Syntax

OutReal (*WordPortNumber*)

Parameters

WordPortNumber	Integer expression representing I/O output words.

Return Values

Returns the specified output port status in 32 bits floating-point data (IEEE754 compliant).

See Also

In, InW, InBCD, InReal, Out, OutW, OpBCD, OutReal

OutReal Function Example

```
Real rdata01

rdata01 = OutReal(0)
```

OutW Statement

Simultaneously sets 16 output bits.

Syntax

OutW *wordPortNum, outputData* [, *Forced*]

Parameters

wordPortNum	Integer expression representing I/O output words.
outputData	Specifies output data (integers from 0 to 65535) using an expression or numeric value.
Forced	Optional. Usually omitted.

Description

Changes the current status of user I/O output port group specified by the word port number to the specified output data.

Note

Forced Flag

This flag is used to turn On the I/O output at Emergency Stop and Safety Door Open from NoPause task, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), or background tasks.
Be sure that the I/O outputs change by Emergency Stop and Safety Door Open when designing the system.

See Also

In, InW, Out

OutW Statement Example

```
OutW 0, 25
```

OutW Function

Returns the status of one word (2 bytes) of outputs.

Syntax

OutW(*wordPortNum***)**

Parameters

wordPortNum	Integer expression representing I/O output words.

Return Values

The output status 16 bit value for the specified port.

See Also

OutW Statement

OutW Function Example

```
OutW 0, &H1010
```

P# (1. Point Definition) Statement

Defines a robot point by assigning it to a point expression.

Syntax

point = pointExpr
pointLabel = pointExpr

Parameters

point	Expression including numeric number or ( ) (parenthesis) **P***number* **P***(expr)*
pointLabel	Point label
pointExpr	One of the following point data P point number, Point label, Here, Pallet, Point data function (Here function, XY function, JA function, Pulse function, etc..) For details of *pointExpr,* refer to *P# (2. 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 Definition Example

The following examples are done from the command 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
> plist 4
P4 = XY(-450,200,-30,100)/R
```

Add 90 to U coordinate of Pallet(3, 5), and define resulting point as P6:

```
> P5 = Here
> P6 = pallet(3,5) +U(90)
```

P# (2. Point Expression) Statement

Specifies a robot point for assignment and motion commands.

Syntax

point [{ + | - } *point*] [*local*] [*hand(arm)*] [*elbow*] [*wrist*] [*j4flag*] [*j6flag*] [*j1flag*] [*j2flag*] [*relativeOffsets*] [*absoluteCoords*]

Parameters

point	The base point specification. This can be one of the following: **P***number* **P**(*expr*) *pointLabel* **Pallet**(*palletNumber*, *palletIndex*) **Here** **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.
hand(arm)	Optional for SCARA robot (including RS series) and 6-axis robots (including N series). Specify /**L** or /**R** for lefty or righty hand (arm) orientation.
elbow	Optional for 6-axis robots (including N series). Specify /**A** or /**B** for above or below orientation.
wrist	Optional for 6-axis robots (including N series). Specify /**F** or /**NF** for flip or no flip orientation.
j4flag	Optional for 6-axis robots (including N series). Specify /**J4F0** or /**J4F1**.
j6flag	Optional for 6-axis robots (including N series). Specify /**J6F0** - /**J6F127**.
j1flag	Optional for RS series and 6-axis robots (excluding N series). Specify /**J1F0** or /**J1F1**.
j2flag	Optional for RS series. Specify /**J2F0** - /**J2F127**.
j1angle	Optional for RS and N series. Specify /**J1A** (real value).
j4angle	Optional for N series. Specify /**J4A** (real value).
relativeOffsets	Optional. One or more relative coordinate adjustments. **{+ | -} {X | Y | Z | U | V | W | RZ | RY | RX | R | S | T | ST } (***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 | R | S | T | ST } (***expr***)**

Description

Point expressions are used in point assignment statements and motion commands.

```
    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 Here +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.

To make a relative rotation around the coordinate axis of the 6-axis robots (including N series), execute the statement as follows. The following statement rotates the tool 20 degrees in the X-axis positive direction based on the current tool orientation.

```
Go Here +RX(20)
```

You can also use relative tool offsets:

```
Go Here +TLX(20) -TLY(5.5)
```

When the 6-axis robot (including N series) moves to a point calculated by such as pallet or relative offsets, the wrist part may rotate to an unintended direction. The point calculation above does not depend on robot models and results in motion without converting the required point flag.
LJM function prevents the unintended wrist rotation.

```
Go LJM(Here +X(20))
```

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 Here :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 Here :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 set the current position to P1:

```
P1 = Here @1
```

See Also

Go, LJM, Local, Pallet, Pdel, Plist, Hand, Elbow, Wrist, J4Flag, J6Flag, J1Flag, J2Flag

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 Here :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)
```

PAgl Function

Returns a joint value from a specified point.

Syntax

PAgl (*point*, *jointNumber)*

Parameters

point	Point expression.
jointNumber	Specifies the joint number (integer from 1 to 9) using an expression or numeric value. The additional S axis is 8 and T axis is 9.

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, CR, CS, CT, PPls

PAgl Function Example

```
Real joint1

joint1 = PAgl(P10, 1)
```

Pallet Statement

Defines and displays pallets.

Syntax

(1) **Pallet** [**Outside**,] [*palletNumber, P1, P2, P3* [, *P4*], *columns, rows*]
(2) **Pallet** [**Outside**,] *palletNumber*, *coordinateData* 1, *coordinateData* 2, *coordinateData* 3
[, *coordinateData* 4], *columns*1, *rows*2
(3) **Pallet**

Parameters

Outside	Optional. Allow row and column indexes outside of the range of the specified rows and columns.
palletNumber	Pallet number represented by an integer number from 0 to 15.
P1, P2, P3	Point variables which define standard 3 point pallet position.
P4	Optional. Point variable which is used with P1, P2 and P3 to define 4 point pallet.
columns	Integer expression representing the number of points on the P1(*coordinateData* 1)-to-P2(*coordinateData 2*) side of the pallet. Range is from 1 to 32767.
rows	Integer expression representing the number of points on the P1(*coordinateData* 1)-to-P3(*coordinateData* 3) side of the pallet. Range is from 1 to 32767.
coordinateData1, 2, 3	Point data which is used for pallet definition (standard 3-point definition)
coordinateData 4	Optional. Point data which is used with coordinateData 1 to 3 for 4-point pallet definition.

Return Values

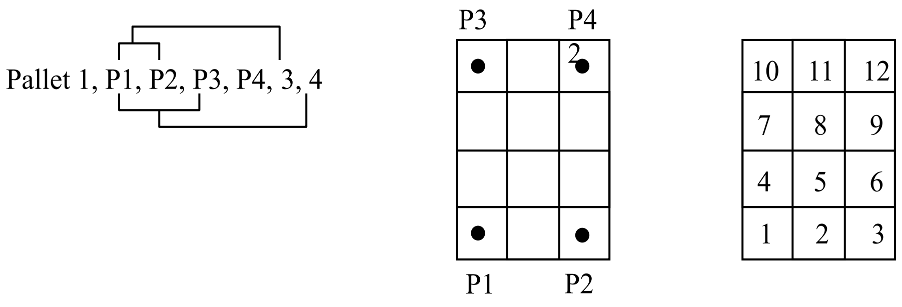
(3) Displays all defined pallets when parameters are omitted.

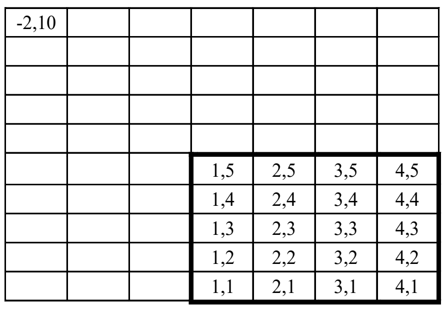
Description

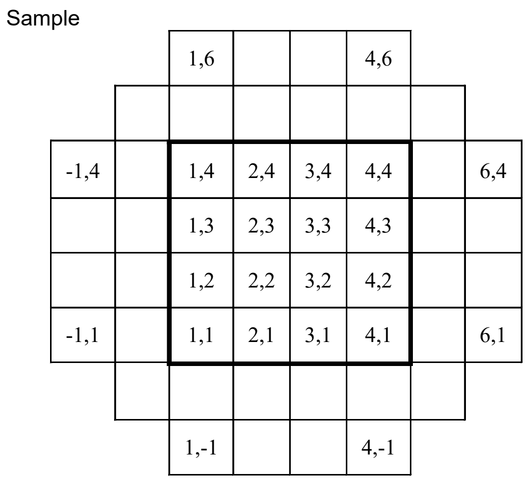
Defines a pallet by teaching the robot, as a minimum, points P1 (*coordinateData* 1), P2 (*coordinateData* 2) and P3 (*coordinateData* 3) and by specifying the number of points from P1 (*coordinateData* 1)to P2 (*coordinateData 2*) and from P1 (*coordinateData* 1) to P3 (*coordinateData* 3).

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.

For example:

```
Pallet Outside 1, P1, P2, P3, 4, 5
Jump Pallet(1, -2, 10)
```

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 5×1 pallet
```

UVW Coordinate Values

When the UVW coordinate values of the 3 (or 4) points specified with the Pallet statement vary, the UVW coordinate values of the point 1 and the coordinate system data 1 are used.
The UVW coordinate values of the point numbers from 2 to 4 and the coordinate system numbers from 2 to 4 are ignored.

Additional Axes Coordinate Values

When the coordinate values of the 3 (or 4) points specified with the Pallet statement include the additional ST axis coordinate values, Pallet includes these additional coordinates in the position calculations. In the case where the additional axis is used as the running axis, the motion of the running axis is considered and calculated with the Pallet definition. You need to define a pallet larger than the robot motion range considering the position of the running axis. Even if you define additional axes that are not affected by the pallet definition, be careful of the positions of additional axes when defining the pallet.

See Also

Pallet Function

Pallet Statement Example

The following instruction from the command 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:

```
P3
  13 14 15
  10 11 12
   7  8  9
   4  5  6
   1  2  3
P1        P2
```

Pallet Function

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 –32768 to 32767.
row	The pallet row represented by an integer expression from –32768 to 32767.

Description

Pallet returns a position in a pallet which was previously defined by the Pallet statement. Use this function with motion commands such as Go and Jump to cause the arm to move to the specified pallet position.

The pallet position number can be defined arithmetically or simply by using an integer.

Notes

Pallet Motion of 6-axis Robot (including N series)

When the 6-axis robot (including N series) moves to a point calculated by such as pallet or relative offsets, the wrist part may rotate to an unintended direction. The point calculation above does not depend on robot models and results in motion without converting the required point flag.
LJM function prevents the unintended wrist rotation.

Pallet Motion of RS series

In the same way as the 6-axis, when the RS series robot moves to a point calculated by such as pallet or relative offsets, Arm #1 may rotate to an unintended direction. LJM function can be used to convert the point flag to prevent the unintended rotation of Arm #1.

In addition, the U axis of RS series may go out of the motion range when the orientation flag is converted, and it causes an error.
To prevent this error, LJM function adjusts the U axis target angle to inside the motion range. It is available when the orientation flag “2” is selected.

UVW Coordinate Values

When the UVW coordinate values of the 3 (or 4) points specified with the Pallet statement vary, the UVW coordinate values of the point 1 and the coordinate system data 1 are used.
The UVW coordinate values of the point numbers from 2 to 4 and the coordinate system numbers from 2 to 4 are ignored.

Additional Axes Coordinate Values

When the coordinate values of the 3 (or 4) points specified with the Pallet statement include the additional ST axis coordinate values, Pallet includes these additional coordinates in the position calculations. In the case where the additional axis is used as the running axis, the motion of the running axis is considered and calculated with the Pallet definition. You need to define a pallet larger than the robot motion range considering the position of the running axis. Even if you define additional axes that are not affected by the pallet definition, be careful of the positions of additional axes when defining the pallet.

See Also

LJM, Pallet

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
```

```
Function main
   Integer i, j

   P0 = XY(300, 300, 300, 90, 0, 180)
   P1 = XY(200, 280, 150, 90, 0, 180)
   P2 = XY(200, 330, 150, 90, 0, 180)
   P3 = XY(-200, 280, 150, 90, 0, 180)

   Pallet 1, P1, P2, P3, 10, 10

   Motor On
   Power High
   Speed 50; Accel 50, 50
   SpeedS 1000; AccelS 5000

   Go P0
   P11 = P0 -TLZ(50)

   For i = 1 To 10
         For j = 1 To 10
            'Specify points
            P10 = P11                 'Depart point
         P12 = Pallet(1, i, j)        'Target point
         P11 = P12 -TLZ(50)           'Start approach point
         'Converting each point to LJM
         P10 = LJM(P10)
         P11 = LJM(P11, P10)
         P12 = LJM(P12, P11)
         'Execute motion
            Jump3 P10, P11, P12 C0
      Next
   Next
Fend
```

```
Function main2
   P0 = XY(300, 300, 300, 90, 0, 180)
   P1 = XY(400, 0, 150, 90, 0, 180)
   P2 = XY(400, 500, 150, 90, 0, 180)
   P3 = XY(-400, 0, 150, 90, 0, 180)
   Pallet 1, P1, P2, P3, 10, 10

   Motor On
   Power High
   Speed 50; Accel 50, 50
   SpeedS 1000; AccelS 5000

   Go P0

   Do
      ' Specify points
      P10 = Here -TLZ(50)                                  'Depart point
      P12 = Pallet(1, Int(Rnd(9)) + 1, Int(Rnd(9)) + 1)'Target point
      P11 = P12 -TLZ(50)                                   'Start approach point

      If TargetOK(P11) And TargetOK(P12) Then              'Point check
      ' Converting each point to LJM
      P10 = LJM(P10)
      P11 = LJM(P11, P10)
      P12 = LJM(P12, P11)
      'Execute motion
         Jump3 P10, P11, P12 C0
      EndIf
    Loop
Fend
```

PalletClr Statement

Clears a defined pallet.

Syntax

PalletClr *palletNumber*

Parameters

palletNumber	Pallet number represented by integer expression from 0 to 15.

See Also

Pallet

PalletClr Example

```
PalletClr 1
```

ParseStr Statement / Function

Parses 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$()	Output array of strings containing the tokens. The array declared by ByRef cannot be specified.
delimiters$	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

Executes simultaneous four joint Point to Point motion, passing near but not through the specified points.

Syntax

Pass *point* [, {**On | Off | MemOn | MemOff**} *bitNumber* [*, point ...*]] [LJM [*orientationFlag*]]

Parameters

point	**P***number* or **P**(*expr*) or point label. When the point data is continued and in the ascending order or the descending order, specify two point numbers binding with colon as P(1:5).
bitNumber	The I/O output bit or memory I/O bit to turn on or off. Integer number between 0 - 511 or output label.
LJM	Optional. Convert the depart point, approach point, and target destination using LJM function.
orientationFlag	Optional. Specifies a parameter that selects an orientation flag for LJM function.

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.

With LJM parameter, the program using LJM function can be more simple.
For example, the following four-line program

```
    P11 = LJM(P1, Here, 1)
    P12 = LJM(P2, P11, 1)
    P13 = LJM(P3, P12, 1)
    Pass P11, P12, P13
```

can be… one-line program.

```
    Pass P1, P2, P3 LJM 1
```

LJM parameter is available for 6-axis (including N series) and RS series robots.
When using *orientationFlag* with the default value, it can be omitted.

```
    Pass P1, P2, P3 LJM
```

See Also

Accel, Go, Jump, Speed

Pass Statement 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

Temporarily stops program execution all tasks for which pause is enabled.

Syntax

Pause

Description

When the Pause is executed, program execution for all tasks with pause enabled (tasks that do not use NoPause or NoEmgAbort in Xqt command) is suspended. Also, if any task is executing a motion statement, it will be paused even if pause is not enabled for that task.
However, Pause cannot stop the background tasks.

Note

QP and its Effect 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.

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
  Go P1
  On 1
  Jump P2
  Off 1
  Pause      'Suspend program execution
  Go P40
  Jump P50
Fend
```

PauseOn Function

Returns the pause status.

Syntax

PauseOn

Return Values

True if the status is pause, otherwise False.

Description

PuseOn function is used only for NoPause, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), and background tasks.

See Also

ErrorOn, EstopOn, SafetyOn, Wait, Xqt

PauseOn Function Example

The following example shows a program that monitors the controller pause and switches the I/O On/Off when pause occurs. However, when the status changes to pause by Safety Door open, the I/O does not turn On/Off.

```
Function main

   Xqt PauseMonitor, NoPause
   :
   :
Fend

Function PauseMonitor
   Boolean IsPause
   IsPause = False
   Do
         Wait 0.1
         If SafetyOn = On Then
               If IsPause = False Then
                     Print "Saftey On"
                     IsPause = True
               EndIf
         ElseIf PauseOn = On Then
               If IsPause = False Then
                     Print "InPause"
                     If SafetyOn = Off Then
                           Off 10
                           On 12
                     EndIf
               IsPause = True
               EndIf
         Else
               If IsPause = True Then
                     Print "OutPause"
                     On 10
                     Off 12
                     IsPause = False
               EndIf
         EndIf
   Loop

Fend
```

PDef Function

Returns the definition status of a specified point.

Syntax

PDef (*point*)

Parameters

point	An integer value or **P***number* or **P**(*expr*) or point label. Cautions for compatibility No variables can be specified for *point* parameter To use variables, write `PDef(P(varName))`.

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
Integer i
For i = 0 to 10
    If PDef (P(i)) Then
        Print "P(";i;") is defined"
    EndIf
Next
```

PDel Statement

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 Statement 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
>
```

PDescription Statement

Define a comment of specified point data.

Syntax

PDescription point data, Newcomment

Parameters

Point data	An integer value or **P***number* or **P**(*expr*) or point label. No variables can be specified for *point data* parameter. To use variables, write `PDescription Statement (P(varName))`, `“new comment”`.
New comment	String expression representing comment of specified point.

Description

PDescription save a description in specified point data of controller memory.
Description saved in memory of the controller is delated from memory when creating or executing a program. Execute the “SavePoints” to save in point file if necessary.

See Also

PDef Function, PDescription$ Function, PLabel, PLabel$ Function

PDescription Statement Example

```
PDescription 1, "Comment"
```

PDescription$ Function

Returns description of point that defined to the specified point number.

Syntax

PDescription$(*pointData*)

Parameters

pointData	An integer value or **P***number* or **P**(*expr*) or point label. No variables can be specified for *point data* parameter. To use variables, write `PDescription$(P(varName))`.

Return Values

Returns descriptions of specified number as a string.

See Also

PDef Function, PDescription, PLabel, PLabel$ Function

PDescription$ Function Example

```
Print PDescription$(1)
Print PDescription$(P(i))
```

PeakSpeedClear Statement

Clears and initializes the peak speed for one or more joints.

Syntax

PeakSpeedClear [*j1* [, *j2* [, *j3* [, *j4* [, *j5* [, *j6* [, *j7* [, *j8* [, *j9*]]]]]]]]]

Parameters

j1 – j9	Integer expression representing the joint number. If no parameters are supplied, then the peak speed values are cleared for all joints. The additional S axis is 8 and T axis is 9. If non-existent joint number is supplied, an error occurs.

Description

PeakSpeedClear clears the peak speed values for the specified joints.
You must execute PeakSpeedClear before executing PeakSpeed.

This command does not support the PG additional axes.

See Also

AvgSpeed, PeakSpeed

PeakSpeedClear Statement Example

<Example 1>

The following is the example to display the speed values of specified joints after clearing the peak speed values of all joints.

```
> PeakSpeedClear
> Go P1
> PeakSpeed 1
    -0.273
> PeakSpeed
    -0.273     -0.164
    -0.080      0.258
    -0.005      0.401
     0.000      0.000
     0.000
>
```

<Example 2>

The following is the example to display the peak speed values of specified joints after clearing the peak speed values of J1, J4, and J5 for the vertical multi-axis robots.

```
> PeakSpeedClear 4, 1, 5
> Go P1
> PeakSpeed 1
     -0.273
> PeakSpeed 4
     0.258
```

PeakSpeed Statement

Displays the peak speed values for the specified joint.

Syntax

PeakSpeed [*jointNumber*]

Parameters

jointNumber	Optional. Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

Return Values

Displays current peak speed values for all joints.

Description

PeakSpeed statement displays the value of the maximum absolute speed for the joint with a sign. The peak speed is a real number from -1 to 1 with 1 being the maximum speed.

Execute PeakSpeedClear first, and then execute PeakSpeed to display the peak speed value for the joint.

When using the virtual controller or conducting dry-run, the average of the absolute speed values is calculated from the commanded speed instead of the actual speed.
This command does not support the PG additional axes.

See Also

AvgSpeed, PeakSpeedClear, PeakSpeed Function

PeakSpeed Statement Example

```
> PeakSpeedClear
> Go P1
> PeakSpeed 1
     -0.273
> PeakSpeed
    -0.273      0.163
    -0.080      0.258
    -0.005     -0.401
     0.000      0.000
     0.000
>
```

PeakSpeed Function

Returns the peak speed for the specified joint.

Syntax

PeakSpeed (*jointNumber*)

Parameters

jointNumber	Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

Return Values

Real value from -1 to 1.

Description

PeakSpeed function returns the value of the maximum absolute speed for the joint with a sign. The peak speed is a real number from -1 to 1 with 1 being the maximum speed.

Execute PeakSpeedClear statement first, and then execute PeakSpeed statement to display the peak speed value for the joint.

When using the virtual controller or conducting dry-run, the average of the absolute speed values is calculated from the commanded speed instead of the actual speed.
This command does not support the PG additional axes.

See Also

AvgSpeed, PeakSpeedClear, PeakSpeed

PeakSpeed Function Example

This example uses the PeakSpeed function in a program:

```
Function DisplayPeakSpeed
   Integer i

   PeakSpeedClear
   Go P1
   Print "Peak Speeds:"
   For i = 1 To 6
      Print "Joint ", i, " = ", PeakSpeed (i)
   Next i
Fend
```

PerformMode Statement

Sets the mode of the robot.

Syntax

(1) **PerformMode** [*modeNumber*] [, *robotNumber*]
(2) **PerformMode**

Parameters

modeNumber Specify the operation mode with an integer value (0 to 2) or with the following constant. This parameter is optional only when the statement is executed from the command window.

Constant	Value	Description
MODE_STANDARD	0	Sets the Standard mode
MODE_BOOST	1	Sets the Boost mode
MODE_LOW_VIBRATION	2	Sets the Low-oscillation mode

robotNumber Specify the robot number by an integer value.
If omitted, currently selected robot will be used.

Result

When specified by the syntax (1), the mode will be set by the mode number that is specified.
When specified by the syntax (2), the mode number of the currently selected robot will be displayed.

Description

PerformMode is a function to change the preference of manipulator performance (mode) according to the intended use. For details of mode, refer to each manipulator manual.

Standard

The cycle time, the motion duty, and the oscillation at the motion stop are balanced.
This mode is available for any kind of application.

BOOST

This mode is specialized to reduce the operating time of a task.
Although this mode aggravates the motion duty and oscillation at the motion stop compared to the standard mode, it can reduce operation time.
Recommended application: Transportation

Low-oscillation

This mode is specialized to reduce the oscillation at the motion stop.
Although this mode increases the operating time compared to the standard mode, it can reduce the oscillation at the motion stop.
Recommended application: Transportation and assembly of precision components

Performance comparison

Mode	Comparison item		
	Operating time (*1)	Oscillation	Motion Duty (*2)
Standard	Normal	Normal	Normal
Boost	Improved	Decreased	Decreased
Low-oscillation	Decreased	Improved	Normal

(*1) Traveling time of the manipulator moving from the current position to the target point.

(*2) Rate of operation time in maximum acceleration without overload error.

Note

- When it's not supported products, an error may occur to change mode to boost or low-oscillation.
- Target motion commands: PTP motion commands (Go, BGo, TGo, Jump, JTran, PTran, Pulse)

* Following performance of the CP motion are not affected by Precede statement.

Trajectory accuracy

Upper limit values of AccelS, AccelR, SpeedS, SpeedR

Frequency of the acceleration setting error and the speed setting error

Conditions that automatically initialize the mode (to the Standard mode)

The table below shows the conditions which automatically initializes the mode.

	Change of the Mode
Controller power ON	Changes to the standard mode
Controller reboot	Changes to the standard mode
Motor ON	Changes to the standard mode
Build / Rebuild	Mode does not change
Reset	Mode does not change
SFree	Changes to the standard mode

See Also

Bo, Go, Jump, JTran, PerformMode Function, TGo

PerformMode Statement Example

```
PerformMode MODE_STANDARD
Go P1
PerformMode 2
Go P2
```

PerformMode Function

Returns the status of the robot operation mode.

Syntax

PerformMode (*[robotNumber]*)

Parameters

robotNumber	Specify the robot number to check the status by an integer value. If omitted, currently selected robot will be used.

Return Values

Returns the integer value representing the currently set operation mode.

0	= Standard mode
1	= Boost mode
2	= Low-oscillation mode

See Also

PerformMode

PerformMode Function Example

```
Print PerformMode(1)
```

PG_FastStop Statement

Stop the PG axes immediately.

Syntax

PG_FastStop

Description

The PG_FastStop stops the current PG robot immediately with no deceleration.
To stop normally, use the PG_SlowStop statement.

See Also

PG_Scan, PG_SlowStop

PG_FastStop Statement Example

The following program moves the PG axis for 10 seconds and stops it.

```
Function main
  Motor On
  PG_Scan 0
  Wait 10
  PG_FastStop          ' Immediately stops the continuous motion
Fend
```

PG_LSpeed Statement

Sets the pulse speed of the time when the PG axis starts accelerating and fishishes decelerating.

Syntax

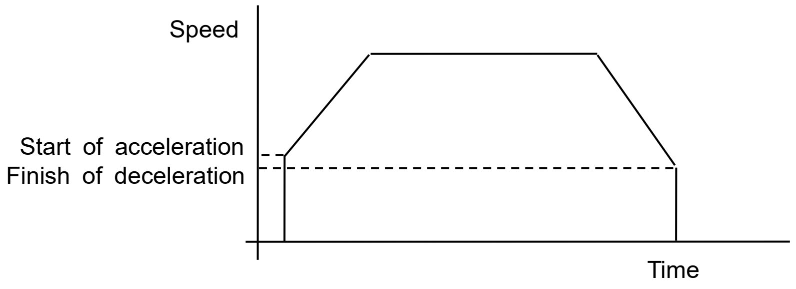
PG_LSpeed *accelSpeed* As Integer [, *decelSpeed* As Integer],

Parameters

speed	Integer expression that contains the pulse speed (1 to 32767 pulse/second)
decalSpeed	Integer expression that contains the pulse speed (1 to 32767 pulse/second)

Description

PG_LSpeed specifies the pulse speed when the PG axis starts accelerating and finishes decelerating. It is useful when setting the initial/ending speed of a stepping motor to higher within the range of max starting frequency to offer the best performance of motor, or setting the speed to lower to prevent the stepping motor from stepping out. The default is 300 pulse/second and do not change to use.



If omitted the finishing speed of deceleration, the speed set value is used.

The PG_LSpeed value initializes to the default values when any one of the following conditions occurs:

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

See Also

PG_LSpeed function

PG_LSpeed Statement Example

You can use the PG_LSpeed in the command window or in the program. The following examples show the both cases.

```
Function pglspeedtst
        Motor On
        Power High
   Speed 30;Accel 30,30
   PG_LSpeed 1000
   Go P0
Fend
```

To set the PG_LSpeed value from the command window.

```
> PG_LSpeed 1000,1100
>
```

PG_LSpeed Function

Returns the pulse speed at the time when the current PG axis starts accelerating and finishes decelerating.

Syntax

PG_LSpeed [(*paramNumber*)]

Parameters

paramNumber	One of the numbers below that specifies the number of set value. If omitted, 1 is used. 1: Pulse speed at acceleration starts 2: Pulse speed at deceleration finishes

Return Values

Integer value from 1 to 32767 in units of pulse/second.

See Also

PG_LSpeed

PG_LSpeed Function Example

```
Integer savPGLSpeed

savPGLSpeed = PG_LSpeed(1)
```

PG_Scan Statement

Starts the continuous spinning motion of the PG robot axes.

Syntax

PG_Scan *direction* As Integer

Parameters

direction	Spinning direction 0: + (CW) direction 1: − (CCW) direction

Description

The PG_Scan starts the continuous spinning motion of the current PG robot.
To execute the continuous spinning motion, you need to enable the PG parameter continuous spinning by the robot configuration.
When the program execution task is completed, the continuous spinning stops.

See Also

PG_FastStop

PG_Scan Statement Example

The following example spins the PG axis for 10 seconds and stops it suddenly.

```
Function main
  Motor On
  Power High
  Speed 10; Accel 10,10
  PG_Scan 0
  Wait 10
  PG_SlowStop
Fend
```

PG_SlowStop Statement

Stops slowly the PG axis spinning continuously.

Syntax

PG_SlowStop

Description

PG_SlowStop decelerates the continuous spinning motion of the current PG robot and bring it to a stop.

See Also

PG_Scan, PG_FastStop

PG_SlowStop Statement Example

The following example spins the PG axis for 10 seconds and stop it suddenly.

```
Function main
  Motor On
  PG_Scan 0
  Wait 10
  PG_SlowStop      ' Stops suddenly the continuous spinning motion
Fend
```

PLabel Statement

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, PDescription, PDescription$ Function, PLabel$ Function, PNumber Function

PLabel Statement Example

```
    PLabel 1, "pick"
```

PLabel$ Function

Returns the point label associated with a point number.

Syntax

PLabel$(*point***)**

Parameters

point	An integer value or **P***number* or **P**(*expr*) or point label. Cautions for compatibility No variables can be specified for *point* parameter To use variables, write `PLabel$(P(varName))`.

See Also

PDef Function, PDescription, PDescription$ Function,PLabel, PNumber Function

PLabel$ Function Example

```
Print PLabel$(1)
Print PLabel$(P(i))
```

Plane Statement

Specifies and displays the approach check plane.

Syntax

(1) **Plane** *PlaneNum* [, *robotNumber*], *pCoordinateData*
(2) **Plane** *PlaneNum* [, *robotNumber*], *pOrigin*, *pXaxis, pYaxis*
(3) **Plane** *PlaneNum* [, *robotNumber*]
(4) **Plane**

Parameters

PlaneNum	Integer value representing the plane number from 1 to 15.
robotNumber	Integer values representing the robot number If omitted, the current robot is used.
pCoordinateData	Point data representing the coordinate data of the approach check plane.
pOrigin	Integer expression representing the origin point using the robot coordinate system.
pXaxis	Integer expression representing a point along the X axis using the robot coordinate system if X alignment is specified.
pYaxis	Integer expression representing a point along the Y axis using the robot coordinate system if Y alignment is specified.

Return Values

When using syntax (3), the setting of the specified plane is displayed.
When using syntax (4), the settings of all plane numbers for the current robot are displayed.

Description

Plane is used to set the approach check plane. The approach check plane is for checking whether the robot end effector is in one of the two areas divided by the specified approach check plane. The position of the end effector is calculated by the current tool. The approach check plane is set using the XY plane of the base coordinate system. The approach check plane detects the end effector when it approaches the area on the + Z side of the approach check plane.

When the approach check plane is used, the system detects approaches in any motor power status during the controller is ON.

The details of each syntax are as follows.

(1) Specifies a coordinate system to create the approach check plane using the point data representing the translation and rotation based on the base coordinate system, and sets the approach check plane.

Example:
```
    Plane 1, XY(x, y, z, u, v, w)
    Plane 1, P1
```

(2) Defines the approach check plane (XP coordinate) by specifying the origin point, point along the X axis, and point along the Y axis. Uses the X, Y, Z coordinates and ignores U, V, W coordinates. Calculates the Z axis in righty and sets the approach checking direction.

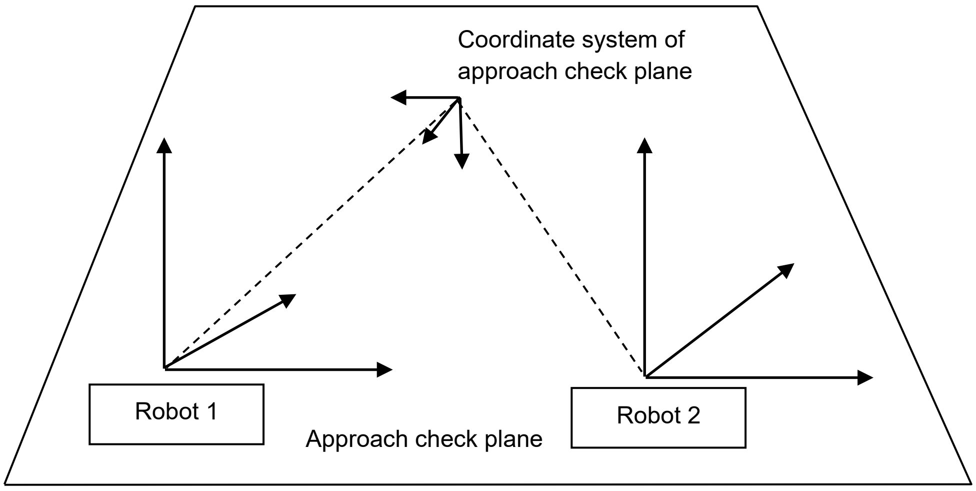
Example:
```
    Plane 1, P1, P2, P3
```

(3) Displays the setting of the specified approach check plane.

(4) Displays all the approach check plane.

You can use the GetRobotInsidePlane function and the InsidePlane function to get the result of the approach check plane. The GetRobotInsidePlane function can be used as the condition for a Wait command. You can provide the detection result to the I/O by setting the remote output setting.

To use one plane with more than one robot, you need to define planes from each robot coordinate system.



Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Notes

Tool Selection

The approach check is executed for the current tool. When you change the tool, the approach check may display the tool approach from inside to outside of the plane or the other way although the robot is not operating.

Additional axis

For the robot which has the additional ST axes (including the running axis), the approach check plane to set doesn't depend on the position of an additional axis, but is based on the robot base coordinate system.

See Also

Box, GetRobotInsidePlane, InsidePlane, PlaneClr, PlaneDef

Tip

Set Plane statement from Robot Manager

EPSON RC+ has a point and click dialog box for defining the approach check plane. The simplest method to set the Plane values is by using the Plane page on the Robot Manager.

Plane Statement Example

These are examples to set the approach check plane using Plane statement.

Check direction is the lower side of the horizontal plane that is –20 mm in Z axis direction in the robot coordinate system:

```
> plane 1, xy(100, 200, -20, 90, 0, 180)
```

Approach check plane is the XY coordinate created by moving 50 mm in X axis and 200 mm in Y axis, rotating 45 degrees around Y axis:

```
> plane 2, xy(50, 200, 0, 0, 45, 0)
```

Set the approach check plane using the tool coordinate system of the robot. (6-axis robot)

```
> plane 3, here
```

Plane Function

Returns the specified approach check plane.

Syntax

Plane(*PlaneNum* [, *robotNumber*])

Parameters

PlaneNum	Integer expression representing the plane number from 1 to 15.
robotNumber	Integer values representing the robot number If omitted, the current robot is used.

Return Values

Returns coordinate data for specified approach check plane.

See Also

GetRobotInsidePlane, InsidePlane, Plane, PlaneClr, PlaneDef

Plane Function Example

```
P1 = Plane(1)
```

PlaneClr Statement

Clears (undefines) a Plane definition.

Syntax

PlaneClr *PlaneNum* [*, robotNumber*]

Parameters

PlaneNum	Integer expression representing the plane number from 1 to 15.
robotNumber	Integer value representing the robot number If omitted, the current robot is used.

Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

GetRobotInsidePlane, InsidePlane, Plane, PlaneDef

PlaneClr Statement Example

```
PlaneClr 1
```

PlaneDef Function

Returns the setting of the approach check plane.

Syntax

PlaneDef (*PlaneNum* [, *robotNumber*])

Parameters

PlaneNum	Integer expression representing the plane number from 1 to 15.
robotNumber	Integer value representing the robot number If omitted, the current robot is used.

Return Values

True if approach detection plane is defined for the specified plane number, otherwise False.

See Also

GetRobotInsidePlane, Box, InsidePlane, Plane, PlaneClr

PlaneDef Function Example

```
Function DisplayPlaneDef(planeNum As Integer)

   If PlaneDef(planeNum) = False Then
      Print "Plane ", planeNum, "is not defined"
   Else
      Print "Plane 1: ",
      Print Plane(PlaneNum)
   EndIf
Fend
```

PList Statement

Displays point data in memory for the current robot.

Syntax

(1) **PList**
(2) **PList** *pointNumber*
(3) **PList** *startPoint***,**
(4) **PList** *startPoint***,** endpoint

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, no data will be displayed.
When a start point number is specified larger than the end point number, then an error occurs.

(1) PList
Displays the coordinate data for all points.

(2) PList *pointNumber*
Displays the coordinate data for the specified point.

(3) PList *startPoint*,
Displays the coordinate data for all points starting with *startPoint*.

(4) PList *startPoint*, *endPoint*
Displays the coordinate data for all points starting with *startPoint* and ending with *endPoint*.

PList Statement Example

Display type depends on the robot type and existence of additional axes.
The following examples are for a Scara robot without additional axes.

Displays the specified point data:

```
> plist 1
P1   = XY(   200.000,    0.000,  -20.000,    0.000 )  /R /0
>
```

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  =   XY( 290.000,    0.000,  -20.000,    0.000 ) /R /0
P12  =   XY( 300.000,    0.000,    0.000,    0.000 ) /R /0
P20  =   XY( 285.000,   10.000,  -30.000,   45.000 ) /R /0
>
```

Displays the point data starting with point number 10.

```
> plist 10,
P10  =   XY( 290.000,    0.000,  -20.000,    0.000 ) /R /0
P12  =   XY( 300.000,    0.000,    0.000,    0.000 ) /R /0
P20  =   XY( 285.000,   10.000,  -30.000,   45.000 ) /R /0
P30  =   XY( 310.000,   20.000,  -50.000,   90.000 ) /R /0
```

PLocal Statement

Sets the local attribute for a point.

Syntax

PLocal(*point***) =** *localNumber*

Parameters

point	An integer value or **P***number* or **P**(*expr*) or point label. Cautions for compatibility No variables can be specified for *point* parameter To use variables, write `PLocal(P(varName))`.
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

Returns the local number for a specified point.

Syntax

PLocal(*point*)

Parameters

point	An integer value or **P***number* or **P**(*expr*) or point label. Cautions for compatibility No variables can be specified for *point* parameter To use variables, write `PLocal(P(varName))`.

Return Values

Local number for specified point.

See Also

PLocal

PLocal Function Example

```
Integer localNum

localNum = PLocal(pick)
```

Pls Function

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. The additional S axis is 8 and T axis is 9.

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$)
```

PosFound Function

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

Switches Power Mode to high or low and displays the current status.

Syntax

(1) **Power** { **High | Low** } [, Forced]
(2) **Power**

Parameters

High | Low	The setting can be High or Low. The default is Low.
Forced	Optional. This parameter is usually omitted.

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.

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.

The following operations will switch to low power mode. In this case, speed and acceleration settings will be limited to the default value. The default value is described in the each manipulator specification table. See also the *EPSON RC+ Users Guide: 2. Sefety*.

Conditions to cause Power Low:
Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

Settings limited to the default value
Speed
Accel
SpeedS
AccelS

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 Command window (directly) or in a program, the speed is set to the default value. If a higher speed motion is required, set Power High.
If you switched to low power mode while the manipulator operating in high power mode, overspeed error or low power torque error may occur.

High Power Mode (Power High) and Its Effect on Max Speed:

In high power mode, higher speeds than the default value can be set.

Forced Flag

The power mode can be changed during robot operation (including the pause state).
If the mode is switched to high power mode while the robot is moving in low power mode, the subsequent motion will be changed to high speed with the specified speed.
If the mode is switched to low power mode while the robot is moving in high power mode, the overspeed error or low power torque error may occur.
Stop the robot and specify the Forced flag to switch to low power mode.

See Also

Accel, AccelS, Speed, SpeedS

Power Statement Example

The following examples are executed from the command 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             'Displays current speed values
Low Power Mode
   50
   50     50

> Accel             'Displays current accel values
Low Power Mode
   100    100
   100    100
   100    100

> Power High        'Sets high power mode

> Jump P2           'Moves robot at high speed
```

Power Function

Returns status of power.

Syntax

Power [(*robotNumber*)]

Parameters

robotNumber	Specify the robot number to check the status by an integer value. If omitted, currently selected robot will be used.

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

Return the pulse position of a specified joint value from a specified point.

Syntax

PPls (*point*, *jointNumber)*

Parameters

point	Point expression.
jointNumber	Expression or numeric value specifying the joint number (integer from 1 to 9) The additional S axis is 8 and T axis is 9.

Return Values

Returns the calculated joint position (long value, in pulses).

See Also

Agl, CX, CY, CZ, CU, CV, CW, Pagl

PPls Function Example

```
Long pulses1

pulses1 = PPls(P10, 1)
```

Print Statement

Outputs data to the current display window, including the Run window, Operator window, Command 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 display 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.

Note

This command can handle up to 256 bytes.

Make Sure Print is used with Wait or a motion within a loop

The Controller may freeze up if only Print is used in loop (loops with no Wait or no motion).
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 #

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 " + Chr$(34) + "P100" + Chr$(34) +
" is ", uvar
Fend
```

Print # Statement

Outputs data to the specified file, communications port, database, or device.

Syntax

Print #*portNumber, expression* [*,expression...*] [,]

Parameters

portNumber	ID number representing a file, communications port, database, or device. File number can be specified in ROpen, WOpen, and AOpen statements. Communications port number can be specified in OpenCom (RS232) and OpenNet (TCP/IP) statements. Database number can be specified in OpenDB statement. Device ID integers are as follows. 21 RC+ 24 TP (TP1 only) 20 TP3
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 or the device specified by *portNumber*.

Notes

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

Maximum data length

This command can handle up to 256 bytes.
However, if the target is a database, it can handle up to 4096 bytes.
If the target is the communications port (TCP/IP), it can handle up to 1024 bytes.

Exchange variable data with other controller

- When more than one string variable or both of numeric variable and string variable is specified, a comma (",") character has to be added expressly to the string data.
  The following programs are examples to exchange the string variable and numeric variable between the Controllers using a communication port.

Sending end (Either pattern is OK.)

```
Print #PortNum, "$Status,", InData, OutData
Print #PortNum, "$Status", ",",InData, OutData
```

Receiving end

```
Input #PortNum, Response$, InData, OutData
```

File write buffering

File writing is buffered. The buffered data can be written with Flush statement. Also, when closing a file with Close statement, the buffered data can be written.

Be sure to use Print # with Wait command or a motion command within a loop

Do not use only Print # in a loop

The Controller may freeze up if only Print # is used in loop (loops with no Wait or no motion).

Depending on the Controller status, information may not be displayed properly even if the Wait command or a motion command is used. If the output is TP1, set Wait time to 1 (seconds) or more. In other cases, set Wait time to 0.1 (seconds) or more.

Bad example

```
Do
    Print #24, "1234"
Loop
```

Good example

```
Do
    Print #24,"1234"
    Wait 1
Loop
```

See Also

Input#, Print, Write, WriteBin

Print # Statement 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 " + Chr$(34) + "port 1" + Chr$(34)
  Print #2 " Next message for " + Chr$(34) + "port 2" + Chr$(34)
Fend
```

PTCLR Statement

Clears and initializes the peak torque for one or more joints.

Syntax

PTCLR [*j1* [, *j2* [, *j3* [, *j4* [, *j5* [, *j6* [, *j7* [, *j8* [, *j9*]]]]]]]]]

Parameters

j1 – j9	Integer expression representing the joint number. If no parameters are supplied, then the peak torque values are cleared for all joints. The additional S axis is 8 and T axis is 9. If non-existent joint number is supplied, an error occurs.

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

<Example 1>

The following is the example to display the torque values of specified joints after clearing the peak torque values of all joints.

```
> ptclr
> go p1
> ptrq 1
    0.227
> ptrq
    0.227      0.118
    0.249      0.083
    0.000      0.000
>
```

<Example 2>

The following is the example to display the torque values of specified joints after clearing the peak torque values of J1, J4, and J5 for the vertical multi-axis robots.

```
> ptclr 4, 1, 5
> go p1
> ptrq 1
    0.227
> ptrq 4
    0.083
```

PTPBoost Statement

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 to 100.
departBoost	Optional. Jump depart boost value. Integer expression from 0 to 100.
approBoost	Optional. Jump approach boost value. Integer expression from 0 to 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 becomes 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:

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

See Also

PTPBoost Function, PTPBoostOK

PTPBoost Statement Example

```
PTPBoost 50, 30, 30
```

PTPBoost Function

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 to 100.

See Also

PTPBoost Statement, PTPBoostOK

PTPBoost Function Example

```
Print PTPBoost(1)
```

PTPBoostOK Function

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 using 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

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.

If a robot includes an additional axis and it is the servo axis, the function will consider the motion time of the additional axis.
If the additional axis is a PG axis, the motion time of the robot will be 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

Perform a relative move of one joint in pulses.

Syntax

PTran *joint, pulses*

Parameters

joint	Integer expression representing which joint to move. The additional S axis is 8 and T axis is 9.
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

Displays the peak torque for the specified joint.

Syntax

PTRQ [*jointNumber*]

Parameters

jointNumber	Optional. Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

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

Returns the peak torque for the specified joint.

Syntax

PTRQ(*jointNumber*)

Parameters

jointNumber	Integer expression representing the joint number. The additional S axis is 8 and T axis is 9.

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

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*] , [*J7*] , [*J8, J9*]
(2) **Pulse**

Parameters

J1 ~ 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 6-axis robots (including N series) and Joint type 6-axis robots.
J7	Optional. For Joint type 7-axis robots.
J8, *J9*	Optional. For the additional axis.

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 Error

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 Command 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

Returns a robot point whose coordinates are specified in pulses for each joint.

Syntax

Pulse (*J1, J2, J3, J4* [*, J5 , J6*] *[, J7*] *[, J8 , J9*])

Parameters

J1 ~ 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 6-axis robots (including N series) and Joint type 6-axis robots.
J7	Optional. For Joint type 7-axis robots.
J8, J9	Optional. For the additional axis.

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

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

QP Statement Example

This Command 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
>
```

QPDecelR Statement

Sets the deceleration speed of quick pause for the change of tool orientation during the CP motion.

Syntax

(1) **QPDecelR** *QPDecelR*
(2) **QPDecelR**

Parameters

QPDecelR	Real value representing the deceleration speed of quick pause during the CP motion (deg/sec^2).

Result

If omitted the parameter, the current QPDecelR set value will be displayed.

Description

QPDecelR statement is enabled when the ROT parameter is used in the Move, Arc, Arc3, BMove, TMove, and Jump3CP statements.
While quick pause is executed in these statements, a joint acceleration error may occur. This is because the deceleration speed of quick pause that is automatically set in a normal quick pause is over the joint allowable deceleration speed. Specifically, the error is likely to occur when the AccelR value in the CP motion is too high or jogging the robot near a singularity. In these cases, use the QPDecelR and set a lower quick pause deceleration speed. But if the setting is too low, the distance for quick pause will increase. Therefore, set the possible value. Normally, you don’t need to set QPDecelR.

You cannot use values lower than the deceleration speed of orientation change in the CP motion set with QPDecelR and AccelR. If you do, a parameter out of range error occurs.
Also, after you set QPDecelR, if a higher value than the set QP deceleration speed is set with the AccelR, the QPDecelR will automatically set the QP deceleration speed same as the deceleration speed set with the AccelR.

The QPDecelR Statement value initializes to the default max deceleration speed when any one of the following conditions occurs:

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or QuitAll stops tasks

See Also

QPDecelR Function, QPDecelS, AccelR

QPDecelR Statement Example

The following program sets the QPDecelR of the Move statement.

```
Function QPDecelTest
   AccelR 3000
   QPDecelR 4000
   SpeedR 100
   Move P1 ROT
   :
   :
Fend
```

QPDecelR Function

Returns the set deceleration speed of quick pause for the change of tool orientation during the CP motion.

Syntax

QPDecelR

Return Values

Real value that contains the set deceleration speed of quick pause for the tool orientation change in the CP motion (deg/s^2).

See Also

QPDecelR, QPDecelS Function

QPDecelR Function Example

```
Real savQPDecelR

savQPDecelR = QPDecelR
```

QPDecelS Statement

Sets the deceleration speed of quick pause in the CP motion.

Syntax

(1) **QPDecelS** *QPDecelS* [, *departDecel*, *approDecel*]
(2) **QPDecelS**

Parameters

QPDecelS	Real value that specifies the deceleration speed of quick pause in the CP motion. (mm/sec^2)
departDecel	Real value that specifies the deceleration speed of quick pause in the Jump3 depart motion (mm/sec^2)
approDecel	Real value that specifies the deceleration speed of quick pause in the Jump3 approach motion (mm/sec^2)

Return Values

If omitted the parameter, the current QPDecelS set value is displayed.

Description

While quick pause is executed in the CP motion, a joint acceleration error may occur. This is because the deceleration speed of quick pause that is automatically set in a normal quick pause is over the joint allowable deceleration speed. Specifically, the error is likely to occur when the AccelS value in the CP motion is too high or jogging the robot near a singularity. In these cases, use the QPDecelS and set a lower quick pause deceleration speed. But if the setting is too low, the distance for quick pause will increase. Therefore, set the possible value. Normally, you don't need to set QPDecelS.

You cannot use values lower than the deceleration speed of the CP motion set with AccelS. If you do, a parameter out of range error occurs.
Also, after you set QPDecelS, if a higher value than the set QP deceleration speed is set with the AccelS, the QPDecelS will automatically set the QP deceleration speed same as the deceleration speed set with the AccelS.
The QPDecelS Statement value initializes to the default max deceleration speed when any one of the following conditions occurs:

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

See Also

QPDecelS Function, QPDecelR, AccelS

QPDecelS Statement Example

The following program sets the QPDecelS of the Move statement.

```
Function QPDecelTest
   AccelS 3000
   QPDecelS 4000
   SpeedS 100
   Move P1
   .
   .
   .
Fend
```

QPDecelS Function

Returns the set deceleration speed of quick pause during the CP motion.

Syntax

QPDecelS (*paramNumber*)

Parameters

paramNumber	Integer expression specifying the one of the following values. 1: Quick pause deceleration speed during the CP motion 2: Quick pause deceleration speed in depart motion during the Jump3 and Jump3CP 3: Quick pause deceleration speed in approach motion during the Jump3 and Jump3CP

Return Values

Real value representing the quick pause deceleration speed (mm/s^2).

See Also

QPDecelS, QPDecelR Function

QPDecelS Function Example

```
Real savQPDecelS

savQPDecelS = QPDecelS(1)
```

Quit Statement

Terminates execution of a specified task or all tasks.

Syntax

Quit { *taskIdentifier* | **All** }

Parameters

taskIdentifier	Task name or integer expression representing the task number. Task name is a function name used in an Xqt statement or a function started from the Run window or Operator window. Task number range is: Normal tasks : 1 to 32 Background task : 65 to 80 Trap tasks : 257 to 267
All	Specifies this parameter if all tasks except the background task should be terminated.

Description

Quit stops the tasks that are currently being executed, or that have been temporarily suspended with Halt.

Quit also stops the task when the specified task is NoPause task, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), or the background tasks.
Quit All stops all tasks including the tasks above other than the background tasks.

Quit All sets the robot control parameter as below:

Robot Control parameter

Current robot Speed, SpeedR, SpeedS	(Initialized to default values)
Current robot QPDecelR , QPDecelS	(Initialized to default values)
Current robot LimZ parameter	(Initialized to 0)
Current robot CP parameter	(Initialized to Off)
Current robot SoftCP parameter	(Initialized to Off)
Current robot Fine	(Initialized to default values)
Current robot Power Low	(Low Power Mode set to On)
Current robot PTPBoost	(Initialized to default values)
Current robot TCLim, TCSpeed	(Initialized to default values)
Current robot PgLSpeed	(Initialized to default values)

See Also

Exit, Halt, Resume, Xqt

Quit Statement 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

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

Initializes the random-number generator.

Syntax

(1) **Randomize** *seedValue*
(2) **Randomize**

Parameters

seedValue	Specify a real value (0 or more) to be basis to retrieve a random number.

See Also

Rnd Function

Randomize Statement Example

```
Function main
    Real r
    Randomize
    Integer randNum

    randNum = Int(Rnd(10)) + 1
    Print "Random number is:", randNum
Fend
```

Range Statement

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]
[, j7Min, j7Max]
[, j8Min, j8Max, j9Min, j9Max]
(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 (including N series) and Joint type 6-axis robots. The lower limit for joint 5 specified in pulses.
j5Max	Optional for 6-Axis robots (including N series) and Joint type 6-axis robots. The upper limit for joint 5 specified in pulses.
j6Min	Optional for 6-Axis robots (including N series) and Joint type 6-axis robots. The lower limit for joint 6 specified in pulses.
j6Max	Optional for 6-Axis robots (including N series) and Joint type 6-axis robots. The upper limit for joint 6 specified in pulses.
j7Min	Optional for Joint type 7-axis robots. The lower limit for joint 7 specified in pulses.
j7Max	Optional for Joint type 7-axis robots. The upper limit for joint 7 specified in pulses.
j8Min	Optional for the additional S axis. The lower limit for joint 8 specified in pulses.
j8Max	Optional for the additional S axis. The upper limit for joint 8 specified in pulses.
j9Min	Optional for the additional T axis. The lower limit for joint 9 specified in pulses.
j9Max	Optional for the additional T axis. The upper limit for joint 9 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.
When parameters are omitted, the current Range values are displayed.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Potential Errors

Attempt to Move Out of Acceptable Range

If the robot arm attempts to move through one of the joint limits error 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.

Note

Range of the lower/upper limits of Joint #6 in pulse differs depending on manipulator model

C4	:	–419430399 to 419430399
C8, C12, N2, N6	:	–26847955 to 26847955

See Also

JRange, SysConfig, XYLim

Range Statement Example

This simple example from the command window displays the current range configurations and then changes them.

```
> range
-18205, 182045, -82489, 82489, -36864, 0, -46695, 46695
>
> range 0, 32000, 0, 32224, -10000, 0, -40000, 40000
>
```

Read Statement

Reads characters from a file or communications port.

Syntax

Read *#portNumber*, *stringVar*$, *count*

Parameters

portNumber	ID number representing a file or communications port to read from. File number can be specified in ROpen, WOpen, and AOpen statements. Communication port number can be specified in OpenCom (RS-232C) or OpenNet (TCP/IP) statements.
stringVar$	Name of a string variable that will receive the character string.
count	Maximum number of bytes to read.

Notes

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

See Also

ChkCom, ChkNet, OpenCom, OpenNet, Write, ReadBin

Read Statement Example

```
Integer numOfChars
String data$

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    Read #1, data$, numOfChars
EndIf
```

ReadBin Statement

Reads binary data from a file or communications port.

Syntax

ReadBin *#portNumber*, *var*
ReadBin *#portNumber, array(), count*

Parameters

portNumber	ID number representing a file or communications port to read from. File number can be specified in BOpen statement. Communication port number can be specified in OpenCom (RS-232C) or OpenNet (TCP/IP) statements.
var	Name of a byte, integer, or long variable that will receive the data.
array()	Name of a byte, integer, or long array variable that will receive the data. Specify a one dimension array variable.
count	Specify the number of bytes to read. The specified count has to be less than or equal to the number of array elements and also smaller than 256 bytes. If the communication port (TCP/IP) is the subject, the count has to be less than or equal to the number of array and also smaller than 1024 bytes.

Notes

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

See Also

Write, WriteBin, Read

ReadBin Statement Example

```
Integer data
Integer dataArray(10)

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    ReadBin #1, data
EndIf

NumOfChars = ChkCom(1)
 If numOfChars > 10 Then
    ReadBin #1, dataArray(), 10
EndIf
```

Real Statement

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 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

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.
Number of valid digits are six digits for Real type.

See Also

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Short, String, UByte, UInt32, UInt64, UShort

Real Statement 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(5, 5, 5)     '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
```

RealAccel Function

Returns the Accel value adjusted by OLAccel.

Syntax

RealAccel(*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 1% or more

Usage

By using RealAccel, the maximum acceleration speed with which the robot can operate continuously can be acquired.
Steps are as follows:
(1) Operate the robot with the OLAccel command On.
(2) Execute the OLRate command and check if the overload ratio rises.
(3) If the overload ratio rises, auto adjustment begins when the overload ratio exceeds 0.5.
(4) After a certain period of time has passed, execute the OLRate command and check that the overload ratio does not rise.
(5) After checking that the overload ratio does not rises, execute the RealAccel function.
(6) The value returned by the RealAccel function is the maximum acceleration speed that the robot can operate continuously in the step (1).
* If the RealAccel function is executed while the overload ratio is rising, maximum acceleration speed of continuous motion cannot be acquired.
* If the overheat error occurs, maximum acceleration speed of continuous motion cannot be acquired by the above procedure.

See Also

Accel, OLAccel, OLRate

RealAccel Function Example

Following is the example of the RealAccel function used in the program.

```
Integer RealAccel1, RealDecel1

Accel 100, 100
OLAccel on

'Acquire the current acceleration speed.
RealAccel1 = RealAccel (1)
RealDecel1 = RealAccel (2)

Display the current acceleration speed
Print RealAccel1
Display the current deceleration speed
Print RealDecel1
```

RealPls Function

Returns the pulse value of the specified joint.

Syntax

RealPls*(jointNumber)*

Parameters

jointNumber	The specific joint for which to get the current pulse count. The additional S axis is 8 and T axis is 9.

Return Values

Returns an integer value representing the current encoder pulse count for the joint specified by *jointNumber*.

Description

RealPls 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

RealPls Function Example

```
Function DisplayPulses

  Long joint1Pulses

  joint1Pulses = RealPls(1)
  Print "Joint 1 Current Pulse Value: ", joint1Pulses
Fend
```

RealPos Function

Returns the current position of the specified robot.

Syntax

RealPos

Return Values

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

Description

RealPos is used to read the current position of the robot.

See Also

CurPos, CX, CY, CZ, CU, CV, CW, RealPls

RealPos Function Example

```
Function ShowRealPos

  Print RealPos
Fend

P1 = RealPos
```

RealTorque Function

Returns the current torque instruction value of the specified joint.

Syntax

RealTorque(*jointNumber*)

Parameters

jointNumber	Specifies the joint number to acquire the torque instruction value using an expression or numeric value. The additional S axis is 8 and T axis is 9.

Return Values

Returns the real value (-1 to 1) representing the proportion in the maximum torque on current power mode. The positive value means the positive direction of the joint angle and the negative value means the negative direction.

See also

TC, TCSpeed, TCLim

RealTorque Function Example

```
Print "Current Z axis torque instruction value (SCARA):",
RealTorque(3)
```

Recover Statement

Executes safeguard position recovery and returns status.
This is for the experienced user and you need to understand the command specification before use.

Syntax

(1) **Recover** *robotNumber* **|** *All*
(2) **Recover** *robotNumber* **|** *All* , *WithMove* **|** *WithoutMove*

Parameters

robotNumber	Robot number that you want to execute recovery for. If omitted, all robots are executed recovery.
All	All robots execute recovery If omitted, same as *All*.
WithMove	A constant whose value is 0. Turns motor on and executes safeguard position recovery. If omitted, same as *WithMove*.
WithoutMove	A constant whose value is 1. Turns the robot motor on. Not usually used. Realizes the special recovery with AbortMotion.

Return Values

Boolean value. True if recover was completed, False if not.

Description

To execute Recover statement from a program, you need to set the [Enable advanced task commands] checkbox in the [Setup] menu-[System Configuration]-[Controller]-[Preferences] page of the EPSON RC+.

Recover can be used after the safeguard is closed to turn on the robot motors and move the robot back to the position it was in when the safeguard was open with low power PTP motion. After Recover has completed successfully, you can execute the Cont method to continue the cycle.

When more than one robot is used in the controller and *All* is specified, all robots are recovered.

See Also

AbortMotion, Cont, Recover Function, RecoverPos

Recover Statement Example

CAUTION	■ When executing the Recover command from a program, you must understand the command specification and confirm that the system has the proper conditions for this command. Improper use such as continuous execution of a command within a loop may deteriorate the system safety.

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

Function monitor
    Do
        If Sw(SGOpenSwitch) = On then
            Wait Sw(SGOpenSwitch) = Off and Sw(RecoverSwitch) = On
            Recover All
        EndIf
    Loop
Fend
```

Recover Function

Executes safeguard position recovery and returns status.
This is for the experienced user and you need to understand the command specification before use.

Syntax

(1) **Recover**
(2) **Recover** (*robotNumber* | *All*)
(3) **Recover** (*robotNumber* | *All* , *WithMove* | *WithoutMove*)

Parameters

robotNumber	Robot number that you want to execute recovery for. If omitted, all robots are executed recovery.
All	All robots execute recovery If omitted, same as *All*.
WithMove	A constant whose value is 0. Turns motor on and executes safeguard position recovery. If omitted, same as *WithMove*.
WithoutMove	A constant whose value is 1. Turns the robot motor on. Not usually used. Realizes the special recovery with AbortMotion.

Return Values

Boolean value. True if recover was completed, False if not.

Description

To execute Recover statement from a program, you need to select the [Enable advanced task commands] checkbox in the EPSON RC+ [Setup]-[System Configuration]-[Controller]-[Preferences].

Recover can be used after the safeguard is closed to turn on the robot motors and move the robot back to the position it was in when the safeguard was open with low power PTP motion. After Recover has completed successfully, you can execute the Cont method to continue the cycle.

When more than one robot is used in the controller and *All* is specified, all robots are recovered.

⚠ CAUTION	■ When executing the Recover command from a program, you must understand the command specification and confirm that the system has the proper conditions for this command. Improper use such as continuous execution of a command within a loop may deteriorate the system safety.

See Also

AbortMotion, Cont, Recover Statement, RecoverPos

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
```

RecoverPos Function

Returns the position where a robot was in when safeguard was open.
This is for the experienced and you need to understand the command specification before use.

Syntax

RecoverPos ([*robotNumber*])

Parameters

robotNumber	Integer value that specifies a robot number If omitted, the current robot number is used.

Return Values

Returns the position the specified robot was in when the safeguard was open.
In the case where the safeguard was not open or the robot has completed the recovery, the coordinates of the returned point data are 0.

Description

This function returns the robot recovery position when using the Cont or Recover commands.

See Also

AbortMotion, Cont, Recover, Recover Function, RealPos

RecoverPos Function Example

If the straight distance of recovery is less than 10 mm, it executes recovery. If more than 10 mm, it finishes the program.

```
If Dist(RecoverPos, RealPos) < 10 Then
     Recover All
Else
     Quit All
EndIf
```

Redim Statement

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	Optional. New dimensions of an array variable may be declared. You must supply the same number of dimensions as when the variable was declared. The subscripts syntax is as follows

(ubound1, [ubound2], [ubound3])

ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension.

The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1.

When specifying the upper bound value, make sure the number of total elements is within the range shown below:

	Others than String	String
Local variable	2,000	200
Global Preserve variable	4,000	400
Global variable and module variable	100,000	10,000

Description

Use Redim to change an array's dimensions at run time. Use Preserve to retain previous values.
The array variable declared by Byref cannot use Redim.

Frequent Redim will decrease the speed of program execution. Especially, we recommend using the minimum of Redim for the global preserve variables.

See Also

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 Statement

Renames a file.

Syntax

Rename *oldFileName*, *newFileName*

Parameters

oldFileName	String expression containing the path and name of the file to rename. See ChDisk for the details.
newFileName	The new name to be given to the file specified by *oldFileName*. See ChDisk for the details.

Description

Changes name of specified file *oldFileName* to *newFileName*.

If path is omitted, Rename searches for *oldFileName* in the current directory.

Rename is only enabled when *oldFileName* and *newFileName* are specified in the same drive.

A file may not be renamed to a filename that already exists in the same path.

Wildcard characters are not allowed in either *oldFileName* or *newFileName*.

See Also

Copy

Rename Statement Example

Example from the command window:

```
> Rename A.PRG  B.PRG
```

RenDir Statement

Rename a directory.

Syntax

Rendir *oldDirName* As String, *newDirName* As String

Parameters

oldDirName	A string expression specifying the path and name of the directory to rename.
newDirName	A string expression specifying the path and new name to be given to the directory specified by *oldDir*. See ChDisk for the details of path.

Description

The same path used for *oldDirName* must be included for *newDirName*.

If both paths of the parameters above are omitted and directory name is only specified, the current directory is specified.

Wildcard characters are not allowed in either *oldDirName* or *newDirName*.

Note

- This statement is executable only with the PC disk.

See Also

MkDir

RenDir Statement Example

```
RenDir "c:\mydata", "c:\mydata1"
```

Reset Statement

Resets the controller into an initialized state.

Syntax

(1) **Reset**
(2) **Reset Error**

Description

Reset resets the items shown below.
Reset Error finishes all non-background tasks and resets the error status and robot control parameters.
To execute the Reset Error statement from programs you need to set the [Enable advanced task commands] preference in the [Setup]-[System Configuration]-[Controller]-[Preference] page of the EPSON RC+.

Emergency Stop Status (reset by Reset only)
Error status
Output Bits (reset by Reset only)
All Output Bits output set to Off except the I/O for Remote and the I/O for hand.
User can set Option Switch to turn this feature off.
Robot Control parameter

Speed, SpeedR, SpeedS	(Initialized to default values)
Accel, AccelR, AccelS	(Initialized to default values)
QPDecelR , QPDecelS	(Initialized to default values)
LimZ parameter	(Initialized to 0)
CP parameter	(Initialized to Off)
SoftCP parameter	(Initialized to Off)
Fine	(Initialized to default values)
Power Low	(Low Power Mode set to On)
PTPBoost	(Initialized to default values)
TCLim, TCSpeed	(Initialized to default values)
PgLSpeed	(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.
Critical error state will not be canceled by Reset.
When critical error occurs, turn Off the controller and solve the cause of the error.
The Reset Statement cannot be executed from a background task or tasks started with the Trap Emergency or Trap Error. Emergency Stop status cannot be reset from programs.

Note

Reset Outputs Preference

([Setup]-[System Configuration]-[Controller]-[Preferences]) If the [Reset command turns off outputs] checkbox is selected, all outputs except bit for hand will be turned OFF when the Reset command is issued. This is important to remember when wiring the system such that turning the outputs off should not cause tooling to drop or similar situations.
For details of hand, refer to Hand Function Manual.

See Also

Accel, AccelS, Fine, LimZ, Motor, Off, On, PTPBoost, SFree, SLock, Speed, SpeedS

Reset Statement Example

Example from the command window.

```
>reset
>
```

ResetElapsedTime Statement

Resets the takt time measurement timer used in ElapsedTime Function.

Syntax

ResetElapsedTime

Description

Resets and starts the takt time measurement timer.

See Also

ElapsedTime Function

ResetElapsedTime Statement Example

```
ResetElapsedTime         'Resets the takt time measurement timer
For i = 1 To 10          'Executes 10 times
    GoSub Cycle
Next
Print ElapsedTime / 10 'Measures a takt time and displays it
```

Restart Statement

Restarts the current main program group.
This command is for the experienced user and you should understand the command specification before use.

Syntax
Restart

Description
Restart stops all tasks and re-executes the last main program group that was running. Background tasks continue to run.
All Trap settings are reset and even if Restart stops tasks, it doesn't execute Trap Abort.
Restart resets the Pause status.
If you execute Restart during error status, reset the error first using a method such as the Reset Error statement.
Restart cannot be used during Emergency Stop status as it causes an error. Emergency Stop status cannot be reset from programs.

⚠ CAUTION	■ When executing the Restart command from a program, you must understand the command specification and confirm that the system has the proper conditions for this command. Improper use such as continuous execution of a command within a loop may deteriorate the system safety.

Note
When using remote control I/O system, do not execute the Restart command of the SPEL+ program and the Start signal of the remote input at the same time. Error 2503 may occur when the programs are executed at the same time.

参照

See Also
Quit, Reset, Trap, Xqt

Restart Statement Example

```
Function main
    Trap Error Xqt eTrap
    Motor On
    Call PickPlac
Fend

Function eTrap

    Wait Sw(ERresetSwitch)
    Reset Error
    Wait Sw(RestartSwitch)
    Restart
Fend
```

Resume Statement

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. Task name is a function name used in an Xqt statement or a function started from the Run window or Operator window. Task number range is: Normal tasks : 1 to 32 Background task : 65 to 80 Trap tasks : 257 to 267
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

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 Returns to the statement following the GoSub instruction.)

Potential Error

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 will occur. A standalone 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
     Exit Function

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

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$ Function 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 Command window.

```
> Print Right$("ABCDEFG", 2)
 FG

> Print Right$("ABC", 3)
 ABC
```

RmDir Statement

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. If the directory name is specified without a path, then the subdirectory in the current directory is specified. See ChDisk for the details of path.

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 Command window, quotes may be omitted.

Note

- This statement is executable only with the PC disk.

Rmdir Statement Example

Example from the command window:

```
> RmDir \mydata
```

Rnd Function

Returns 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

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, Local, Move, Pulse, Robot Function, Speed, SpeedS

Robot Statement 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

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
```

RobotInfo Function

Returns status information for the robot.

Syntax

RobotInfo(*index*)

Parameters

index Integer expression that represents the index of the information to retrieve.

Return Values

The specified information is returned as an integer.

Description

The information for each bit of the returned value is shown in the table below:

Index	Bit	Value	Description
0	0	&H1	Undefined
	1	&H2	Resettable error has occurred
	2	&H4	Non-resettable error has occurred
	3	&H8	Motors are on
	4	&H10	Current power is high
	5	&H20	Undefined
	6	&H40	Undefined
	7	&H80	Undefined
	8	&H100	Robot is halted
	9	&H200	Robot not halted (executing motion or in quick pause)
	10	&H400	Robot stopped by pause or safeguard
	11		Undefined
	12		Undefined
	13		Undefined
	14	&H4000	TILL condition was satisfied by preceding motion command
	15	&H8000	SENSE condition was satisfied by preceding motion command
	16-31		Undefined
1	0	&H1	Robot is tracking (Conveyor tracking)
	1	&H2	Robot is waiting for recovery motion (WaitRecover status)
	2	&H4	Robot is being recovered
	3-31		Undefined
2	0	&H1	Robot is at home position
	1-31		Undefined
3	0	&H1	Joint 1 servo is engaged
	1	&H2	Joint 2 servo is engaged
	2	&H4	Joint 3 servo is engaged
	3	&H8	Joint 4 servo is engaged
	4	&H10	Joint 5 servo is engaged
	5	&H20	Joint 6 servo is engaged
	6	&H40	Joint 7 servo is engaged
	7	&H80	S axis servo is engaged
	8	&H100	T axis servo is engaged
	9-31		Undefined
4	N/A	0 - 32 −1	Number of tasks executing robot commands 0 = command executing from command window or macro −1 = no task is using the manipulator

Index	Bit	Value	Description
5	0	&H1	Joint 1 brake is on
	1	&H2	Joint 2 brake is on
	2	&H4	Joint 3 brake is on
	3	&H8	Joint 4 brake is on
	4	&H10	Joint 5 brake is on
	5	&H20	Joint 6 brake is on
	6	&H40	Joint 7 brake is on
	7	&H80	S axis brake is on
	8	&H100	T axis brake is on
	9-31		Undefined

See Also

CtrlInfo, RobotInfo$, TaskInfo

RobotInfo Function Example

```
If (RobotInfo(3) And &H1) = &H1 Then
  Print "Joint 1 is locked"
Else
  Print "Joint 1 is free"
EndIf
```

RobotInfo$ Function

Returns text information for the robot.

Syntax

RobotInfo$(*index*)

Parameters

index	Integer expression that represents the index of the information to retrieve.

Return Values

A string containing the specified information.

Description

Index	Description
0	Robot name
1	Model name
2	Default point file name
3	Undefined
4	Serial number of robot

See Also

CtrlInfo, RobotInfo, TaskInfo

RobotInfo$ Function Example

```
Print "Robot Name: ", RobotInfo$(0)
```

RobotModel$ Function

Returns the robot model name.

Syntax

RobotModel$

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$ Function Example

```
Print "The robot model is ", RobotModel$
```

RobotName$ Function

Returns the robot name.

Syntax

RobotName$

Return Values

A string containing the robot name.

See Also

RobotInfo, RobotModel$

RobotName$ Function Example

```
Print "The robot name is ", RobotName$
```

RobotSerial$ Function

Returns the robot serial number.

Syntax

RobotSerial$

Return Values

A string containing the robot serial number.

See Also

RobotInfo, RobotName$, RobotModel$

RobotSerial$ Function Example

```
Print "The robot serial number is ", RobotSerial$
```

RobotType Function

Returns the robot type.

Syntax

RobotType

Return Values

1: Joint

2: Cartesian

3: SCARA

5: 6-AXIS

6: RS series

7: N series

See Also

RobotModel$

RobotType Function Example

```
If RobotType = 3 Then
  Print "Robot type is SCARA"
EndIf
```

ROpen Statement

Opens a file for reading.

Syntax

ROpen *fileName* **As** *#fileNumber*

.

.

Close *#fileNumber*

Parameters

fileName	A string expression containing the file name to read from including the path. If only file name is specified, a file in the current directory is specified. See ChDisk for the details.
fileNumber	Integer expression from 30 to 63

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.

Notes

- PC disk only
- A network path is available.

The *fileNumber* identifies the file as long as the file is open and until it is closed the same file number cannot be used to the other files.
The fileNumber is used for the file operation commands (Input#, Read, Seek, Eof, Close)

Close statement closes the file and releases the file number.

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

ROpen Statement Example

```
Integer fileNum, i, j

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

fileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print "data = ", j
Next i
Close #fileNum
```

ROTOK Function

Returns whether an ROT modifier parameter can be added when issuing a motion command to a destination.

Syntax

ROTOK (*destination coordinates*)
ROTOK (*reference coordinates*, *destination coordinates*)

Parameters

destination coordinates	Specifies the destination coordinates in point data to check whether an ROT modifier parameter can be added.
reference coordinates	Specifies the reference coordinates in point data to check whether an ROT modifier parameter can be added.

Return Values

Returns "True" if an ROT modifier parameter can be added, and "False" if not.

Description

Checks whether an ROT modifier parameter can be added before actually moving the robot.
An ROT modifier parameter is a parameter that can be added to Move and other linear interpolation motion commands to prioritize acceleration/deceleration in favor of tool rotation.
The ROTOK function determines whether an error will occur when adding an ROT parameter to Move and other linear interpolation motion commands from the reference coordinates to the destination before moving the robot.
If the reference coordinates are omitted, the result returned will be determined based on the current position (Here).
An error will occur when adding an ROT modifier parameter with a rotation angle of "0", or a small rotation angle. This function determines such an error. Note, however, that it cannot determine whether the robot joint speed and joint acceleration limits have been exceeded during motion (error 4242, etc.). If this occurs, reduce values for SpeedS, SpeedR, AccelS, and AccelR.

Notes

About the Controllers to use

It cannot be used with T/VT series.

See Also

Move

ROTOK Function Example

```
If ROTOK(P1) = True Then
  Move P1 ROT
Else
  Move P1
EndIf
```

RSet$ Function

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

```
    temp$ = "123"
    temp$ = RSet$(temp$, 10)  ' temp$ = "       123"
```

RShift Function

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 (integer from 0 to 31) 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.)

Note

Numeric Data Type:

The numeric data (*number*) may be any valid numeric data type. RShift works with data types: Byte, Double, Int32, Integer, Long, Real, Short, UByte, UInt32, and UShort.

See Also

And, LShift, LShift64, Not, Or, RShift64, Xor

RShift Function 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 command window.

```
> Print RShift(10,1)
5
> Print RShift(8,3)
1
> Print RShift(16,2)
4
```

RShift64 Function

Shifts numeric data to the right by a user specified number of bits.

Syntax

RShift64*(number, shiftBits)*

Parameters

number	Numeric expression to be shifted.
shiftBits	The number of bits (integer from 0 to 63) 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

RShift64 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 RShift64 is that it simply returns the result of *number* / $2^{shiftBits}$. (*Number* is divided by 2 *shiftBit* times.)

Note

Numeric Data Type:

The numeric data (*number*) may be any valid numeric data type. RShift64 works with Int64 and UInt64 data types.

See Also

And, LShift, LShift64, Not, Or, RShift, Xor

RShift64 Function Example

The example shown below shows a program which shows all the possible RShift64 values for an Integer data type starting with the integer set to "0".

```
Function rshif64tst
  UInt64 num, snum, i
  num = 18446744073709551615
  For i = 1 to 63
     Print "i =", i
     snum = RShift64(num, i)
     Print "RShift64(18446744073709551615, ", i, ") = ", snum
  Next i
Fend
```

Some other example results from the RShift64 instruction from the command window.

```
> Print RShift64(10,1)
5
> Print RShift64(8,3)
1
> Print RShift64(16,2)
4
```

RTrim$ Function

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

Runs an EPSON RC+ dialog from a SPEL$^+$ program.

Syntax

(1) **RunDialog** *dialogID*
(2) **RunDialog** DLG_ROBOTMNG [, *robotAllowed*]

Parameters

dialogID Integer expression containing a valid dialog ID. These values are predefined constants as shown below.

DLG_ROBOTMNG	100	Run the Robot Manager dialog
DLG_IOMON	102	Run I/O Monitor
DLG_VGUIDE	110	Run Vision Guide dialog

robotAllowed This parameter is only available when DLG_ROBOTMNG is specified as *dialog ID*. Specifies a robot that is available in the Robot Manager in bit value.

Example	Set value	bit15	bit14	...	bit2	bit1	bit0
Robot 1	&H0001	Off	Off		Off	Off	On
Robot 2	&H0002	Off	Off		Off	On	Off
Robot 1 and 2	&H0003	Off	Off		Off	On	On
:							
Robot 16	&H1000	On	Off		Off	Off	Off

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 Statement Example

```
If Motor = Off Then
    RunDialog DLG_ROBOTMNG
    If Motor = Off Then
        Print "Motors are off, aborting program"
        Quit All
    EndIf
EndIf
```

SafetyOn Function

Return the Safety Door open status.

Syntax

SafetyOn

Return Values

True if the Safety Door is Open, otherwise False.

Description

SafetyOn function is used only for NoPause task, NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt), and background tasks.

See Also

ErrorOn, EstopOn, PauseOn, Wait, Xqt

SafetyOn Function Example

The following example shows a program that monitors the Safety Door open and switches the I/O On/Off when Safety Door open occurs.

Note

Forced Flag

This program example uses Forced flag for On/Off command.
Be sure that the I/O outputs change during error, or at Emergency Stop or Safety Door Open when designing the system.

```
Function main

   Xqt SafetyOnOffMonitor, NoPause
   :
   :
Fend

Function SafetyOnOffMonitor
   Do
         Wait SafetyOn = On
         Print "Saftey Open"
         Off 10, Forced
         On 12, Forced

         Wait SafetyOn = Off
         Print "Saftey Close"
         On 10, Forced
         Off 12, Forced
   Loop
Fend
```

SavePoints Statement

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 ".pts". You cannot specify a file path and fileName doesn't have any effect from ChDisk. See ChDisk for the details.

Description

SavePoints saves points for the current robot to the specified file in the current project directory. A ".pts" 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.

The point data is stored in the Compact Flash inside of the controller. Therefore, SavePoints starts writing into the Compact Flash. Frequent writing into the Compact Flash will shorten the Compact Flash lifetime. We recommend using SavePoints only for saving the point data.

Potential Errors

Out of Disk Space

If there is no space remaining an error will occur.

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 occur.

See Also

ImportPoints, LoadPoints

SavePoints Statement Example

```
ClearPoints
For i = 1 To 10
   P(i) = XY(i, 100, 0, 0)
Next i
SavePoints "TEST.PTS"
```

Seek Statement

Changes position of file pointer for a specified file.

Syntax

Seek *#fileNumber*, *pointer*

Parameters

fileNumber	Integer expression from 30 to 63
pointer	Integer expression for the desired position to seek, starting from 0 to the length of the file.

See Also

BOpen, Read, ROpen, UOpen, Write, WOpen

Seek Statement Example

```
Integer fileNum
String data$

fileNumber = FreeFile
UOpen "TEST.DAT" As #fileNum
Seek #fileNum, 20
Read #fileNum, data$, 2
Close #fileNum
```

Select...Send Statement

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 may include constants, variables, and logical operators that use And, Or and Xor. *caseExpr* also may include constants, variables, and logical operators that use And, Or and Xor. In this case, the calculation result of *caseExpr* is compared to that of *selectExpr* and do not specify the variable in *caseExpr* because the motion becomes complicated.

See Also

If...Then...Else

Select…Send Statement 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
```

SelectDB Statement

Searches the data in the table in an opened database.

Syntax

SelectDB (*#fileNumber*, *TableName*, *SelectCondition*, *SortMethod*)

Parameters

#fileNumber	Integer value from 501 to 508 representing the database number specified with the OpenDB statement
TableName	Table name you want to search in If the database type specified with *#fileNumber* is an Excel workbook, specify an Excel worksheet or named table When specifying an Excel sheet, add $ to end of the worksheet name and enclose the name with [ ]. When specifying an area with a name in an Excel worksheet, enclose the name with [ ].
SelectCondition	Conditions of the search. AND, OR are available to specify the multiple conditions. If omitted, the all data in the table is searched.
SortMethod	Order to show searched data Specify Sort key and Sort order (ascending order [ASC] / descending order [DESC]). If the Sort order is omitted, the ascending Sort key order is specified. If the *SortMethod* is omitted, the order is decided by the opened database.

Return Values

Returns total numbers of lines.

Description

Sorts the data which meets the *SelectCondition* in the specified table of the opened database based on the Sort conditions.
You should execute SelectDB before reading / writing data with the Input# and Print# statements.
If the opened database is an Excel workbook, write a row name to use for the search in the first line of the worksheet and area defined with the name.
For Excel 2007 workbook, the worksheet name must be specified. You cannot access to area defined with the name.

Note

- Connection of PC with installing RC+ is required.

See Also

OpenDB,CloseDB, UpdateDB,DeleteDB, Input #, Print #

SelectDB Example

The following example uses the SQL server 2000 sample database, Northwind.
The Employees table is searched with the condition TitleOfCourtesy = Ms. with EmployeeID in descending order.

```
       Integer count, i, eid
       String Lastname$, Firstname$, Title$

       OpenDB #501, SQL, "(LOCAL)", "Northwind"
       count  =  SelectDB(#501,  "Employees",  "TitleOfCourtesy  =  'Ms.'",
    "EmployeeID DESC")
       For i = 0 To count - 1
          Input #501, eid, Lastname$, Firstname$, Title$
          Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
       Next
       CloseDB #501
```

Using Access database

The following example uses Microsoft Access 2007 sample database “Students” and loads the data whose ID is more than 10 from the table “Students” in the ID descending order.

```
       Integer count, i, eid
       String Lastname$, Firstname$, dummy$

       OpenDB #502, Access, "c:\MyDataBase\Students.accdb"
       count = SelectDB(#502, "Students", "ID > 10'", "ID")
       For i = 0 To count - 1
          Input #502, eid, dummy$, Lastname$, Firstname$
          Print eid, ",", Lastname$, ",", Firstname$
       Next
       CloseDB #502
```

Using Excel workbook

The following example uses Microsoft Excel workbook “Students“ and loads the data in worksheet “Student” whose Age is under 25 with the ID in ascending order.

```
       Integer count, i, eid
       String Lastname$, Firstname$

       OpenDB #503, Excel, "c:\MyDataBase\Students.xls"
       count = SelectDB(#503, "[Students$]", "Age < 25", "ID ASC")
       For i = 0 To count - 1
          Input #503, eid, Lastname$, Firstname$
          Print eid, ",", Lastname$, ",", Firstname$
       Next
       CloseDB #503
```

Sense Statement

Specifies and displays input condition that, if satisfied, completes the Jump in progress by stopping the robot above the target position.

Syntax

Sense [*condition*]

Parameters

condition	Input status specified as a trigger [*Event*] comparative operator (=, <>, >=, >, <, <=) [*Integer expression*] The following functions and variables can be used in the *Event*: Functions : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus Variables : Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort global preserve variable, Global variable, module variable In addition, using the following operators you can specify multiple event conditions. Operator : And, Or, Xor Example : Sense Sw(5) = On Sense Sw(5) = On And Sw(6) = Off

Description

Sense is used to stop approach motion during a Jump, Jump3, and Jump3CP instructions. The Sense condition must include at least one of the functions above.

When variables are included in the Sense condition, their values are computed when setting the Sense condition. No use of variable is recommended. Otherwise, the condition may be an unintended condition. 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 initial Sense condition is:
Sense Sw(0) = On 'Robot does not execute downward motion when 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 after executing a motion command using Sense modifier.

To use a variables in the event condition expression

- Available variables are Integer type (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)
- Array variables are not available
- Local variables are not available
- If a variable value cannot satisfy the event condition for more than 0.01 seconds, the system cannot retrieve the change in variables.
- Up to 64 can wait for variables in one system (including the ones used in the event condition expressions such as Wait). If it is over 64, an error occurs during the project build.
- If you try to transfer a variable waiting for variables as a reference with Byref, an error occurs.
- When a variable is included in the right side member of the event condition expression, the value is calculated when the motion command start. We recommend not using variables in an integer expression to avoid making unintended conditions.

See Also

In, JS, Jump, Jump3, Jump3CP, MemIn, MemSw, Stat, Sw, SF_GetStatus

Sense Example

This is a simple example on the usage of the Sense instruction.

```
Function test
   .
   .
TrySense:
    Sense Sw(1) = Off     'Specifies the arm stops above the target when
                          'the input bit 1 is Off.
    Jump P1 C2 Sense
    If JS = True Then
         GoSub ERRPRC     'If the arm remains stationary
         GoTo TrySense    'above the point specified,
                          'then execute ERRPRC and go to TrySense.
    EndIf
    On 1; Wait 0.2; Off 1
   .
   .
Fend
```

<Other Syntax Examples>

```
> Sense Sw(1)=1 And MemSw(1)=1

> Sense Sw(0) Or (Sw(1) And MemSw(1))
```

SetCom Statement

Sets or displays parameters for RS-232C port.

Syntax

SetCom *#portNumber* [, *baud*] [, *dataBits*] [, *stopBits*] [, *parity*] [, *terminator*] [, *HWFlow*] [, *SWFlow*] [, *timeOut*]

Parameters

portNumber	Integer value representing a RS-232C port number Real Part 1 to 8 Windows Part 1001 to 1008
baud	Optional. Specifies the baud rate. Valid values are: 110 2400 19200 300 4800 38400 600 9600 56000 1200 14400 115200 (Default: 9600) When using the Windows Part port , some data may drop in the baud rate of 19200 or more.
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.

Description

When all the parameter is omitted, displays a communication port setting.

If the several ports are used in the communication at one time with more than 19200 baud rate, error 2929 or 2922 may occur. In this case, select the lower baud rate or avoid using several ports at one time.

When using the Windows Part port, some data may drop in the baud rate of 19200 or more.

If any data drops, select the lower baud rate or use the Real Part port.

Parameters are stored to the Compact Flash inside the Controller. When you execute SetCom, the data is written to the Compact Flash. If a data is written to the Compact Flash frequently, it may shorten the Compact Flash life. Using SetCom only when changing the parameter is recommended.

See Also

OpenCom, CloseCom, SetNet

SetCom Statement Example

```
SetCom #1, 9600, 8, 1, N, CRLF, NONE, NONE, 0

SetCom #2, 4800
```

SetLatch Statement

Sets the latch function of the robot position using the R-I/O input.

Syntax

SetLatch { *#portNumber*, *triggerMode, continuance latch times*}

Parameters

#portNumber Port number of the R-I/O input port to connect the trigger input signal.
The table below shows the port numbers you can specify.
Specify the port number of the unit that the object robot is connected.

		Point	Port Number
Control Unit	INPUT	2 points	24, 25
	OUTPUT	-	-
Drive Unit 1	INPUT	2 points	56, 57
	OUTPUT	-	-
Drive Unit 2	INPUT	2 points	280, 281
	OUTPUT	-	-

The following constants are defines as the port number.

Constant	Port Number
SETLATCH_PORT_CU_0	24
SETLATCH_PORT_CU_1	25
SETLATCH_PORT_DU1_0	56
SETLATCH_PORT_DU1_1	57
SETLATCH_PORT_DU2_0	280
SETLATCH_PORT_DU2_1	281

triggerMode The trigger input signal logic to connect with the R-I/O. The logic can be specified with the following constants.

Constant	Value	Explanation
SETLATCH_TRIGGERMODE_TRAILINGEDGE	0	Negative logic
SETLATCH_TRIGGERMODE_LEADINGEDGE	1	Positive logic

With the negative logic, it latches the robot position at the switch edge from the input signal High to Low.

With the positive logic, it latches the robot position at the switch edge from the input signal from Low to High.

continuance latch times Specify the continuance latch times of robot position by R-I/O input signal.
1, 2, 3 and 4 is available. After LatchEnable On, point data can be latched for the specified number of continuance latches. Maximum is 4 times.
Parameter can be omitted. When it's omitted, it's only 1 time of continuance latch.

Description

Sets the condition of the robot position latch using the R-I/O input signals. One robot cannot wait the trigger signals of several ports simultaneously.
Executing SetLatch needs approx. 40 msec for processing.

Note

If you specify a port number of the unit unrelated to the selected robot, the error "I/O input/output bit number is out of available range" occurs.

See Also

LatchEnable, LatchState Function, LatchPos Function

SetLatch Statement Example

```
Function main
    SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE, 4
                                                    ' Positive logic continuance latch 4 times
    LatchEnable On              'Enable the latch function
    Go P1
    Wait LatchState = True             'Wait a trigger
    Print LatchPos(WithoutToolArm, 1)     'Display the latched position 1
    Print LatchPos(WithoutToolArm, 2)     'Display the latched position 2
    Print LatchPos(WithoutToolArm, 3)     'Display the latched position 3
    Print LatchPos(WithoutToolArm, 4)     'Display the latched position 4
    LatchEnable Off                       'Disable the latch function
Fend
```

Omitting parameter:

```
Function main
     SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE   ' Positive logic
     LatchEnable On                                  ' Enable the latch function
     Go P1
     Wait LatchState = True                          ' Wait a trigger
     Print LatchPos                                  ' Display the latched position
     LatchEnable Off                                 ' Disable the latch function
Fend
```

SetIn Statement

For Virtual IO, sets specified input port (8 bits) to the specified value.

Syntax

SetIn *portNumber*, *value*

Parameters

portNumber	Integer expression representing the input port number.
value	Integer expression between 0 and 255 to set the specified port to.

Description

SetIn provides the ability to set up to 8 bits of virtual inputs at once.

See Also

SetSW, SetInW

SetIn Statement Example

```
> setin 0, 1   ' Sets the first bit of port 0 to On.
```

SetInW Statement

For Virtual IO, sets specified input word (16 bits) to the specified value.

Syntax

SetInW *portNumber*, *value*

Parameters

portNumber	Integer expression representing the input port number.
value	Number between 0 and 65535 to set the specified word to.

Note

Rule of word port which contains the input bit of Real Time I/O

The input bit of the Real Time I/O cannot be reflected.
Specify the setting value of the word ports which contain the input bit of Real Time I/O =1, 3, 17, 19 by an integer from 0 to 255.
The value larger than 255 will result in an error.

Description

SetInW provides the ability to set up to 16 bits of virtual inputs at once.

See Also

SetSw, SetIn

SetInW Statement Example

```
> setinw 0, 1   ' Sets the first bit of word 0 to On.
```

SetNet Statement

Sets parameters for a TCP/IP port.

Syntax

(1) SetNet *#portNumber, hostAddress[, TCP_IP_PortNum [, terminator [, SWFlow [, timeOut, [, protocol [, CloseNet timeout]]]]]]*

(2) SetNet

Parameters

portNumber	Specifies which TCP/IP port to set parameters for. Valid values are 201 to 216.
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.
protocol	Specifies the protocol (TCP/UDP/UDP_SEND/UDP_RECV) of communication. TCP: TCP communication UDP: UDP communication UDP_SEND: UDP send UDP_RECV: UDP receive
CloseNet timeOut	Specifies the time before closing the socket with CloseNet in seconds. (Integer from 0 to 5) When 0 is set, closes the socket without waiting for a response to the shutdown request.

Description

Parameters are stored to the Compact Flash inside the Controller. When you execute SetNet, the data is written to the Compact Flash. If a data is written to the Compact Flash frequently, it may shorten the Compact Flash life. Using SetNet only when changing the parameter is recommended.

See Also

OpenNet, WaitNet, CloseNet, SetCom

SetNet Statement Example

```
SetNet #201, "192.168.0.1", 2001, CRLF, NONE, 0, TCP, 5
```

SetSw Statement

For Virtual IO, sets specified input bit to the specified value.

Syntax

SetSw *bitNumber*, *value*

Parameters

bitNumber	Integer expression representing the input bit number.
value	Integer expression with a value of 0 (Off) or 1 (On).

Description

SetSw provides the ability to turn on or off one input bit.

See Also

SetIn, SetInW

SetSw Statement Example

```
> setsw 2, on   ' Sets the 2nd input bit to On.
```

SF_GetParam Function

Returns information on the safety function parameters.

Syntax

SF_GetParam (*index*)

Parameters

index	Integer or constant expression that represents the index of the information to retrieve.

Return Values

An integer containing the specified information.
If the index "_EN" is specified at the end of the constant, 1 is returned if the parameter is enabled or 0 if it is disabled.

Description

Returns the specified safety function parameter values.

Index	Constant	Description
1	DRYRUNOFF	Dry run is disabled
2	SLS_1_HAND_EN	SLS_1 hand speed is monitored
3	SLS_1_SPEED	Monitored speed setting value for SLS_1
4	SLS_1_ELBOW_EN	SLS_1 elbow (SCARA robots: J2, 6-axis robots: J3) speed is monitored *1
5	SLS_1_JOINT_EN	SLS_1 joint speed is monitored
6	SLS_1_JOINTSPEED	Monitored joint speed setting value for SLS_1
7	SLS_1_WRIST_EN	SLS_1 wrist (6-axis robots: J5) speed is monitored *1
8	SLS_1_SHOULDER_EN	SLS_1 shoulder (6-axis robots: J2) speed is monitored *1
9	SLS_2_HAND_EN	SLS_2 hand speed is monitored
10	SLS_2_SPEED	Monitored speed setting value for SLS_2
11	SLS_2_ELBOW_EN	SLS_2 elbow (SCARA robots: J2, 6-axis robots: J3) speed is monitored *1
12	SLS_2_JOINT_EN	SLS_2 joint speed is monitored
13	SLS_2_JOINTSPEED	Monitored joint speed setting value for SLS_2
14	SLS_2_WRIST_EN	SLS_2 wrist (6-axis robots: J5) speed is monitored *1
15	SLS_2_SHOULDER_EN	SLS_2 shoulder (6-axis robots: J2) speed is monitored *1
16	SLS_3_HAND_EN	SLS_3 hand speed is monitored
17	SLS_3_SPEED	Monitored speed setting value for SLS_3
18	SLS_3_ELBOW_EN	SLS_3 elbow (SCARA robots: J2, 6-axis robots: J3) speed is monitored *1
19	SLS_3_JOINT_EN	SLS_3 joint speed is monitored
20	SLS_3_JOINTSPEED	Monitored joint speed setting value for SLS_3
21	SLS_3_WRIST_EN	SLS_3 wrist (6-axis robots: J5) speed is monitored *1
22	SLS_3_SHOULDER_EN	SLS_3 shoulder (6-axis robots: J2) speed is monitored *1
23	SLS_T2_HAND_EN	SLS_T2 hand speed is monitored
24	SLS_T2_SPEED	Monitored speed setting value for SLS_T2
25	SLS_T2_ELBOW_EN	SLS_T2 elbow (SCARA robots: J2, 6-axis robots: J3) speed is monitored *1
26	SLS_T2_JOINT_EN	SLS_T2 joint speed is monitored
27	SLS_T2_JOINTSPEED	Monitored joint speed setting value for SLS_T2
28	SLS_T2_WRIST_EN	SLS_T2 wrist (6-axis robots: J5) speed is monitored *1

Index	Constant	Description
29	SLS_T2_SHOULDER_EN	SLS_T2 shoulder (6-axis robots: J2) speed is monitored *1
30	SLS_T_SPEED	Monitored speed setting value for SLS_T
31	SLS_T_JOINT_EN	SLS_T joint speed is monitored
32	SLS_T_JOINTSPEED	Monitored joint speed setting value for SLS_T
33	SLS_HAND_OFS_X	TCP offset position in X-axis direction of SLS
34	SLS_HAND_OFS_Y	TCP offset position in Y-axis direction of SLS
35	SLS_HAND_OFS_Z	TCP offset position in Z-axis direction of SLS
36	SLS_1_DELAY	Delay time setting value for SLS_1
37	SLS_2_DELAY	Delay time setting value for SLS_2
38	SLS_3_DELAY	Delay time setting value for SLS_3
39	SLS_JOINT_POS_EN	Joint angle is monitored
40	SLS_JOINT_POS_ANGLE	Monitored joint angle setting value
41	SLP_A_XU_EN	XU (wall: X2, restricted area: X1) position of SLP_A is monitored *2
42	SLP_A_XU_POS	Setting value for XU (wall: X2, restricted area: X1) monitored position of SLP_A *2
43	SLP_A_XL_EN	XL (wall: X1, restricted area: X2) position of SLP_A is monitored *2
44	SLP_A_XL_POS	Setting value for XL (wall: X1, restricted area: X2) monitored position of SLP_A *2
45	SLP_A_YU_EN	YU (wall: Y2, restricted area: Y1) position of SLP_A is monitored *2
46	SLP_A_YU_POS	Setting value for YU (wall: Y2, restricted area: Y1) monitored position of SLP_A *2
47	SLP_A_YL_EN	YL (wall: Y1, restricted area: Y2) position of SLP_A is monitored *2
48	SLP_A_YL_POS	Setting value for YL (wall: Y1, restricted area: Y2) monitored position of SLP_A *2
49	SLP_A_ZU_EN	Z2 position of SLP_A is monitored *2
50	SLP_A_ZU_POS	Setting value for Z2 monitored position of SLP_A *2
51	SLP_A_ZL_EN	Z1 position of SLP_A is monitored *2
52	SLP_A_ZL_POS	Setting value for Z1 monitored position of SLP_A *2
53	SLP_B_XU_EN	XU (wall: X2, restricted area: X1) position of SLP_B is monitored *2
54	SLP_B_XU_POS	Setting value for XU (wall: X2, restricted area: X1) monitored position of SLP_B *2
55	SLP_B_XL_EN	XL (wall: X1, restricted area: X2) position of SLP_B is monitored *2
56	SLP_B_XL_POS	Setting value for XL (wall: X1, restricted area: X2) monitored position of SLP_B *2
57	SLP_B_YU_EN	YU (wall: Y2, restricted area: Y1) position of SLP_B is monitored *2
58	SLP_B_YU_POS	Setting value for YU (wall: Y2, restricted area: Y1) monitored position of SLP_B *2
59	SLP_B_YL_EN	YL (wall: Y1, restricted area: Y2) position of SLP_B is monitored *2
60	SLP_B_YL_POS	Setting value for YL (wall: Y1, restricted area: Y2) monitored position of SLP_B *2
61	SLP_B_ZU_EN	Z2 position of SLP_B is monitored *2
62	SLP_B_ZU_POS	Setting value for Z2 monitored position of SLP_B *2
63	SLP_B_ZL_EN	Z1 position of SLP_B is monitored *2
64	SLP_B_ZL_POS	Setting value for Z1 monitored position of SLP_B *2
65	SLP_C_XU_EN	XU (wall: X2, restricted area: X1) position of SLP_C is monitored *2
66	SLP_C_XU_POS	Setting value for XU (wall: X2, restricted area: X1) monitored position of SLP_C *2
67	SLP_C_XL_EN	XL (wall: X1, restricted area: X2) position of SLP_C is monitored *2
68	SLP_C_XL_POS	Setting value for XL (wall: X1, restricted area: X2) monitored position of SLP_C *2

Index	Constant	Description
69	SLP_C_YU_EN	YU (wall: Y2, restricted area: Y1) position of SLP_C is monitored *2
70	SLP_C_YU_POS	Setting value for YU (wall: Y2, restricted area: Y1) monitored position of SLP_C *2
71	SLP_C_YL_EN	YL (wall: Y1, restricted area: Y2) position of SLP_C is monitored *2
72	SLP_C_YL_POS	Setting value for YL (wall: Y1, restricted area: Y2) monitored position of SLP_C *2
73	SLP_C_ZU_EN	Z2 position of SLP_C is monitored *2
74	SLP_C_ZU_POS	Setting value for Z2 monitored position of SLP_C *2
75	SLP_C_ZL_EN	Z1 position of SLP_C is monitored *2
76	SLP_C_ZL_POS	Setting value for Z1 monitored position of SLP_C *2
77	SLP_J2_MON_RAD	Setting value for J2 axis monitored range radius of SLP
78	SLP_J3_MON_RAD	Setting value for J3 axis monitored range radius of SLP
79	SLP_J5_MON_RAD	Setting value for J5 axis monitored range radius of SLP
80	SLP_J6_MON_RAD	Setting value for J6 axis monitored range radius of SLP
81	SLP_J1_RANGE_MAX	Maximum setting value for J1 axis limit range of soft axis limiting
82	SLP_J1_RANGE_MIN	Minimum setting value for J1 axis limit range of soft axis limiting
83	SLP_J2_RANGE_MAX	Maximum setting value for J2 axis limit range of soft axis limiting
84	SLP_J2_RANGE_MIN	Minimum setting value for J2 axis limit range of soft axis limiting
85	SLP_J3_RANGE_MAX	Maximum setting value for J3 axis limit range of soft axis limiting
86	SLP_J3_RANGE_MIN	Minimum setting value for J3 axis limit range of soft axis limiting
87	SLP_J4_RANGE_MAX	Maximum setting value for J4 axis limit range of soft axis limiting
88	SLP_J4_RANGE_MIN	Minimum setting value for J4 axis limit range of soft axis limiting
89	SLP_J5_RANGE_MAX	Maximum setting value for J5 axis limit range of soft axis limiting
90	SLP_J5_RANGE_MIN	Minimum setting value for J5 axis limit range of soft axis limiting
91	SLP_J6_RANGE_MAX	Maximum setting value for J6 axis limit range of soft axis limiting
92	SLP_J6_RANGE_MIN	Minimum setting value for J6 axis limit range of soft axis limiting
93	SIN_1_SLS_1_EN	SLS_1 function assignment state for SAFETY_IN1
94	SIN_1_SLS_2_EN	SLS_2 function assignment state for SAFETY_IN1
95	SIN_1_SLS_3_EN	SLS_3 function assignment state for SAFETY_IN1
96	SIN_1_SLP_A_EN	SLP_A function assignment state for SAFETY_IN1
97	SIN_1_SLP_B_EN	SLP_B function assignment state for SAFETY_IN1
98	SIN_1_SLP_C_EN	SLP_C function assignment state for SAFETY_IN1
99	SIN_1_SG_EN	Protective stop function assignment state for SAFETY_IN1
100	SIN_1_ESTOP_EN	Emergency stop function assignment state for SAFETY_IN1
101	SIN_2_SLS_1_EN	SLS_1 function assignment state for SAFETY_IN2
102	SIN_2_SLS_2_EN	SLS_2 function assignment state for SAFETY_IN2
103	SIN_2_SLS_3_EN	SLS_3 function assignment state for SAFETY_IN2
104	SIN_2_SLP_A_EN	SLP_A function assignment state for SAFETY_IN2
105	SIN_2_SLP_B_EN	SLP_B function assignment state for SAFETY_IN2
106	SIN_2_SLP_C_EN	SLP_C function assignment state for SAFETY_IN2
107	SIN_2_SG_EN	Protective stop function assignment state for SAFETY_IN2
108	SIN_2_ESTOP_EN	Emergency stop function assignment state for SAFETY_IN2
109	SIN_3_SLS_1_EN	SLS_1 function assignment state for SAFETY_IN3
110	SIN_3_SLS_2_EN	SLS_2 function assignment state for SAFETY_IN3
111	SIN_3_SLS_3_EN	SLS_3 function assignment state for SAFETY_IN3
112	SIN_3_SLP_A_EN	SLP_A function assignment state for SAFETY_IN3
113	SIN_3_SLP_B_EN	SLP_B function assignment state for SAFETY_IN3
114	SIN_3_SLP_C_EN	SLP_C function assignment state for SAFETY_IN3

Index	Constant	Description
115	SIN_3_SG_EN	Protective stop function assignment state for SAFETY_IN3
116	SIN_3_ESTOP_EN	Emergency stop function assignment state for SAFETY_IN3
117	SIN_4_SLS_1_EN	SLS_1 function assignment state for SAFETY_IN4
118	SIN_4_SLS_2_EN	SLS_2 function assignment state for SAFETY_IN4
119	SIN_4_SLS_3_EN	SLS_3 function assignment state for SAFETY_IN4
120	SIN_4_SLP_A_EN	SLP_A function assignment state for SAFETY_IN4
121	SIN_4_SLP_B_EN	SLP_B function assignment state for SAFETY_IN4
122	SIN_4_SLP_C_EN	SLP_C function assignment state for SAFETY_IN4
123	SIN_4_SG_EN	Protective stop function assignment state for SAFETY_IN4
124	SIN_4_ESTOP_EN	Emergency stop function assignment state for SAFETY_IN4
125	SIN_5_SLS_1_EN	SLS_1 function assignment state for SAFETY_IN5
126	SIN_5_SLS_2_EN	SLS_2 function assignment state for SAFETY_IN5
127	SIN_5_SLS_3_EN	SLS_3 function assignment state for SAFETY_IN5
128	SIN_5_SLP_A_EN	SLP_A function assignment state for SAFETY_IN5
129	SIN_5_SLP_B_EN	SLP_B function assignment state for SAFETY_IN5
130	SIN_5_SLP_C_EN	SLP_C function assignment state for SAFETY_IN5
131	SIN_5_SG_EN	Protective stop function assignment state for SAFETY_IN5
132	SIN_5_ESTOP_EN	Emergency stop function assignment state for SAFETY_IN5
133	SOUT_1_STO	STO function assignment state for SAFETY_OUT1
134	SOUT_1_SLS_1	SLS_1 function assignment state for SAFETY_OUT1
135	SOUT_1_SLS_2	SLS_2 function assignment state for SAFETY_OUT1
136	SOUT_1_SLS_3	SLS_3 function assignment state for SAFETY_OUT1
137	SOUT_1_SLS_T2	SLS_T2 function assignment state for SAFETY_OUT1
138	SOUT_1_SLS_T	SLS_T function assignment state for SAFETY_OUT1
139	SOUT_1_SLP_A	SLP_A function assignment state for SAFETY_OUT1
140	SOUT_1_SLP_B	SLP_B function assignment state for SAFETY_OUT1
141	SOUT_1_SLP_C	SLP_C function assignment state for SAFETY_OUT1
142	SOUT_1_EP_RC	Emergency stop (Controller) function assignment state for SAFETY_OUT1
143	SOUT_1_EP_TP	Emergency stop (Teach Pendant) function assignment state for SAFETY_OUT1
144	SOUT_1_EN_SW	Enable switch function assignment state for SAFETY_OUT1
145	SOUT_2_STO	STO function assignment state for SAFETY_OUT2
146	SOUT_2_SLS_1	SLS_1 function assignment state for SAFETY_OUT2
147	SOUT_2_SLS_2	SLS_2 function assignment state for SAFETY_OUT2
148	SOUT_2_SLS_3	SLS_3 function assignment state for SAFETY_OUT2
149	SOUT_2_SLS_T2	SLS_T2 function assignment state for SAFETY_OUT2
150	SOUT_2_SLS_T	SLS_T function assignment state for SAFETY_OUT2
151	SOUT_2_SLP_A	SLP_A function assignment state for SAFETY_OUT2
152	SOUT_2_SLP_B	SLP_B function assignment state for SAFETY_OUT2
153	SOUT_2_SLP_C	SLP_C function assignment state for SAFETY_OUT2
154	SOUT_2_EP_RC	Emergency stop (Controller) function assignment state for SAFETY_OUT2
155	SOUT_2_EP_TP	Emergency stop (Teach Pendant) function assignment state for SAFETY_OUT2
156	SOUT_2_EN_SW	Enable switch function assignment state for SAFETY_OUT2
157	SOUT_3_STO	STO function assignment state for SAFETY_OUT3
158	SOUT_3_SLS_1	SLS_1 function assignment state for SAFETY_OUT3

Index	Constant	Description
159	SOUT_3_SLS_2	SLS_2 function assignment state for SAFETY_OUT3
160	SOUT_3_SLS_3	SLS_3 function assignment state for SAFETY_OUT3
161	SOUT_3_SLS_T2	SLS_T2 function assignment state for SAFETY_OUT3
162	SOUT_3_SLS_T	SLS_T function assignment state for SAFETY_OUT3
163	SOUT_3_SLP_A	SLP_A function assignment state for SAFETY_OUT3
164	SOUT_3_SLP_B	SLP_B function assignment state for SAFETY_OUT3
165	SOUT_3_SLP_C	SLP_C function assignment state for SAFETY_OUT3
166	SOUT_3_EP_RC	Emergency stop (Controller) function assignment state for SAFETY_OUT3
167	SOUT_3_EP_TP	Emergency stop (Teach Pendant) function assignment state for SAFETY_OUT3
168	SOUT_3_EN_SW	Enable switch function assignment state for SAFETY_OUT3
169	POS_ROT_U	Setting value for installation plane rotation U_ROT
170	POS_ROT_V	Setting value for installation plane rotation V_ROT
171	POS_ROT_W	Setting value for installation plane rotation W_ROT
172	POS_OFS_X	Setting value for installation position X_OFS
173	POS_OFS_Y	Setting value for installation position Y_OFS
174	POS_OFS_Z	Setting value for installation position Z_OFS

*1 The correspondence between the monitored parts J2, J3, J5 for Safety Limited Speed in Safety Function Manager and the speed exceeded parts hand, wrist, elbow, and shoulder referred in this manual is as follows:
SCARA robots
J2: Elbow
J3: Not applicable
J5: Not applicable
Hand: Hand
6-axis robots
J2: Shoulder
J3: Elbow
J5: Wrist
Hand: Hand

*2 The correspondence between the monitored position X1, X2, Y1, Y2, Z1, Z2 for Safety Limited Position in Safety Function Manager and the monitored position XL, XU, YL, YU, ZL, ZU referred in this manual is as follows:
When "Wall" is selected for the monitored position
X1 = XL, X2 = XU
Y1 = YL, Y2 = YU
Z1 = ZL, Z2 = ZU (6-axis robots only)

When "Restricted Area" is selected for the monitored position
X1 = XU, X2 = XL
Y1 = YU, Y2 = YL
Z1 = ZL, Z2 = ZU (6-axis robots only)

This command can be used with the Controllers with Safety Board.

SF_GetParam Function Example

```
If SF_GetParam (SLS_1_HAND_EN) = 1 Then
   Print "SLS_1 hand speed monitoring is enabled."
EndIf
```

SF_GetParam$ Function

Returns text information on the safety function parameters.

Syntax

SF_GetParam$ (*index*)

Parameters

index	Integer or constant expression that represents the index of the information to retrieve.

Return Values

A string value containing the specified information.

Description

Returns the specified safety function parameter values.

Index	Constant	Description
1	SF_TOOLVERSION	Version of setting tool
2	SF_CHECKSUM	Safety function parameter checksum
3	SF_LAST_MODIFIED	Last modified date of safety function parameter
4	SF_ROBOT_MODEL_NAME	Manipulator name
5	SF_ROBOT_CHECKSUM	Robot parameters checksum
6	SF_HOFS	Encoder offset (Hofs) for origin of actual robot
7	SF_HOFS_LAST_MODIFIED	Last modified date of Hofs

This command can be used with the Controllers with Safety Board.

SF_GetParam$ Function Example

```
String Checksum$
Checksum$ = SF_GetParam$(SF_CHECKSUM)
Print "Safety function parameter Checksum is ", Checksum$

> Print SF_GetParam$(SF_LAST_MODIFIED)
2022/01/01 00:00:00
```

SF_GetStatus Function

Returns the status bit of the safety function.

Syntax

SF_GetStatus (*index*)

Parameters

index	Integer expression that represents the index of the information to retrieve.

Return Values

An integer containing the specified information of index.

Description

The return values bit information is shown in the following table.

Index	Bit	Value	Description
0	0-6	-	Reserved
	7	&H80	Failure detection of safety board
1	0	&H1	SLS_1 is enabled
	1	&H2	SLS_2 is enabled
	2	&H4	SLS_3 is enabled
	3-7	-	Reserved
2	0	&H1	SLP_A is enabled
	1	&H2	SLP_B is enabled
	2	&H4	SLP_C is enabled
	3-6	-	Reserved
	7	&H80	Soft Axis Limiting is enabled (always on)
3	0	&H1	SAFETY_IN1 signal is High (the function is off) *1
	1	&H2	SAFETY_IN2 signal is High (the function is off) *1
	2	&H4	SAFETY_IN3 signal is High (the function is off) *1
	3	&H8	SAFETY_IN4 signal is High (the function is off) *1
	4	&H10	SAFETY_IN5 signal is High (the function is off) *1
	5-7	-	Reserved
4	0	&H1	SAFETY_OUT1 signal is High (the function is off) *2
	1	&H2	SAFETY_OUT2 signal is High (the function is off) *2
	2	&H4	SAFETY_OUT3 signal is High (the function is off) *2
	3-7	-	Reserved
5~11	0-7	-	Reserved
12	0-7	-	STF_ID
13	0-7	-	STF_DET_L
14	0-7	-	STF_DET_U
15	0-7	-	Reserved

*1 The safety input signal is negative logic (Active Low).

If a signal level High is input to the safety input, this function will return 1 and the function assigned to the safety input will not operate.

If a signal level Low is input to the safety input, this function will return 0 and the function assigned to the safety input will operate. This occurs if a device is not connected to the safety input.

*2 The safety output signal is negative logic (Active Low).

If one of the functions assigned to the safety output is operating, the safety output will be enabled, the signal level at the safety output will be Low, and this function will return 0.
If none of the functions assigned to the safety output are operating, the safety output will be disabled, the signal level at the safety output will be High, and this function will return 1.
If no function is assigned to the safety output, the signal level at the safety output will be Low and this function will return 0.

When you use STF_ID, STF_DET_L, or STF_DET_H, it is possible to check the cause of an error.
How to check using SPEL+ will be described later.
The information for STF_ID, STF_DET_L, and STF_DET_H is as follows.

STF_ID	Error	STF_DET_L	STF_DET_U
100	Stop notification Safety input	Safety input port SAFETY_IN1 0×01 SAFETY_IN2 0×02 SAFETY_IN3 0×04 SAFETY_IN4 0×08 SAFETY_IN5 0×10	Not used
101	Stop notification SLS_1 speed exceeded Joint	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
102	Stop notification SLS_1 speed exceeded Part	Part Hand 0×01 / Elbow 0×02 *1 Wrist 0×04 / Shoulder 0×08 *1	Not used
103	Stop notification SLS_2 speed exceeded Joint	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
104	Stop notification SLS_2 speed exceeded Part	Part Hand 0×01 / Elbow 0×02 *1 Wrist 0×04 / Shoulder 0×08 *1	Not used
105	Stop notification SLS_3 speed exceeded Joint	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
106	Stop notification SLS_3 speed exceeded Part	Part Hand 0×01 / Elbow 0×02 *1 Wrist 0×04 / Shoulder 0×08 *1	Not used
107	Stop notification SLS_T speed exceeded Joint	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
108	Stop notification SLS_T speed exceeded Part	Part Hand 0×01 / Elbow 0×02 *1 Wrist 0×04 / Shoulder 0×08 *1	Not used
109	Stop notification SLS_T2 speed exceeded Joint	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
110	Stop notification SLS_T2 speed exceeded Part	Part Hand 0×01 / Elbow 0×02 *1 Wrist 0×04 / Shoulder 0×08 *1	Not used
115	Stop notification SLP_A position exceedance Monitored position	Monitored position YL 0×01 / YU 0×02 XL 0×04 / XU 0×08 ZL 0×10 / ZU 0×20 *2	Joint number J6 0×08 J5 0×04 J3 0×02 J2 0×01

STF_ID	Error	STF_DET_L	STF_DET_U
116	Stop notification SLP_B position exceedance Monitored position	Monitored position YL 0×01 / YU 0×02 XL 0×04 / XU 0×08 ZL 0×10 / ZU 0×20 *2	Joint number J6 0×08 J5 0×04 J3 0×02 J2 0×01
117	Stop notification SLP_C position exceedance Monitored position	Monitored position YL 0×01 / YU 0×02 XL 0×04 / XU 0×08 ZL 0×10 / ZU 0×20 *2	Joint number J6 0×08 J5 0×04 J3 0×02 J2 0×01
118	Stop notification Soft Axis Limiting	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
121	Stop notification Switch input	Switch number Enable switch 0×01 Emergency stop switch (Teach Pendant) 0×02 Emergency stop switch (Controller connection) 0×04	Not used
122	Stop notification Mode control	Mode Parameter communication permission 0×08 Disable Safety Function (Safety Board) 0×04 Switch Operation mode 0×02 Parameter setting verified 0×01	Not used
123	Stop notification Deceleration monitoring	Detection error Abnormal deceleration 0×08, 0×04 Deceleration completed 0×02 After the monitoring time 0×01	Not used
124	Stop notification Joint angle monitoring	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
131	Failure and stop notification Encoder communication failure	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
132	Failure and stop notification Position error	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
133	Failure and stop notification Double input error	Error detected part Safety input port SAFETY_IN1 0×01 SAFETY_IN2 0×02 SAFETY_IN3 0×04 SAFETY_IN4 0×08 SAFETY_IN5 0×10 Enable switch 0×20 Emergency stop switch (Teach Pendant) 0×40 Emergency stop switch (Controller connection) 0×80	Not used

STF_ID	Error	STF_DET_L	STF_DET_U
134	Failure and stop notification Double output error	Error detected part Safety output port SAFETY_OUT1 0×01 SAFETY_OUT2 0×02 SAFETY_OUT3 0×04 STO 0×80	Not used
135	Failure and stop notification Safety Board error	Error detected part Communication bus 0×20 Power (3.3V) 0×08 Power (5V) 0×04 Watchdog timer detection 0×02 Relay welding 0×01	Not used
136	Failure and stop notification Safety Board MCU error	Error detected part Sequence monitor 0×10 CPU 0×08 RAM 0×04 Program ROM 0×02 Data ROM 0×01	When DET_L = 0×01 (Date ROM) 0×00 - 0×FE Data failure part 0×FF Parameter failure
137	Failure and stop notification Safety Board Duplication internal abnormality	Error detected part TCP position not matched 0×02 Status not matched 0×01	Not used
138	Failure and stop notification Encoder internal abnormality	Joint number J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	Not used
139	Failure and stop notification Controller internal abnormality	Error detected part Operation mode receive error 0×01	Not used

*1 The correspondence between the monitored parts J2, J3, J5 for Safety Limited Speed in Safety Function Manager and the speed exceeded parts hand, wrist, elbow, and shoulder referred in this manual is as follows:

SCARA robots

J2: Elbow

J3: Not applicable

J5: Not applicable

Hand: Hand

6-axis robots

J2: Shoulder

J3: Elbow

J5: Wrist

Hand: Hand

*2 The correspondence between the monitored position X1, X2, Y1, Y2, Z1, Z2 for Safety Limited Position in Safety Function Manager and the monitored position XL, XU, YL, YU, ZL, ZU referred in this manual is as follows:

When "Wall" is selected for the monitored position

X1 = XL, X2 = XU

Y1 = YL, Y2 = YU

Z1 = ZL, Z2 = ZU (6-axis robots only)

When "Restricted Area" is selected for the monitored position

X1 = XU, X2 = XL

Y1 = YU, Y2 = YL

Z1 = ZL, Z2 = ZU (6-axis robots only)

- How to check the cause of an error using STF_ID, STF_DET_L and STF_DET_H

  Enter and confirm the commands in the following order in the command window and so on.

```
> SF_GetStatus (12)
115      'Indicates the error of "Stop notification SLP_A position exceedance".
> SF_GetStatus (13)
1        'Indicates that "YL" direction monitored position has been exceeded. (Check for STF_ID: 115.)
> SF_GetStatus (14)
1        'Indicates the "J2Flag" exceeded. (Check for STF_ID: 115.)
```

  To summarize the above, we can see that the error occurred because "the J2Flag cross the monitored position in the YL of SLP_A."

  Error information is also recorded in the EPSON RC+ system history. The cause of the error is recorded in "Additional information". See SPEL+ error messages in the SPEL+ Language Reference manual.

This command can be used with the Controllers with Safety Board.

SF_GetStatus Function Example

```
If (SF_GetStatus(3) And &H1) = &H1 Then
    Print "SAFETY_IN1 is High"
Else
    Print "SAFETY_IN1 is Low"
EndIf
```

SF_LimitSpeedS Statement

Sets and displays the speed adjustment value for the function that adjusts the speed at the position set by the Tool command when SLS is enabled.

Syntax

SF_LimitSpeedS [SLS number [, speed adjustment value]]

Parameters

SLS number The SLS number as an integer value (1 to 3) or the constant listed below. Optional.

Constant	Value	Description
SLS_1	1	Speed adjustment function when SLS_1 is enabled
SLS_2	2	Speed adjustment function when SLS_2 is enabled
SLS_3	3	Speed adjustment function when SLS_3 is enabled
SLS_T	9	Speed adjustment function when SLS_T is enabled* 1
SLS_T2	10	Speed adjustment function when SLS_T2 is enabled* 1

*1 For SLS_T and SLS_T2, you cannot set the speed adjustment value with this command. The speed is adjusted at the monitored speed set in Safety Function Manager. For more information, refer to the following manuals:
Robot Controller Safety Function Manual

Speed adjustment value	The speed as an integer value (0 to 10000, unit: mm/sec) Optional. If 0 is specified, the speed adjustment value is set automatically. The initial value is 0.

Description

Sets and displays the function that adjusts the speed of the position set by the Tool command when SLS is enabled. The part whose speed is adjusted by this function is the tool position currently selected by the Tool command. Note that this is not the TCP position set in the safety function parameters.

If the second parameter is omitted, the speed adjustment value of the specified SLS number will appear.
If all parameters are omitted, the speed adjustment value of all SLS numbers will appear.

This command can be used with the Controllers with Safety Board.
It is available only when Safety Function Options are activated.

Note

The position that is speed adjusted by SF_LimitSpeedS is the tool position selected by the Tool command.

The speed adjustment value set in SF_LimitSpeedS is applied to the speed at the tool position set by the Tool command. The Safety Function Manager TCP position is not automatically set in the Tool command. Use the Tool command to set the proper TCP location.
Also, if automatic speed adjustment does not function correctly, use SF_PeakSpeedS and SF_RealSpeedS to measure the robot arm operation speed. Use Speed, SpeedFactor, SpeedS, etc. to control the robot arm operation speed so that it does not exceed the SLS monitoring speed (safety function parameter).

SF_LimitSpeedS Example

- How to set the speed adjustment value for SLS_1 to 1500mm/sec

```
SF_LimitSpeedS SLS_1, 1500
```

- How to display the speed adjustment value for SLS_2 (using the command window)

```
> SF_LimitSpeedS SLS_2
SLS_2: 400
```

- How to display the speed adjustment value for all SLS numbers (using the command window)

```
> SF_LimitSpeedS
SLS_1: 1500
SLS_2: 400
SLS_3: 750
SLS_T: 250
SLS_T2: 3000
```

SF_LimitSpeedS Function

Returns the speed adjustment value of the function that adjusts the speed at the position set by the Tool command when SLS is enabled.

Syntax

SF_LimitSpeedS (SLS number)

Parameters

SLS number The SLS number as an integer value or the constant listed below.

Constant	Value	Description
SLS_1	1	Speed adjustment function when SLS_1 is enabled
SLS_2	2	Speed adjustment function when SLS_2 is enabled
SLS_3	3	Speed adjustment function when SLS_3 is enabled
SLS_T	9	Speed adjustment function when SLS_T is enabled
SLS_T2	10	Speed adjustment function when SLS_T2 is enabled

Return Values

Returns the speed adjustment value [mm/sec] for the specified SLS number.

Description

Returns the speed adjustment value [mm/sec] for the SLS number specified by the function that adjusts speed when SLS is enabled.

This command can be used with the Controllers with Safety Board.

SF_LimitSpeedS Function Example

```
Integer i
i = SF_LimitSpeedS(SLS_1)
Print "SLS_1 limit speed is ", i
```

SF_LimitSpeedSEnable Statement

Sets and displays On/Off for the function that adjusts the speed of the position set by the Tool command when SLS is enabled.

Syntax

SF_LimitSpeedSEnable [SLS number [, {On | Off}]]

Parameters

SLS number The SLS number as an integer value or the constant listed below. Optional.

Constant	Value	Description
SLS_1	1	Speed adjustment function when SLS_1 is enabled
SLS_2	2	Speed adjustment function when SLS_2 is enabled
SLS_3	3	Speed adjustment function when SLS_3 is enabled
SLS_T	9	Speed adjustment function when SLS_T is enabled* 1
SLS_T2	10	Speed adjustment function when SLS_T2 is enabled* 1

*1 For SLS_T and SLS_T2, you cannot set the speed adjustment function with this command. SLS_T turns on the speed adjustment function when the operation modes are TEACH and TEST T1. SLS_T2 turns on the speed adjustment function when the operation mode is TEST T2. For more information, refer to the following manuals:
Robot Controller Safety Function Manual

On | Off On/Off for the speed adjustment function. Optional. The default setting is On.

Return Values

None

Description

Sets and displays On/Off for the function that adjusts the speed at the position set by the Tool command when the specified SLS number is monitored. When On, the speed at the position set for the tool is adjusted when SLS is enabled. When Off, the speed is not adjusted even when SLS is enabled. Regardless of the On/Off setting, the speed at the tool position is not adjusted when SLS is disabled.

If the second parameter is omitted, the speed adjustment status (On/Off) of the specified SLS number will appear.
If all parameters are omitted, the speed adjustment status (On/Off) of all SLS numbers will appear.

This command can be used with the Controllers with Safety Board.
It is available only when Safety Function Options are activated.

SF_LimitSpeedSEnable Example

- How to enable the speed adjustment function for SLS_1

```
SF_LimitSpeedSEnable SLS_1, On
```

- How to display the speed adjustment function status for SLS_2 (using the command window)

```
> SF_LimitSpeedSEnable SLS_2
SLS_2: Off
```

- How to display the speed adjustment function status for all SLS numbers (using the command window)

```
> SF_LimitSpeedSEnable
SLS_1: On
SLS_2: Off
SLS_3: Off
SLS_T: On
SLS_T2: On
```

Note

SF_LimitSpeedSEnable cannot be used at the same time as singularity avoidance, conveyor tracking, and force control

SF_LimitSpeedSEnable is On by default. If you adjust the speed with this function during operation of singularity avoidance, conveyor tracking, or force control, an operation error (4093, 4403, or 5830 respectively) occurs.
When using SLS for these operations, set SF_LimitSpeedSEnable to Off before starting the operation. Also, use SF_PeakSpeedS and SF_RealSpeedS to measure the robot arm operation speed, and use Speed, SpeedFactor, SpeedS, etc. to control the robot arm operation speed so that it does not exceed the SLS monitored speed (safety function parameter).

Note

The position that is speed adjusted by SF_LimitSpeedS is the tool position set by the Tool command.

When SF_LimitSpeedSEnable is On, the applied speed adjustment value (set by SF_LimitSpeedS) is applied to the speed at the tool position set by the Tool command. The Safety Function Manager TCP position is not automatically set in the Tool command. Use the Tool command to set the proper TCP location.
Also, if automatic speed adjustment does not function correctly, use SF_PeakSpeedS and SF_RealSpeedS to measure the robot arm operation speed. Use Speed, SpeedFactor, SpeedS, etc. to control the robot arm operation speed so that it does not exceed the SLS monitoring speed (safety function parameter).

SF_LimitSpeedSEnable Function

Returns the status of the function that adjusts the speed at the position set by the Tool command when SLS is enabled.

Syntax

SF_LimitSpeedSEnable (SLS number)

Parameters

SLS number The SLS number as an integer value (1 to 3) or the constant listed below.

Constant	Value	Description
SLS_1	1	Speed adjustment function when SLS_1 is enabled
SLS_2	2	Speed adjustment function when SLS_2 is enabled
SLS_3	3	Speed adjustment function when SLS_3 is enabled
SLS_T	9	Speed adjustment function when SLS_T is enabled
SLS_T2	10	Speed adjustment function when SLS_T2 is enabled

Return Values

When the speed adjustment function for the specified SLS number is On, returns 1.
When the speed adjustment function for the specified SLS number is Off, returns 0.

Description

Returns the status for the function that adjusts the TCP operation speed when the specified SLS number is monitored.

This command can be used with the Controllers with Safety Board.

SF_LimitSpeedSEnable Function Example

```
If SF_LimitSpeedSEnable(SLS_1) = 0 Then
    Print "SLS_1 linked speed adjustment function is disabled."
Endif
```

SF_PeakSpeedS Statement

Displays the peak speed value for the speed monitoring point.

Syntax

SF_PeakSpeedS [Speed monitoring point number]

Parameters

Speed monitoring point number The speed monitoring point number as an integer value (1~4). Optional.

Value	Description
1	Hand
2	Elbow
3	Wrist
4	Shoulder

Return Values

None

Description

Displays the peak speed value for the specified speed monitoring point.
If the parameter is omitted, the peak speed value for all speed monitoring points will appear.
The speed of the hand is the speed at the TCP offset position set in Safety Function Manager.

This command can be used with the Controllers with Safety Board.

SF_PeakSpeedS Example

- How to display the peak speed value of the hand (using the command window)

```
> SF_PeakSpeedS 1
250
```

- How to display the peak speed value of all speed monitoring points (using the command window)

```
> SF_PeakSpeedS
250          150
```

The order of display is as follows:

Hand Elbow

SF_PeakSpeedS Function

Returns the peak speed of the speed monitoring point.

Syntax

SF_PeakSpeedS (Speed monitoring point number)

Parameters

Speed monitoring point number The speed monitoring point number as an integer value (1~4). Optional.

Value	Description
1	Hand
2	Elbow
3	Wrist
4	Shoulder

Return Values

Returns the peak speed [mm/sec].

Description

Returns the peak speed of the specified speed monitoring point.
The speed of the hand is the speed at the TCP offset position set in Safety Function Manager.

This command can be used with the Controllers with Safety Board.

SF_PeakSpeedS Function Example

```
Print "Hand peak speed is ", SF_PeakSpeedS (1)
```

SF_PeakSpeedSClear Statement

Clears and initializes the peak speed value for the speed monitoring point.

Syntax

SF_PeakSpeedSClear [Speed monitoring point number 1 [, Speed monitoring point number 2]]

Parameters

Speed limiting point number 1 The first speed limiting point number as an integer value (1~4). Optional.

Speed limiting point number 2 The second speed limiting point number as an integer value (1~4). Optional.

Value	Description
1	Hand
2	Elbow
3	Wrist
4	Shoulder

Return Values

None

Description

Clears (initializes) the peak speed value for the specified speed monitoring point.

If no parameters are specified, the peak speed values for all speed monitoring points are cleared.

This command can be used with the Controllers with Safety Board.

SF_PeakSpeedSClear Example

- How to clear the peak speed of the hand

```
SF_PeakSpeedSClear 1
```

- How to clear the peak speed values for all speed monitoring points

```
SF_PeakSpeedSClear
```

SF_RealSpeedS Statement

Displays the current speed of the speed monitoring point.

Syntax

SF_RealSpeedS [Speed monitoring point number]

Parameters

Speed monitoring point number The speed monitoring point number as an integer value (1~4). Optional.

Value	Description
1	Hand
2	Elbow
3	Wrist
4	Shoulder

Return Values

None

Description

Displays the current speed [mm/sec] of the specified speed monitoring point.
The speed of the hand is the speed at the TCP offset position set in Safety Function Manager.
If the parameter is omitted, the current speed of all speed monitoring points will appear.

This command can be used with the Controllers with Safety Board.

SF_RealSpeedS Example

- How to display the current speed of the hand (using the command window)

```
> SF_RealSpeedS 1
250
```

- How to display the current speed of all speed monitoring points (using the command window)

```
> SF_RealSpeedS
250          150
```

The order of display is as follows:

Hand Elbow

SF_RealSpeedS Function

Returns the current speed of the speed monitoring point.

Syntax

SF_RealSpeedS (Speed monitoring point number)

Parameters

Speed monitoring point number The speed monitoring point number as an integer value (1~4). Optional.

Value	Description
1	Hand
2	Elbow
3	Wrist
4	Shoulder

Return Values

Returns the current speed [mm/sec].

Description

Returns the current speed [mm/sec] of the specified speed monitoring point.
The speed of the hand is the speed at the TCP offset position set in Safety Function Manager.

This command can be used with the Controllers with Safety Board.

SF_RealSpeedS Function Example

```
Print "Hand real speed is ", SF_RealSpeedS (1)
```

SFree Statement

State free the specified servo axis.

Syntax

SFree *jointNumber* [*, jointNumber,...*]

Parameters

jointNumber	An integer expression representing a servo joint number (1 to 9). The additional S axis is 8 and T axis is 9.

Description

SFree removes servo power from the specified servo joints. This instruction is used for the direct teaching or the part installation by state free joint the specified joint. To release the free joint state, execute the SLock instruction, Motor On or Motor Off.

SFree initializes the robot control parameter.
See Motor On for the details.

Note

SFree Sets Some System Items back to Their Initial State:

SFree, for safety purposes, initializes parameters concerning the robot arm speed or acceleration (Speed ,SpeedS, Accel, AccelS etc.), and the LimZ parameter. For details, refer to the Motor On.
Firmware version earlier than 7.5.1.0, SFree is available with only Motor ON state.
SFree changes the motion as following below depends to the firmware version.

Firmware	SFree Available/ Not available
Before 7.5.1.0	Only when motor is on
After 7.5.1.0	When motor is on, or motor is off

Important

SFree and its Use with the Z Joint and U Joint for SCARA robots (including RS series)

The Z joint has electromagnetic 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.
Some model has electronic brake on the U joint. When the robot has the U joint electronic brake, setting SFree for the U joint does not immediately allow the U joint to be moved. To move the U joint by hand requires the brake to be released continuously by pressing the brake release switch on the top of the robot arm.

SFree is Not Valid with 6-Axis robots (including N series)

When SFree is executed in 6-axis robots (including N series), an error occurs.
To move the arm by hands, release the electromagnetic brake by using Brake Off after tuning OFF the motor by Motor Off.

Executing motion commands while joints are in free joint state

Attempting to execute a motion command while in the free joint condition will cause an error in the Controller's default state. However, to allow motion while 1 or more of the joints are in the free joint state, select the [Allow motion with one or more joints free] checkbox from [Setup]-[System Configuration]-[Controller]-[Preferences].

Do not use SFree during Conveyor Tracking

Error 5057 or 5058 might occur if SFree is used during conveyor tracking. Use SFree after terminating conveyor tracking such as Cnv_AbortTrack.

See Also

Brake, LimZ, Motor, SFree Function, SLock

SFree Statement Example

This is a simple example on the usage of the SFree command. To operate the robot in this exemple, the [Allow motion with one or more joints free] checkbox must be selected from [Setup]-[System Configuration]-[Controller]-[Preferences].

```
Function GoPick
   Speed pickSpeed
   SFree 1, 2       'State J1 and J2 to free joint
                    'and control the Z and U joints for part installation.
   Go pick
   SLock 1, 2       'Release the free joint state of J1 and J2.
Fend
```

SFree Function

Returns free joint status for a specified joint.

Syntax

SFree(*jointNumber*)

Parameters

jointNumber	Integer expression representing the joint number to check. The additional S axis is 8 and T axis is 9.

Return Values

True if the joint is free joint, False if not.

See Also

SFree Statement

SetFree Statement Example

```
If SFree(1) Then
    Print "Joint 1 is free"
EndIf
```

Sgn Function

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 command window example on the usage of the Sgn function.

```
>print sgn(123)
 1
>print sgn(-123)
 -1
>
```

Short Statement

Declares variables of Short type. (2 byte integer variable).

Syntax

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

Parameters

varName	Variable name which the user wants to declare.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

Short is used to declare variables as type integer. Integer variables can contain 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, Int32, Int64, Integer, Long, Real, String, UByte, UInt32, UInt64, UShort

Short Statement Example

The following example shows a simple program that declares some variables using Short.

```
Function shorttest
    Short A(10)          'Single dimension array of Short
    Short B(10, 10)      'Two dimension array of Short
    Short C(5, 5, 5)     'Three dimension array of Short
    Short 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
```

ShutDown Statement

Shuts down EPSON RC+ and optionally shuts down or restarts Windows.

Syntax

ShutDown [*mode*] [, *Forced*]

Parameters

mode	Optional. An integer expression that represents the mode setting described below.

Symbolic constant	Value	Meaning
Mode omitted	–1	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+.

Forced	Optional. Use to force a shutdown.

Description

Use ShutDown to shut down RC+ and optionally shutdown or reboot Windows from your program.
You can force a shutdown by using the Forced parameter.

Note

If you shutdown with the Forced parameter while tasks are running, you could lose data.

Be sure to save data before shutdown.

About the Controllers to use

Shutdown Statement cannot be used with T/VT series.

When the controller configuration is set to “Cooperative mode” in virtual controller

When the controller configuration is set to “Cooperative mode” in the virtual controller, the global Preserve variables will not be saved if ShutDown statement is executed. Do not use ShutDown statement if you need to save the global Preserve variables.

See Also

Restart

ShutDown Statement Example

```
ShutDown 0      ' Shutdown EPSON RC+ and Windows
```

ShutDown Function

Shuts down EPSON RC+ and optionally shuts down or restarts Windows.

Syntax

ShutDown ([*mode*] [, Forced])

Parameters

mode	Optional. An integer expression that represents the mode setting described below.

Symbolic constant	Value	Meaning
Mode omitted	−1	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+.

Forced	Optional. Use to force a shutdown.

Description

Use ShutDown to shut down RC+ and optionally shutdown or reboot Windows from your program.
You can force a shutdown by using the Forced parameter.

Note

If you shutdown with the Forced parameter while tasks are running, you could lose data.

Be sure to save data before shutdown.

About the Controllers to use

Shutdown Function cannot be used with T/VT series.

When the controller configuration is set to “Cooperative mode” in virtual controller

When the controller configuration is set to “Cooperative mode” in the virtual controller, the global Preserve variables will not be saved if ShutDown function is executed. Do not use ShutDown function if you need to save the global Preserve variables.

Return Values

Returns the following integer values.

-1	When a dialog is displayed and the user selects Cancel.
0	If shutdown fails
1	If shutdown is successful

ShutDown Function Example

```
If Shutdown(SHUTDOWN_EPSONRC) = 1 Then
    Print "Shutdown: OK"
Else
    Print "Shutdown: NG"
EndIf
```

Signal Statement

Send a signal to tasks executing WaitSig.

Syntax

Signal *signalNumber*

Parameters

signalNumber	Signal number to transmit. Range is 0 to 63.

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
```

SimGet Statement

Acquire the setting values of each object properties of simulator.

Syntax

SimGet Object.Property, Var
SimGet Robot.Hand.Propoerty, Var

Parameters

Object	String variable that indicates object names acquiring the property values.
Robot	String variable that indicates the robot name which the hand specified by “Hand” is installed.
Hand	String variable that indicates the hand name which acquires the property values.
Property	Property name that acquires values. Descriptions of properties are described later.
Var	Variable that indicates return value.

Description

Use this command to acquire the property setting value of each object of simulator.

Set the following properties to acquire the object setting values.

Property	Descriptions	Unit	Data type	Return value
PositionX	Acquire a position of X coordinate system.	(mm)	Double	
PositionY	Acquire a position of Y coordinate system.	(mm)	Double	
PositionZ	Acquire a position of Z coordinate system.	(mm)	Double	
RotationX	Acquire rotation angle of X axis.	(degree)	Double	
RotationY	Acquire rotation angle of Y axis.	(degree)	Double	
RotationZ	Acquire rotation angle of Z axis.	(degree)	Double	
CollisionCheck	Acquire enable/disable of collision detect.	-	Boolean	True or False
CollisionCheckSelf	Acquire enable/disable of self-collision detect of the robot.	-	Boolean	True or False
Visible	Acquire state of display/non-display.	-	Boolean	True or False
Type	Acquire the types of objects.	-	Integer	Layout: 0 Part: 1 Mounted Device: 3
HalfSizeX	Acquire a length of Box object in the X direction.	(mm)	Double	
HalfSizeY	Acquire a length of Box object in the Y direction.	(mm)	Double	
HalfSizeZ	Acquire a length of Box object in the Z direction.	(mm)	Double	
HalfSizeHeight	Acquire a length of Plane object.	(mm)	Double	
HalfSizeWidth	Acquire a width of Plane object.	(mm)	Double	
PlaneType	Acquire a type of Plane object.	-	Integer	Horizontal: 0 Vertical: 1
Radius	Acquire a radius of Sphere object or Cylinder object.	(mm)	Double	
Height	Acquire a height of Cylinder object.	(mm)	Double	

Property	Descriptions	Unit	Data type	Return value
Name	Acquire an object name.		String	
Color	Acquire a display color of an object.		String	Color name or hexadecimal color code (ARGB)

You can acquire the properties by combinations shown in the list below.

Property	Object							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
PositionX	✓	✓	✓	✓	✓	✓	✓	✓
PositionY	✓	✓	✓	✓	✓	✓	✓	✓
PositionZ	✓	✓	✓	✓	✓	✓	✓	✓
RotationX	✓	✓	✓	✓	✓	✓	✓	✓
RotationY	✓	✓	✓	✓	✓	✓	✓	✓
RotationZ	✓	✓	✓	✓	✓	✓	✓	✓
CollisionCheck	✓	✓	✓	✓	✓	✓	✓	✓
CollisionCheckSelf	✓	-	-	-	-	-	-	-
Visible	-	✓	✓	✓	✓	✓	✓	✓
Type	-	-	✓	✓	✓	✓	✓	-
HalfSizeX	-	-	✓	-	-	-	-	-
HalfSizeY	-	-	✓	-	-	-	-	-
HalfSizeZ	-	-	✓	-	-	-	-	-
HalfSizeHeight	-	-	-	-	-	✓	-	-
HalfSizeWidth	-	-	-	-	-	✓	-	-
PlaneType	-	-	-	-	-	✓	-	-
Radius	-	-	-	✓	✓	-	-	-
Height	-	-	-	-	✓	-	-	-
Name	✓	✓	✓	✓	✓	✓	✓	✓
Color	-	-	✓	✓	✓	✓	-	-

See Also

SimSet

SimGet Statement Example

```
'Acquire X coordinate value of SBox_1 object
Double boxPosX
SimGet SBox_1.PositionX, boxPosX

'Acquire the state of display/non-display of SBox_1 object
Boolean boxVisible
SimGet SBox_1.Visible, boxVisible

'Acquire the type of SBox_1 object
Integer boxType
SimGet SBox_1.Type, boxType
```

SimSet Statement

Set properties of each object of simulator. Operate the robot motion, objects, and simulator settings.

Syntax

(1) Property setting for object

SimSet Object.Property, Value
SimSet Robot.Hand.Property, Value

(2) Motion settings for robot (Pick & Place)

SimSet Robot.**Pick**, Object [,Tool]
SimSet Robot.**Place**, Object

(3) Operation settings for objects (specify the parent object)

SimSet Object.**SetParent** [, ParentObject]

(4) Simulator settings (reset the collision detect)

SimSet ResetCollision

Parameters

(1) Property setting for object

Object	String variable that indicates object names setting the property values.
Robot	String variable that indicates the robot name which the hand specified by "Hand" is installed.
Hand	String variable that indicates hand name which sets the property values.
Property	Property name that sets values. Descriptions of properties are described later.
Value	Formula with new values. Data type depends on properties.

(2) Motion settings for robot (Pick & Place)

Robot	String variable that indicates the robot name to Pick or Place.
Object	String variable that indicates the object name to be Picked or Placed.
Tool	Formula that indicates Tool number which is used at the time of Picking.

(3) Operation settings for object (specify the parent object)

Object	String variable that indicates the object name which sets the parent object.
ParentObject	String variable that indicates the parent object name.

Description

Use this command to set properties of each object of simulator. Also, use the command to change the the robot motion, objects, and simulator settings.

(1) Property setting for object
You can set the objects by specifying the properties shown below.

Property	Descriptions	Unit	Data type	Return value
PositionX	Set a position of X coordinate.	(mm)	Double	Max: 100000 Min: -100000
PositionY	Set a position of Y coordinate.	(mm)	Double	Max: 100000 Min: -100000
PositionZ	Set a position of Z coordinate.	(mm)	Double	Max: 100000 Min: -100000
RotationX	Set rotation angle of X axis.	(degree)	Double	Max: 360 Min: -360
RotationY	Set rotation angle of Y axis.	(degree)	Double	Max: 360 Min: -360
RotationZ	Set rotation angle of Z axis.	(degree)	Double	Max: 360 Min: -360
CollisionCheck	Set enable/disable of collision detect.	-	Boolean	True or False
CollisionCheckSelf	Set enable/disable of self-collision detect of the robot.	-	Boolean	True or False
Visible	Set state of display/non-display.	-	Boolean	True or False
HalfSizeX	Acquire a length of Box object in the X direction.	(mm)	Double	Max: 100000 Min: 0.001
HalfSizeY	Acquire a length of Box object in the Y direction.	(mm)	Double	Max: 100000 Min: 0.001
HalfSizeZ	Acquire a length of Box object in the Z direction.	(mm)	Double	Max: 100000 Min: 0.001
HalfSizeHeight	Acquire a length of Plane object.	(mm)	Double	Max: 100000 Min: 0.001
HalfSizeWidth	Acquire a width of Plane object.	(mm)	Double	Max: 100000 Min: 0.001
PlaneType	Acquire a type of Plane object.	-	Integer	Horizontal: 0 Vertical: 1
Radius	Acquire a radius of Sphere object or Cylinder object.	(mm)	Double	Max: 100000 Min: 0.001
Height	Acquire a height of Cylinder object.	(mm)	Double	Max: 100000 Min: 0.001
Name	Acquire an object name.		String	
Color	Acquire a display color of an object.		String	Color name or hexadecimal color code (ARGB)

You can set the property by combinations shown in the list below.

Property	Objects							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
PositionX	✓	✓	✓	✓	✓	✓	✓	✓
PositionY	✓	✓	✓	✓	✓	✓	✓	✓
PositionZ	✓	✓	✓	✓	✓	✓	✓	✓
RotationX	✓	✓	✓	✓	✓	✓	✓	✓
RotationY	✓	✓	✓	✓	✓	✓	✓	✓
RotationZ	✓	✓	✓	✓	✓	✓	✓	✓
CollisionCheck	✓	✓	✓	✓	✓	✓	✓	✓
CollisionCheckSelf	✓	-	-	-	-	-	-	-
Visible	-	✓	✓	✓	✓	✓	✓	✓
HalfSizeX	-	-	✓	-	-	-	-	-
HalfSizeY	-	-	✓	-	-	-	-	-
HalfSizeZ	-	-	✓	-	-	-	-	-
HalfSizeHeight	-	-	-	-	-	✓	-	-
HalfSizeWidth	-	-	-	-	-	✓	-	-
PlaneType	-	-	-	-	-	✓	-	-
Radius	-	-	-	✓	✓	-	-	-
Height	-	-	-	-	✓	-	-	-
Name	✓	✓	✓	✓	✓	✓	✓	✓
Color	-	-	✓	✓	✓	✓	-	-

(2) Motion settings for robot (Pick & Place)
You can set the following robot motions.

Pick

The robot specified by "Robot" grasps the object specified by "Object".
Grasped object is registered as the part of the robot. Also, if any tool number is specified to "Tool", you can operate grasped motion by using the specified tool number. If the "Tool" settings are omitted, use Tool0 to operate grasped motion.
You cannot grasp the object that is already registered as the part or set as an arm installation tool. Also, you cannot grasp the camera.

Place

The robot specified by "Robot" places the object specified by "Object". The placed object is deregistered as the part of the robot.
You cannot place the objects which registrations are already deregistered.

You can grasp or place the object by combinations shown in the list below.

Motion	Objects							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
Pick	-	-	✓	✓	✓	✓	✓	-
Place	-	-	✓	✓	✓	✓	✓	-

(3) Operation settings for object (specify the parent object)
You can set operations for the following objects

SetParent
Set the object specified by “ParentObject” as the parent object for the object specified by “Object”. “ParentObject” can be omitted. In that case, the object specified by “Object” will be the parent object. If the object specified by “Object” is a child object of some object, the setting as the child object is deregistered.
If the object specified by “Object” is registered as part or arm installation tool, you cannot specify the parent object.

The objects that can specify the SetParent are as follows. For the camera object, only the object set as a fixed camera can use the SetParent.

Operation	Objects							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
SetParent	-	-	✓	✓	✓	✓	✓	✓

You can use SetParent by combinations shown in the list below.

		ParentObject							
		Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
Child Object	Robot	-	-	-	-	-	-	-	-
	Hand	-	-	-	-	-	-	-	-
	Box	-	-	✓	✓	✓	✓	✓	✓
	Sphere	-	-	✓	✓	✓	✓	✓	✓
	Cylinder	-	-	✓	✓	✓	✓	✓	✓
	Plane	-	-	✓	✓	✓	✓	✓	✓
	CAD	-	-	✓	✓	✓	✓	✓	✓
	Camera	-	-	✓	✓	✓	✓	✓	✓

(4) Simulator settings (reset the collision detect)
You can change the following simulator settings.

ResetCollision
Reset the collision detect. If the robot and the object do not collide after executing ResetCollision, reset the collision state and update the 3D display on the simulator. If the robot and the object collide, the collision state does not be reset and 3D display of the simulator will not be updated.

See Also
SimGet

SimSet Statement Example

```
'Set the X coordinate value of SBox_1 object to 100.0mm
SimSet SBox_1.PositionX, 100.0

'Grasp SBox_1 by Tool1 in Robot1
SimSet Robot1.Pick, SBox_1, 1

'Place SBox_1 grasped by Robot1
SimSet Robot1.Place, SBox_1

'Set CAD_1 to the parent object of SBox_1
SimSet SBox_1.SetParent, CAD_1

'Set SBox_1 as the parent object
SimSet SBox_1.SetParent

'Reset the collision detect
SimSet ResetCollision
```

Sin Function

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
```

SingularityAngle Statement

Sets the singularity neighborhood angle necessary for the singularity avoiding function.

Syntax

SingularityAngle {*angle*}

Parameters

angle	Specify the Joint #5 angle (real number equals to or greater than 0.1. Unit: deg) by a formula or a value for determining the wrist singularity neighborhood of the vertical 6-axis robot (including N series).

Result

Current SingularityAngle value will be displayed if the parameter is omitted.

Description

This command is enabled only when the singularity avoiding function is being used.
Default is 10 deg. This command can be used to adjust the start position of the singularity avoidance. If the value smaller than the default is specified, avoidance motion starts at the point closer to the singularity. Usually, it is not necessary to change the parameter. This may be useful to reduce errors which occur when passing the singularity.

If SingularityAngle parameter is changed, the current setting is effective until the next controller startup.

See Also

AvoidSingularit, SingualrityAngle Function, SingularitySpeed

SingularityAngle Statement Example

```
    SingularityAngle 7.0 'Sets the singularity neighborhood angle at 7 degrees
```

SingularityAngle Function

Returns the SingularityAngle setting value.

Syntax

SingularityAngle

Return Values

Returns the singularity neighborhood angle (Unit: deg).

See Also

AvoidSingularity, SingularityAngle, SingularitySpeed, SingularitySpeed Function

SingularityAngle Function Example

```
Real currSingularityAngle
currSingularityAngle = SingularityAngle
```

SingularityDist Statement

Sets the singularity neighborhood distance necessary for the singularity avoiding function.

Syntax

SingularityDist {*distance*}

Parameters

distance	Specify the distance between the point P and Joint #1 rotation axis (real number equals to or larger than 0. Unit: mm) by a formula or a value for determining the shoulder singularity neighborhood or f the vertical 6-axis robot (including N series) and RS series.

Result

Current SingularityDist value will be displayed if the parameter is omitted.

Description

This command is enabled only when the singularity avoiding function is being used.
Default is 30 mm. This command can be used to adjust the start position of the singularity avoidance. If the value smaller than the default is specified, avoidance motion starts at the point closer to the singularity. Usually, it is not necessary to change the parameter. This may be useful to reduce errors which occur when passing the singularity.

If SingularityDist parameter is changed, the current setting is effective until the next controller startup.

See Also

AvoidSingularity, SingularityAngle, SingualrityAngle Function, SingularityDist Function, SingularitySpeed, SingularitySpeed Function

SingularityDist Statement Example

```
SingularityDist 10.0 'Sets the singularity neighborhood distance at 10 mm
```

SingularityDist Function

Returns the SingularityDist setting value.

Syntax

SingularityDist

Return Values

Returns the singularity neighborhood distance (Unit: mm).

See Also

SingularityDist, AvoidSingularity, SingularityAngle, SingularityAngle Function, SingularitySpeed, SingularitySpeed Function

SingularityDist Function Example

```
Real currSingularityDist
currSingularityDist = SingularityDist
```

SingularitySpeed Statement

Sets the singularity neighborhood angular velocity necessary for the singularity avoiding function.

Syntax

SingularitySpeed {*Angular velocity*}

Parameters

Angular velocity	Specify the percentage of the Joint #4 angular velocity with respect to the maximum angular velocity (real number equals to or greater than 0.1. Unit: %) by a formula or a value for determining the wrist singularity neighborhood of the vertical 6-axis robot (including N series).

Result

Current SingularitySpeed value will be displayed if the parameter is omitted.

Description

This command is enabled only when the singularity avoiding function is being used.
Default is 10%. This command can be used to adjust the start position of the singularity avoidance. If the value smaller than the default is specified, avoidance motion starts at the point closer to the singularity. Usually, it is not necessary to change the parameter. This may be useful to reduce errors which occur when passing the singularity.

If SingularitySpeed parameter is changed, the current setting is effective until the next controller startup.

See Also

AvoidSingularity Function, SingualrityAngle, SingularitySpeed

SingularitySpeed Example

```
SingularitySpeed 30.0 'Sets the singularity neighborhood angular velocity at 30%
```

SingularitySpeed Function

Returns the SingularitySpeed setting value.

Syntax

SingularitySpeed

Return Values

Returns the singularity neighborhood angular velocity (Unit: %).

See Also

SingularitySpeed, SingularityAngle, AvoidSingularity

SingularitySpeed Function Example

```
Real currSingularitySpeed
currSingularitySpeed = SingularitySpeed
```

SLock Statement

Release the free joint state for the specified servo axis.

Syntax

SLock *jointNumber* [*, jointNumber,...*]

Parameters

jointNumber	The servo joint number (1 to 9). The additional S axis is 8 and T axis is 9.

Description

SLock release the free joint state which was free joint state by the SFree instruction for the direct teaching or part installation.

If the joint number is omitted, all joints are released free joint.

Executing SLock the 3rd joint (Z) causes the brake to release.

Executing SLock while in Motor Off state will cause an error.

SLock initializes the robot control parameter.
See Motor On for the details.

6-axis robots (including N series) cannot be free joint state by the SFree instruction. When SLock is executed, an error occurs.

See Also

Brake, LimZ, Reset, SFree

SLock Example

This is a simple example on the usage of the SLock command. To operate the robot in this exemple, the [Allow motion with one or more joints free] checkbox must be selected from [Setup]-[System Configuration]-[Controller]-[Preferences].

```
Function test
   .
   .
   .
   SFree 1, 2     'State J1 and J2 to free joint
                  'and control the Z and U joints for part installation.
   Go P1
   SLock 1, 2     'Release the free joint state of J1 and J2.
   .
   .
   .
Fend
```

SoftCP Statement

Specifies the SoftCP motion mode.

Syntax

SoftCP { **On** | **Off** }

Parameters

On | Off	On is used to enable SoftCP motion mode. Off is used to disable SoftCP motion mode.

Description

SoftCP motion mode controls the vibration caused by CP motion with high acceleration/deceleration.
Normal CP motion focuses on path-tracking and uniform-motion which increases the vibration when acceleration/deceleration is high. To reduce the vibration, acceleration/deceleration needs to be reduced with the SpeedS and AccelS commands.
However, some applications don't necessarily require the high performance of path-tracking and uniform-motion but need CP motion with less vibration when acceleration/deceleration is high.
SoftCP motion mode dampens the path-tracking and uniform-motion performance more than in the normal CP motion mode and reduces the vibration in CP motion with high acceleration/deceleration.

SoftCP motion mode applies to the following CP motion commands:
Move, BMove, TMove, Arc, Arc3, CVMove, Jump3CP

If the vibration doesn't matter in the normal CP motion or the performances of path-tracking and uniform-motion are required, don't apply SoftCP motion mode.

Caution

When connection CP motion and PTP motion in CP On

When connecting CP motion and PTP motion as shown below, be sure to enable SoftCP.
If it is not enabled, noise may occur from the robot depending on the motion. After connecting CP motion and PTP motion, disable SoftCP.

```
SoftCP On
Go P1 CP
Move P2
SoftCP Off
```

See Also

SoftCP Function

SoftCP Statement Example

```
SoftCP On
Move P1
Move P2
SoftCP Off
```

SoftCP Function

Returns the status of SoftCP motion mode.

Syntax

SoftCP

Return Values

0 = SoftCP motion mode off, 1 = SoftCP motion mode on.

See Also

SoftCP Statement

SoftCP Function Example

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

Space$ Function

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

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 and 100 representing the arm speed as a percentage of the maximum speed.
departSpeed	Integer expression between 1 and 100 representing the depart motion speed for the Jump instruction. Available only with Jump command.
approSpeed	Integer expression between 1 and 100 representing the approach motion speed for the Jump instruction. Available only with Jump command.

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:

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

In Low Power Mode, the effective speed setting is lower than the default value. If a higher speed is specified directly (from the command 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 command 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 command 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

Returns one of the three speed settings.

Syntax

Speed[(*paramNumber*)]

Parameters

paramNumber	Integer expression which evaluates to one of the values shown below. When omitted, 1 will be taken as the specified number. 1: PTP motion speed 2: Jump depart speed 3: Jump approach speed

Return Values

Integer value from 1 to 100.

See Also

Speed

Speed Function Example

```
    Integer savSpeed

    savSpeed = Speed(1)
    Speed 50
    Go pick
    Speed savSpeed
Fend
```

SpeedFactor Statement

Sets and returns the setting value of speed factor for manipulator motions.

Syntax

(1) **SpeedFactor** *speedRatio*
(2) **SpeedFactor**

Parameters

speedRatio	Integer expression or value between 1 and 100 representing the speed ratio of manipulator motion. (Unit: %)

Return Values

Displays current SpeedFactor value when used without parameters.

Description

SpeedFactor specifies the speed factor for all manipulators and motions set to the Controller. Usually, SpeedFactor is set to 100 % and speed for each manipulator/motion command is set by Speed or SpeedR. SpeedFactor is useful to set specific speed to all motions of all manipulators at one time. For example, the motion with Speed = 80% operates at 40% of the speed, when speed ratio is 50%.

SpeedFactor also changes the acceleration at the same rate in consideration of a balance of acceleration and deceleration of the manipulator motion.

SpeedFactor is equivalent to the speed ratio setting in the operator window and changes along with the value.

SpeedFactor will be initialized to 100% at the Controller startup.

See Also

SpeedFactor Function

SpeedFactor Statement Example

```
Function main
    Motor On
    Power High
    SpeedFactor 80

    Speed 100; Accel 100,100
    Go P1            'Operates with Speed 80; Accel 80,80

    Speed 50; Accel 50,50
    Go P2            'Operates with Speed 40; Accel 40,40
Fend
```

SpeedFactor Function

Returns SpeedFactor setting value.

Syntax

SpeedFactor

Return Values

Integer value representing the SpeedFactor setting.

See Also

SpeedFactor Statement

Speed Function Example

```
Real savSpeedFactor

savSpeedFactor = SpeedFactor
SpeedFactor 80
Go P1
Go P2
SpeedFactor savSpeedFactor
```

SpeedR Statement

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. Valid entries range of the parameters: 0.1 to 1000

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:

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or QuitAll stops tasks

See Also

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

SpeedR Statement Example

```
SpeedR 200
```

SpeedR Function

Returns tool rotation speed value.

Syntax

SpeedR

Return Values

Real value in degrees / second.

See Also

AccelR, SpeedR

SpeedR Function Example

```
Real currSpeedR

currSpeedR = SpeedR
```

SpeedS Statement

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.

Valid entries range of the parameters:
Other than N series: 0.1 to 2000
N series : 0.1 to 1120

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. The default value varies from robot to robot. See the Manipulator 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:

Controller Startup
Motor On
SFree, SLock, Brake
Reset, Reset Error
Stop button or QuitAll stops tasks

In Low Power Mode, the effective SpeedS setting is lower than the default value. If a higher speed is specified directly (from the command 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.

See Also

AccelS, Arc, Jump3, Move, Speed

SpeedS Statement Example

SpeedS can be used from the command 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
        Move P0
        Move P1
        SpeedS fast
        Move P0
        Move P1
    Next i
Fend
```

From the command window the user can also set SpeedS values.

```
> speeds 1000
> speeds 500
> speed 30       'sets point to point speed
> go p0          'point to point move
> speeds 100     'sets straight line speed in mm/Sec
> move P1        'moves in straight line
```

SpeedS Function

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 Function Example

```
Real savSpeeds

savSpeeds = SpeedS

Print "Jump3 depart speed = ", SpeedS(2)
```

Sqr Function

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 Error

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 Command 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
```

ST Function

Returns the coordinate value of the specified additional axis in the point data.

Syntax

ST (*sValue* As Real, *tValue* As Real)

Parameters

sValue	Real value that specifies the S axis coordinate value
tValue	Real value that specifies the T axis coordinate value

Return Values

Coordinate values of the specified additional axis in the point data.

Description

This function is used when you are using the additional ST axes.
When using this function like Go ST(10,20), the additional axis will move to the specified coordinate but the manipulator will not move. If you want to move the manipulator as well, use like `Go XY(60,30,-50,45) : ST( 10,20)`.
For the details of the additional axis, refer to *EPSON RC+ Users Guide: 21. Additional Axis*.

See Also

XY Function

ST Function Example

```
    P10 = ST(10, 20)
```

StartMain Statement

Executes the main function from a background task.
This command is for the experienced user and you need to understand the command specification before use.

Syntax

StartMain *mainFuncname*

Parameters

mainFuncname	Main function name you want to execute (main ~ main63)

Description

To execute StartMain, you need to set the [Enable advanced task commands] preference in the [Setup]-[System Configuration]-[Controller]-[Preferences] page.

If a task is executed using the Xqt statement from a background task, the executed task becomes a background task. With StartMain, you can execute the main function as a non-background task from a background task.

If you have already executed the main function or execute StartMain from a non-background task, an error occurs.

⚠ CAUTION	■ When executing StartMain command from a program, you must understand the command specification and confirm that the system has the proper conditions for this command. Improper use such as continuous execution of a command within a loop may deteriorate the system safety.

See Also

Xqt

StartMain Statement Example

```
Function bgmain

      :
   If Sw(StartMainSwitch) = On And Sw(ErrSwitch) = Off Then
      StartMain main
   EndIf
      :

Fend
```

Stat Function

Returns the execution status information of the controller.

Syntax

Stat*(address)*

Parameters

address	Defines which status bits to check. (0 to 2)

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	0-15	&H1 to&H8000	Task (1~16) 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	Safeguard Input is Closed
	23	&H800000	Enable Switch is Open
	24	&H1000000	Undefined
	25	&H2000000	Undefined
	26	&H4000000	Test mode
	27	&H8000000	T2 mode
	28-31		Undefined
1	0	&H1	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	&H2	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	&H4	Undefined
	3	&H8	Log of stop at intermediate travel position upon satisfaction of condition in Trap statement
	4	&H10	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	&H4000	Axis T motor is on
	15	&H8000	Axis S motor is on
	16	&H10000	7th Joint motor is on
	17-31		Undefined
2	0-15	&H1 to &H8000	Task (17~32) is being executed (Xqt) or in Halt State

See Also

EStopOn Function, TillOn Function, PauseOn Function, SafetyOn Function

Stat Function 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

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 positive or negative number 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 command window.

```
> Print Str$(99999999999999)
 1.000000E+014

> Print Str$(25.999)
 25.999
```

String Statement

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 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	200
Global Preserve variable	400
Global variable and module variable	10,000

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`

Note

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, Int32, Int64, Integer, Long, Real, Short, UByte, UInt32, UInt64, UShort

String Statement Example

```
String password$
String A$(10)           'Single dimension array of string
String B$(10, 10)       'Two dimension array of string
String C$(5, 5, 5)      'Three dimension array of string

Print "Enter password:"
Input password$
If UCase$(password$) = "EPSON" Then
    Call RunMaintenance
Else
    Print "Password invalid!"
EndIf
```

Sw Function

Returns or displays the selected input port status. (i.e. Discrete User I/O)

Syntax

Sw(*bitNumber***)**

Parameters

bitNumber Integer expression representing I/O input bits.

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 command window:

```
> print sw(5)
1
>
```

SyncLock Statement

Synchronizes tasks using a mutual exclusion lock.

Syntax

SyncLock *syncID* [, *timeOut*]

Parameters

syncID	Integer expression representing signal number to receive. Range is from 0 to 63.
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 finished, then the lock it previously locked will releases.

When SynLock is second consecutive used to a same signal number, an error occurs.

If the *timeOut* parameter is used, then the Twcmd_tw function must be used to check if the lock was successful.

Note

In EPSON RC+ 6.0 and 7.0, the lock is automatically released when the task is finished while it is not in EPSON RC+5.0.

See Also

Signal, SyncLock, Tw, Wait, WaitPos

SyncLock Statement Example

The following example uses SyncLock and SyncUnlock to allow only one task at a time to write a message to a communication port.

```
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
  OpenCom #1
  Print #1, msg$
  CloseCom #1
  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

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 0 to 63.

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 Statement 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
```

SyncRobots Statement

Start the reserved robot motion.

Syntax

SyncRobots *robotNumber* [, *robotNumber*] [, ...]
SyncRobots All

Parameters

robotNumber	Integer expression that specifies a robot number you want to start the motion.
All	All robots whose motion is reserved

Description

SyncRobots is used to start the robot motion reserved with the *SYNC* parameter of each motion command. The robots specified by the SyncRobots start to move in the same timing. This is more useful than synchronizing the normal multi-task programs by waiting for the I/O signal event because there is no effect of switching tasks. It can synchronize the robot motion start more precisely.

If a robot number is specified whose motion is not reserved, an error occurs.

See Also

SyncRobots Function

SyncRobots Statement Example

The example below uses the *SYNC* parameter of a motion command and SyncRobots to start the motions of two robots simultaneously.

```
Function Main
    Xqt Func1
    Xqt Func2
    Do
       Wait 0.1
       If (SyncRobots And &H03) = &H03 Then
           Exit Do
       EndIf
    Loop
    SyncRobots 1,2
Fend

Function Func1
  Robot 1
  Motor On
  Go P1 SYNC
Fend

Function Func2
  Robot 2
  Motor On
  Go P1 SYNC
Fend
```

SyncRobots Function

Returns the status of a robot whose motion is reserved.

Syntax

SyncRobots

Return Values

Returns the robot motion in a bit, and if not reserved, "0" is returned.

bit 0: robotNumber 1
bit 1: robotNumber 2
:
bit 15: robotNumber 16

Description

SyncRobots function checks the motion reservation status of the *SYNC* parameter of the robot motion commands. The status the SyncRobots checks are displayed in the bit status corresponding to the robot number. Each bit shows either the robot motion is reserved (1) or not (2). You can start the robot motion reserved using the SyncRobots statement.

See Also

SyncRobots

SyncRobots Function Example

The example below uses the *SYNC* parameter of a motion command and SyncRobots to start the motions of two robots simultaneously.

```
Function Main
    Xqt Func1
    Xqt Func2
    Do
       Wait 0.1
       If (SyncRobots And &H03) = &H03 Then
          Exit Do
       EndIf
    Loop
    SyncRobots 1,2
Fend

Function Func1
  Robot 1
  Motor On
  Go P1 SYNC
Fend

Function Func2
  Robot 2
  Motor On
  Go P1 SYNC
Fend
```

SysConfig Statement

Displays system configuration parameter.

Syntax

SysConfig

Return Values

Returns system configuration parameter.

Description

Display current configured value for system control data. When the robot and controller is received from the factory or after changing the configuration, it is a good idea to save this data. This can be done with Backup Controller from the [Tools]-[Controller dialog].

The following data will be displayed. (The following data is for reference only since data will vary from controller to controller.)

```
' Version:
'   Firmware 1, 0, 0, 0

' Options:
'   External Control Point
'   RC+ API

' HOUR: 414.634

' Controller:
'   Serial #: 0001

' ROBOT 1:
' Name: Mnp01
' Model: PS3-AS10
' Serial #: 0001
' Motor On Time: 32.738
'   Motor 1: Enabled, Power = 400
'   Motor 2: Enabled, Power = 400
'   Motor 3: Enabled, Power = 200
'   Motor 4: Enabled, Power = 50
'   Motor 5: Enabled, Power = 50
'   Motor 6: Enabled, Power = 50

  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, 0, 0, 0, 0, 0
  HOFS 0, 0, 0, 0, 0, 0
  HORDR 63, 0, 0, 0, 0, 0
  RANGE -7427414, 7427414, -8738134, 2621440, -3145728, 8301227, -
5534152, 5534152, -3640889, 3640889, -6553600, 6553600
  BASE 0, 0, 0, 0, 0, 0
  WEIGHT 2, 0
  INERTIA  0.1, 0
  XYLIM 0, 0, 0, 0, 0, 0
```

```
' Extended I/O Boards:
'   1: Installed
'   2: Installed
'   3: None installed
'   4: None installed

' Fieldbus I/O Slave Board:
'   Installed
'   Type: PROFIBUS

' Fieldbus I/O Master Board:
'   None installed

' RS232C Boards:
'   1: Installed
'   2: None installed

' PG Boards:
'   1: None installed
'   2: None installed
'   3: None installed
'   4: None installed
```

SysConfig Statement Example

```
> SysConfig
```

SysErr Function

Returns the latest error status or warning status.

Syntax

SysErr [(infoNo)]

Parameters

infoNo	Optional. Integer number representing the error code or warning code to get. 0 : Error code (When the parameter is omitted, 0 is automatically selected.) 1 : Warning code

Return Values

An integer representing the error code or warning code of the controller.

Description

SysErr is used only for NoEmgAbort task (special task using NoEmgAbort at Xqt) and background tasks.
Error codes or warning codes of controller are the error codes or warning codes displayed on the LCD.
When there are no errors or warnings, the return value will be 0.

See Also

ErrMsg$, ErrorOn, Trap, Xqt

SysErr Function Example

The following example shows a program that monitors the controller error and switches the I/O On/Off according to the error number when error occurs.

Notes

Forced Flag

This program example uses Forced flag for On/Off command.
Be sure that the I/O outputs change during error, or at Emergency Stop or Safety Door Open when designing the system.

After Error Occurrence

As this program, finish the task promptly after completing the error handling.

```
Function main

   Xqt ErrorMonitor, NoEmgAbort
   :
   :
Fend

Function ErrorMonitor
   Wait ErrorOn
   If 4000 < SysErr Then
         Print "Motion Error = ", SysErr
         Off 10, Forced
         On 12, Forced
   Else
         Print "Other Error = ", SysErr
         Off 11, Forced
         On 13, Forced
   EndIf

Fend
```

Tab$ Function

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

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 Command window.

```
> print tan(0)
0.00
> print tan(45)
1.6197751905439
>
```

TargetOK Function

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

Returns the completion status of a task.

Syntax

TaskDone (*taskIdentifier*)

Parameters

taskIdentifier	Task name or integer expression representing the task number. Task name is a function name used in an Xqt statement or a function started from the Run window or Operator window. Task number range is: Normal tasks : 1 to 32 Background task : 65 to 80 Trap tasks : 257 to 267

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)
```

TaskInfo Function

Returns status information for a task.

Syntax

TaskInfo(*taskIdentifier, index*)

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. Specifying a task number: Normal tasks : 1 to 32 Background tasks : 65 to 80 Trap tasks : 257 to 267
index	Integer expression that represents the index of the information to retrieve.

Return Values

An integer containing the specified information.

Description

Index	Description
0	Task number
1	0 – Background task 1 – Normal task, NoPause task, or NoEmgAbort task
2	Task type 0 - Normal task Nothing specified in Xqt or start the task by Normal 1 - NoPause task Specified NoPause in Xqt and start the task 2 - NoEmgAbort task Specified NoEmgAbort in Xqt and start the task 3 - Trap task 4 - Background task
3	−1 - Specified task is not executing. 1 - Specified task is executing. 2 - Specified task is waiting for an event. 3 - Specified task is paused or halted 4 - Specified task is in quick pause state 5 - Specified task is in error state
4	Timeout has occurred during wait for event (same as TW)
5	Event wait time (milliseconds).
6	Current robot number selected by the task
7	Current robot number being used by the task

See Also

CtrlInfo, RobotInfo, TaskInfo$

TaskInfo Function Example

```
If (TaskInfo(1, 3) <> 0 Then
  Print "Task 1 is running"
Else
  Print "Task 1 is not running"
EndIf
```

TaskInfo$ Function

Returns text information for a task.

Syntax

TaskInfo$(*taskIdentifier, index*)

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. Specifying a task number: Normal tasks : 1 to 32; Background tasks : 65 to 80; Trap tasks : 257 to 267
index	Integer expression that represents the index of the information to retrieve.

Return Values

A string containing the specified information.

Description

The following table shows the information that can be retrieved using TaskInfo$:

Index	Description
0	Task name
1	Start date / time
2	Name of function currently executing
3	Line number in the program file that contains the function

See Also

CtrlInfo, RobotInfo, TaskInfo

TaskInfo$ Function Example

```
Print "Task 1 started: "TaskInfo$(1, 1)
```

TaskState Function

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. Specifying a task number: Normal tasks : 1 to 32 Background tasks : 65 to 80 Trap tasks : 257 to 267

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

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

Waits to for a task to terminate.

Syntax

TaskWait (*taskIdentifier*)

Parameters

taskIdentifier	Task name or integer expression representing the task number. Task name is a function name used in an Xqt statement or a function started from the Run window or Operator window. Task number range is: Normal tasks : 1 to 32 Background task : 65 to 80 Trap tasks : 257 to 267

See Also

TaskDone, TaskState

TaskWait Statement Example

```
Xqt 2, conveyor
TaskWait conveyor
```

TC Statement

Returns the torque control mode setting and current mode.

Syntax

(1) **TC** { On | Off }
(2) **TC**

Parameters

On | Off	On : Torque control mode ON Off : Torque control mode OFF

Return Values

When the parameter is omitted, returns the current torque control mode.

Description

TC On/Off set the torque control mode available/unavailable.
The torque control mode sets the motor output limit to generate the constant force. This is used in pressing a hand to an object at constant force or making the close contact and coordinate moving of hand with an object .
Before setting the torque control available, configure the limits of torque control in TCLim.
Under the torque control, the robot moves as positioning to the target while an operation command is executed. When the robot contact an object and motor output is at the torque control limit, the robot stops its operation and keeps the constant torque.

In any of the following cases, the torque mode turns unavailable.

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or QuitAll stops tasks

See Also

TCLim, TCSpeed

TC Statement Example 1

```
Speed 5
Go ApproachPoint

'Set the Z axis torque limit to 20 %
TCLim -1, -1, 20, -1

TC On
Go ContactPoint TargetPoint
Wait 3
Go ApproachPoint
TC Off
```

Notes

When a position error detected on Safety Board mounted controller

Change the program so that "current position of the robot" and "current target position of the robot" are not far apart.
If they are far apart, the Safety Board detects failure, and "Erorr No.9801 Detected a position error by the Safety Board." error occurs. (Only for controllers mounting the Safety Board.)

The current position of the robot can be required by RealPos function.
The current target position of the robot can be required by CurPos function.
When using SCARA robot, it is recommended that the difference between RealPos and CurPos be less than about J1:10 deg., J2:10 deg., J3:40 mm, and J4:90 deg.
The value that causes an error differs depending on the model. It can be set for each model within a range that does not cause an error.
Speed limiting using TCSpeed makes a gap between RealPos and CurPos before contacting the target object. Do not use TCSpeed, or if you want to use it, be sure that the value is the same as Speed.

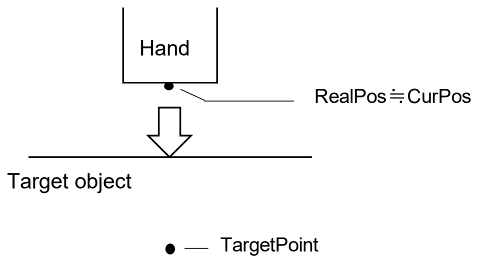


Figure 1. Not contacting target object

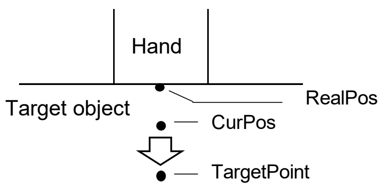


Figure 2. Contacting target object

1. When not contacting the target object
   The current position of the robot (RealPos) and the current target position of the robot (CurPos) are close.

2. When contacting the target object
   The current position of the robot (RealPos) maintains the position where the hand contacted the target object.
   The current target position of the robot (CurPos) moves toward TagetPoint.
   If this distance exceeds a certain amount, error No.9801 will occur.

The following shows a sample of code without occurrence of position error of Safety Board.
With using Till statement, errors can be avoided by proceeding to the next process when the difference between the "current position of the robot" and the "current target position of the robot" is greater than or equal to a certain amount.
It makes cycle time shorter by making the difference as small as possible, it can move on to the next process early.

TC Statement Example 2

```
Speed 5
Go ApproachPoint

'Set the Z axis torque limit to 20 %
TCLim -1, -1, 20, -1

Xqt posDiffChk(10.0) 'Starts a task of checking for position differences. Sets a flag if J3 has 10.0 mm
or more gap.

TC On
Go TargetPoint Till MemSw(0)
Wait 3
Go ApproachPoint
TC Off



Function posDiffChk(Zth As Double)
    Do
        If (Abs(CZ(RealPos) - CZ(CurPos)) > Zth) Then
       'Did the difference between "current position of the robot" and "current target position of
        the robot" exceed the threshold ?
            MemOn (0) ' Position deviation is large, Flag ON
        Else
            MemOff (0) ' Position deviation is large, Flag clear
        EndIf
        Wait 0.01
    Loop
Fend
```

TCLim Statement

Specifies the torque limit of each joint for the torque control mode.

Syntax

TCLim [*j1Torque limit*, *j2Torque limit*, *j3Torque limit*, *j4Torque limit* [, *j5Torque limit*]
[, *j6Torque limit*] [, *j7Torque limit*] [, *j8Torque limit*] [, *j9Torque limit*]]

Parameters

j1Torque limit	Specifies the proportion to the maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j2Torque limit	Specifies the proportion to the maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j3Torque limit	Specifies the proportion to the maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j4Torque limit	Specifies the proportion to the maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j5Torque limit	Option. Specifies the proportion to the maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j6Torque limit	Option. Specifies the proportion to the maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j7Torque limit	Option. Specifies the proportion to the maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j8Torque limit	Option. Specifies the proportion to the S axis maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.
j9Torque limit	Option. Specifies the proportion to the T axis maximum momentary torque (1 to 100 / unit: %) using an expression or numeric value. -1: Disable the torque limit and turns the mode to normal position control.

Return Values

When the parameters are omitted, returns the current torque limit.

Description

Setting to the torque limit becomes available at TC On.

When the limit value is too low, the robot doesn’t work and operation command stops before the robot reaches the target position.

In any of the following cases, TCLim set value is initialized.

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or QuitAll stops tasks

See Also

TC, TCLim Function, TCSpeed

TCLim Statement Example

```
Speed 5
Go ApproachPoint

'Set the Z axis torque limit to 20 %
TCLim -1, -1, 20, -1

TC On
Go TargetPoint
Wait 3
Go ApproachPoint
TC Off
```

Notes

When a position error detected on Safety Board mounted controller

Change the program so that “current position of the robot” and “current target position of the robot” are not far apart.
If they are far apart, the Safety Board detects failure, and “Erorr No.9801 Detected a position error by the Safety Board.” error occurs. (Only for controllers mounting the Safety Board.)
For more details, refer to TC Statement.

TCLim Function

Returns the torque limit of specified joint.

Syntax

TCLim (*jointNumber*)

Parameters

jointNumber	Specifies the joint number to retrieve the torque limit from using an expression or numeric value. The additional S axis is 8 and T axis is 9.

Return Values

Returns the integer number representing the current torque limit (1 to 100). -1 means the torque limit is invalid.

See Also

TC, TCLim, TCSpeed

TCLim Faction Example

```
Print "Current Z axis torque limit:", TCLim(3)
```

TCPSpeed Function

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 actual follow-up delay of the motor has been excluded from this value.
If the robot is executing a PTP (Point to Point) motion command, this function returns “0”.

Even if you are using the additional axis, only the robot travel distance is returned.
For example, it doesn’t include the travel speed of additional axis while you use the additional axis as running axis.

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
```

TCSpeed Statement

Specifies the speed limit in the torque control.

Syntax

TCSpeed [*speed*]

Parameters

speed	Specifies the proportion to the maximum speed (1 to 100 / unit: %) using an expression or numeric value.

Description

Under the torque control, the speed is limited to the TCSpeed setting despite of the speed settings of such as Speed command.
Error occurs if the speed goes over the limit in the torque control.

In any of the following cases, TCSpeed set value is initialized to 100%.

Controller Startup Motor On SFree, SLock, Brake Reset, Reset Error Stop button or QuitAll stops tasks

See Also

TC, TCLim, TCSpeed Function

TCSpeed Statement Example

```
Speed 5
Go ApproachPoint

'Set the Z axis torque limit to 20%
TCLim -1, -1, 20, -1
'Set the speed under the torque control to 5%, the same as the Speed

TcSpeed 5

TC On
Go TargetPoint
Wait 3
Go ApproachPoint
TC Off
```

Notes

When a position error detected on Safety Board mounted controller

Change the program so that “current position of the robot” and “current target position of the robot” are not far apart.
Speed limiting using TCSpeed makes a gap between “current position of the robot” and “current target position of the robot”. Do not use TCSpeed, or if you want to use it, be sure that the value is the same as Speed.
If they are far apart, the Safety Board detects failure, and “Erorr No.9801 Detected a position error by the Safety Board.” error occurs. (Only for controllers mounting the Safety Board.)
For more details, refer to TC Statement.

TCSpeed Function

Returns the speed limit in the torque control.

Syntax

TCSpeed

Return Values

Returns the integer number (1 to 100) representing the current speed limit.

See Also

TC, TCSpeed, TCLim

TCSpeed Function Example

```
Integer var
var = TCSpeed
```

TeachOn Function

Returns the Teach mode status.

Syntax
TeachOn

Return Values
True if it is in the Teach mode, False if not.

Description
TeachOn function is only used in the background task.

See Also
ErrorOn, EstopOn, SafetyOn, Xqt

TeachOn Function Example
The following example monitors the controller as it starts in Teach mode, and turns On/Off the I/O.

```
Function BGMain
   Do
         Wait 0.1
         If TeachOn = True Then
               On teachBit
         Else
               Off teachBit
         EndIf
         If SafetyOn = True Then
               On safetyBit
         Else
               Off safetyBit
         EndIf
         If PauseOn = True Then
               On PauseBit
         Else
               Off PauseBit
         EndIf
   Loop

Fend
```

TGo Statement

Executes Point to Point relative motion, in the current tool coordinate system.

Syntax

TGo *destination* [**CP**] [*searchExpr*] [*!...!*] [*SYNC*]

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.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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 (including N series), 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 parallel processing is used, other processing can be executed in parallel with the motion command.

The CP parameter causes acceleration of the next motion command to start when the deceleration starts for the current motion command. In this case the robot will not stop at the destination coordinate and will continue to move to the next point.

See Also

Accel, CP, Find, !....! Parallel Processing, P#= (Point Assignment), Speed, Till, TMove, Tool

TGo Statement Example

```
> TGo XY(100, 0, 0, 0) 'Move 100 mm 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 Here
  TGo XY(0, 0, -30, 0)
  Print Here

  Go P2
  Print Here
  TGo XY(0, 0, -30, 0)
  Print Here

Fend
```

```
[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /R /0
 X:  300.000 Y:  300.000 Z:  -50.000 U:    0.000 V:    0.000 W:    0.000 /R /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

Specifies and displays event 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 [*eventcondition*]

Parameters

eventcondition	Input status specified as a trigger [*Event*] comparative operator (=, <>, >=, >, <, <=) [*Integer expression*] The following functions and variables can be used in the *Event*: Functions : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr GetRobotInsideBox, GetRobotInsidePlane, Force, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus Variables : Byte, Int32, Integer, Long , Short, UByte, UInt32, UShort global preserve variable, Global variable, module variable In addition, using the following operators you can specify multiple event conditions. Operator : And, Or, Xor Example : Till Sw(5) = On Till Sw(5) = On And Till(6) = Off

Description

The Till statement can be used by itself or as a search expression in a motion command statement.

The Till condition must include at least one of the functions above.

When variables are included, their values are computed when setting the Till condition. No use of variable is recommended. Otherwise, the condition may be an unintended condition. 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.

To use a variables in the event condition expression

- Available variables are Integer type (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)
- Array variables are not available
- Local variables are not available
- If a variable value cannot satisfy the event condition for more than 0.01 seconds, the system cannot retrieve the change in variables.
- Up to 64 can wait for variables in one system (including the ones used in the event condition expressions such as Wait). If it is over 64, an error occurs during the project build.
- If you specify Byref to a waiting variable on any function call, an error occurs.
- When a variable is included in the right side member of the event condition expression, the value is calculated when starting the motion command. We recommend not using variables in an integer expression to avoid making unintended conditions.

See Also

Find, Go, In, InW, Jump, MemIn, MemSw, Move, Stat, Sw, TillOn, SF_GetStatus

Till Statement Example

Shown below are some sample lines from programs using the Till instruction.

```
Till Sw(1) = Off                     'Specifies Till condition (Input bit 1 off)
Go P1 Till                           'Stop if previous line condition is satisfied
Till Sw(1) = On And Sw($1) = On      'Specify new Till condition
Move P2 Till                         'Stop if previous line condition satisfied
Move P5 Till Sw(10) = On             'Stop if condition on this line is satisfied
```

TillOn Function

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 Statement

Displays the current time.

Syntax

Time

Description

Displays the current time in 24 hour format.

See Also

Date, Time$

Time Statement Example

Example from the command window:

```
> Time
10:15:32
```

Time Function

Returns the controller accumulated operating time.

Syntax

Time(*unitSelect*)

Parameters

unitSelect	An integer number ranging from 0 to 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 command 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

Returns the current system time.

Syntax

Time$

Return Values

A string containing the current time in 24 hour format *hh:mm:ss*.

See Also

Date, Date$, Time

Time$ Function Example

```
     Print "The current time is: ", Time$
```

TLClr Statement

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.)

Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

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

TLClr Statement Example

```
    TLClr 1
```

TLDef Function

Returns tool definition status.

Syntax

TLDef (*toolNumber*)

Parameters

toolNumber	Integer expression representing which tool to return status for.

Return Values

True if the specified tool has been defined, otherwise False.

See Also

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

TLDef Function Example

```
Function DisplayToolDef(toolNum As Integer)

   If TlDef(toolNum) = False Then
      Print "Tool ", toolNum, "is not defined"
   Else
      Print "Tool ", toolNum, ": ",
      Print TlSet(toolNum)
   EndIf
Fend
```

TLSet Statement

Defines or displays a tool coordinate system.

Syntax

(1) **TLSet** *toolNum*, *toolDefPoint*
(2) **TLSet** *toolNum*
(3) **TLSet**

Parameters

toolNum	Integer number from 1 to 15 representing which of 15 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

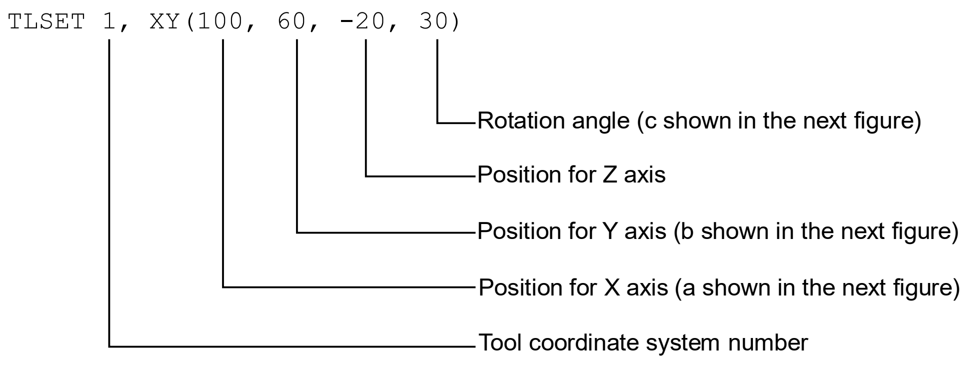
When parameters are omitted, displays all TLSet Definition.
When only the tool number is specified, displays specified TLSet Definition.

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).

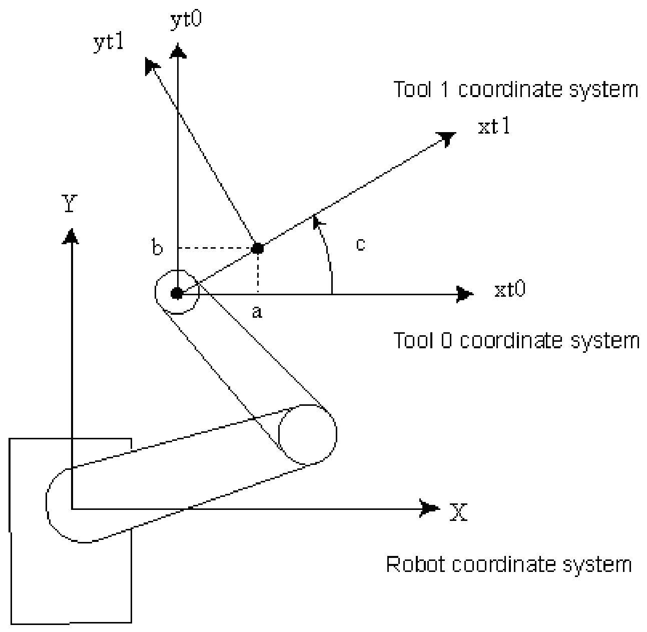
```
TLSet 1, XY(50,100,-20,30)

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.



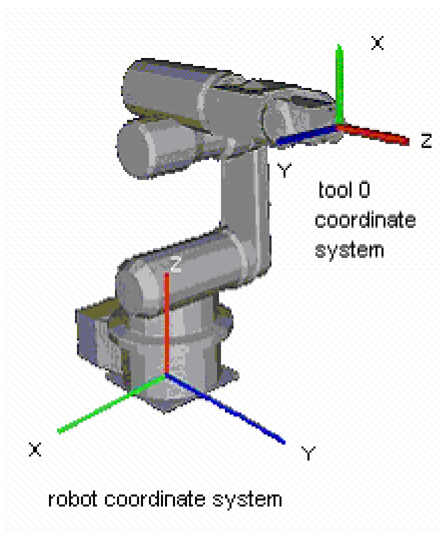
```
TLSET 1, XY(100, 60, -20, 30)
```

Rotation angle (c shown in the next figure)
Position for Z axis
Position for Y axis (b shown in the next figure)
Position for X axis (a shown in the next figure)
Tool coordinate system number

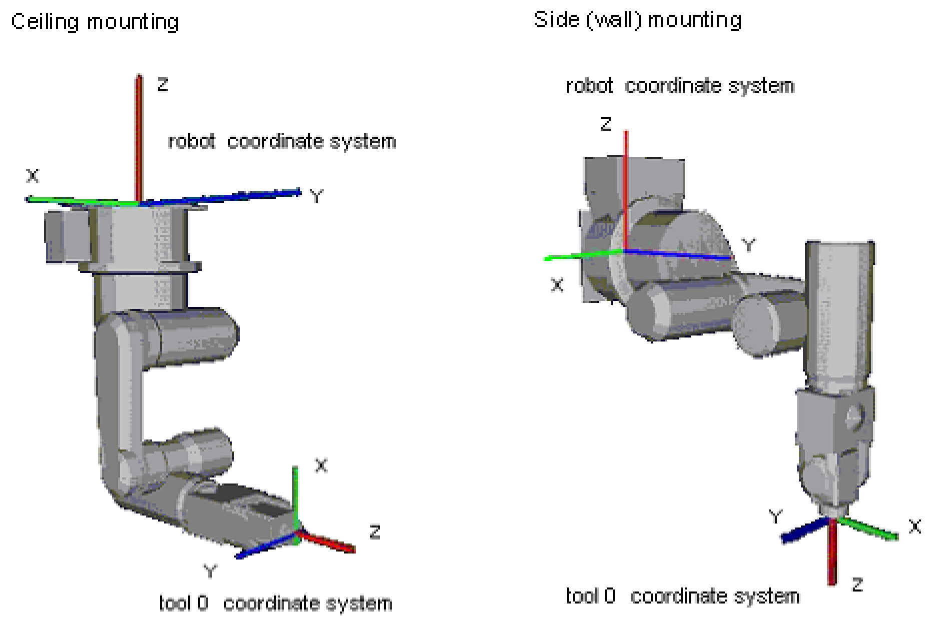


TLSet for 6-Axis robots

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:

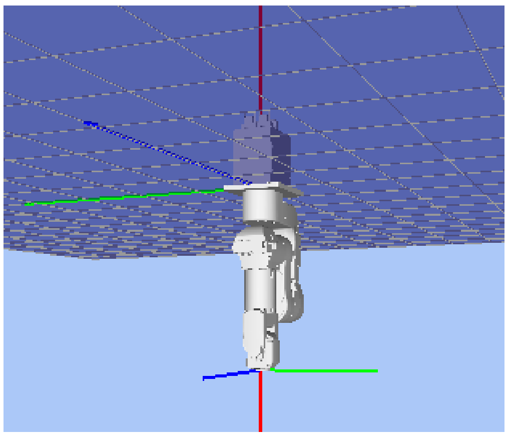


Tool 0 coordinate systems are defined for ceiling and wall mounted robots as shown in the figures below.



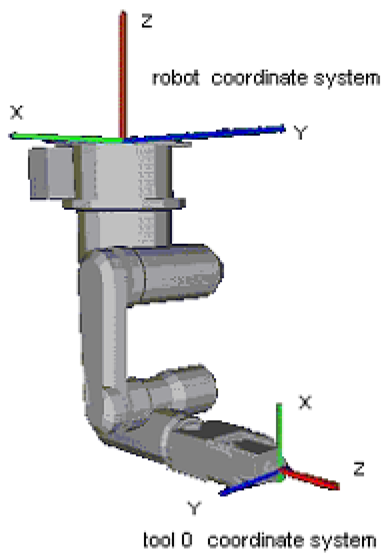
TLSet for N series robots

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 -X axis, the Y axis is aligned with the robot coordinate system's Y axis, and the Z axis is aligned with the robot coordinate system's -Z axis, as shown in the figure below:

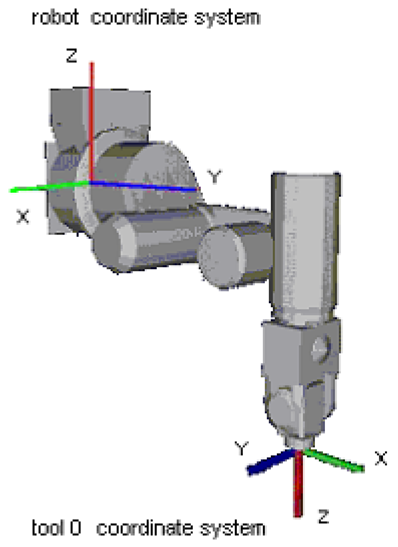


Tool 0 coordinate systems are defined for ceiling and wall mounted robots as shown in the figures below.

Ceiling mounting



Side (wall) mounting



Description

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

Note

TLSet values are maintained

The **TLSet** values are preserved. Use TLClr to clear a tool definition.

See Also

Tool, Arm, ArmSet, TLClr

TLSet Statement Example

The example shown below shows a good test which can be done from the command 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

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

Specifies the number of seconds to wait for the condition specified with the Wait instruction to come true before issuing a timeout error (error 2280).

Syntax

TMOut *seconds*

Parameters

seconds	Real expression representing the number of seconds until a timeout occurs. Valid range is 0 to 2147483 seconds in 1 second intervals.

Description

TMOut sets the amount of time to wait (when using the Wait instruction) until a timeout error 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 Statement Example

```
TMOut 5
Wait MemSw(0) = On
```

TMove Statement

Executes linear interpolation relative motion, in the current tool coordinate system

Syntax

TMove *destination* [**ROT**] [**CP**] [*searchExpr*] [*!...!*] [*SYNC*]

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.
SYNC	Reserves a motion command. The robot will not move until SyncRobots is executed.

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 (including N series), the arm orientation attributes are automatically changed in such a way that joint travel distance is as small as possible. This is equivalent to specifying the LJM modifier parameter for Move statement. Therefore, if you want to change the arm orientation larger than 180 degrees, execute it in several times.

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

Note

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, CP, Find, !....! Parallel Processing, Point Assignment, SpeedS, TGo, Till, Tool

TMove Statement Example

```
> TMove XY(100, 0, 0, 0)  'Move 100 mm 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 Here
  TMove XY(0, 0, -30, 0)
  Print Here

  Go P2
  Print Here
  TMove XY(0, 0, -30, 0)
  Print Here

Fend
```

```
[Output]
 X:  300.000 Y:  300.000 Z:  -20.000 U:    0.000 V:    0.000 W:    0.000 /R /0
 X:  300.000 Y:  300.000 Z:  -50.000 U:    0.000 V:    0.000 W:    0.000 /R /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

Tmr 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. (0 to 63)

Return Values

Elapsed time for the specified timer as a real number in seconds. Valid range is 0 to approx. 1.7E+31. Timer resolution is 0.001 seconds.

Description

Returns elapsed time in seconds since the timer specified was started. Unlike the ElapsedTime function, the Tmr function counts the time while the program is halted.

Timers are reset with TmReset.

```
Real overhead

TmReset 0
overHead = Tmr(0)
```

See Also

ElapsedTime Function, TmReset

Tmr Function Example

```
TmReset 0              'Resets Timer 0
For i = 1 To 10        'Performs operation 10 times
    GoSub Cycle
Next
Print Tmr(0) / 10     'Calculates and display cycle time
```

TmReset Statement

Resets the timers used by the Tmr function.

Syntax

TmReset *timerNumber*

Parameters

timerNumber	Integer expression representing which of the 64 timers to reset. (0 to 63)

Description

Resets and starts the timer specified by *timerNumber*.

Use the Tmr function to retrieve the elapsed time for a specific timer.

See Also

Tmr

TmReset Statement Example

```
TmReset 0           'Resets Timer 0
For i = 1 To 10     'Performs operation 10 times
    GoSub CYL
Next
Print Tmr(0)/10     'Calculates and display cycle time
```

Toff Statement

Turns off execution line display on the LCD.

Syntax

Toff

Description

Execution line will not be displayed on the LCD.

Note

About the Controllers to use

It cannot be used with RC90/T/VT series.

See Also

Ton

Toff Statement Example

```
Function main
  Ton MyTask
  ...
  Toff
Fend
```

Ton Statement

Specifies a task which shows an execution line on the LCD.

Syntax

Ton *taskIdentifier*
Ton

Parameters

taskIdentifier	Task name or integer expression representing the task number. Task name is a function name used in an Xqt statement or a function started from the Run window or Operator window. Task number range is: Normal tasks : 1 to 32

Description

Execution line of task 1 is displayed in initial status.
Ton statement displays the specified task execution line on the LCD.
When taskIdentifier is omitted, the task execution line with Ton statement execution is displayed on the LCD.

Note

About the Controllers to use
It cannot be used RC90/T/VT series.

See Also

Toff

Ton Statement Example

```
Function main
  Ton MyTask
  ...
  Toff
Fend
```

Tool Statement

Selects or displays the current tool.

Syntax

(1) **Tool** *toolNumber*
(2) **Tool**

Parameters

toolNumber	Optional. Integer expression from 0 to 15 representing which of 16 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.

Lifetime of Compact Flash

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

See Also

TGo, TLSet, Tmove

Tool Statement Example

The example shown below shows a good test which can be done from the command 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

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 (User defined trigger)

Defines interrupts and what should happen when they occur.
With the Trap statement, you can jump to labels or call functions when the event occurs.
Trap statement has 2 types as below:
- 4 Traps that interrupts by the user defined input status
- 7 Traps that interrupts by the system status

Trap with user defined trigger is explained here.

Syntax

Trap *trapNumber*, *eventCondition* **GoTo** *label*
Trap *trapNumber*, *eventCondition* **Call** *funcname*
Trap *trapNumber*, *eventCondition* **Xqt** *funcname*
Trap *trapNumber*

Parameters

trapNumber Integer number from 1 to 4 representing which of 4 Trap numbers to use. (SPEL$^+$ supports up to 4 active Trap interrupts at the same time.)

eventCondition Input status specified as a trigger
[*Event*] comparative operator (=, <>, >=, >, <, <=) [*Integer expression*]

The following functions and variables can be used in the *Event*:

Functions : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus

Variables : Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort global preserve variable, Global variable, module variable In addition, using the following operators you can specify multiple event conditions.
Operator : And, Or, Xor
Example : Trap 1, Sw(5) = On Call, TrapFunc
Trap 1, Sw(5) = On And Till(6) = Off, Call TrapFunc

label The label where program execution is to be transferred when Trap condition is satisfied.

funcName The function that is executed when Call or Xqt when the Trap condition is satisfied. The function with argument cannot be specified.

Description

A Trap executes interrupt processing which is specified by GoTo, Call, or Xqt when the specified condition is satisfied.

The Trap condition must include at least one of the functions above.
When variables are included in the Trap condition, their values are computed when setting the Trap condition. No use of variable is recommended. Otherwise, the condition may be an unintended condition.

Once the interrupt process is executed, its Trap setting is cleared. If the same interrupt process is necessary, the Trap instruction must execute it again.

To cancel a Trap setting simply execute the Trap instruction with only the *trapNumber* parameter. e.g. “Trap 3” cancels Trap #3.
When the Function that executed Trap GoTo ends (or exit), the Trap Goto will be canceled automatically.
When the declared task ends, Trap Call will be canceled.
Trap Xqt will be canceled when all tasks have stopped.

If GoTo is specified

In the task set to Trap, the command being executed will be processed as described below. Then, control branches to the specified 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 Call is specified

After executing the same process as GoTo described above, then control branches to the specified line number or label.
Once the function ends, program execution returns to the next statement after the statement where program interruption occurred. Call statements cannot be used in the Trap processing function.
When an error occurs in the trap process function, error handling with OnErr will be invalid and an error will occur.

If Xqt is specified

Program control executes the specified function as an interrupt processing task. In this case, the task which executes the Trap command will not wait for the Trap function to finish and will continue to execute.
You cannot execute a task with an Xqt statement from an interrupt processing task.

Notes

For EPSON RC+4.x user

The Trap Call function of EPSON RC+ 4.x or before is replaced with Trap Xqt in EPSON RC+ 7.0.
The Trap GoSub function of EPSON RC+ 4.x or before is removed in EPSON RC+ 7.0. Instead, use Trap Call.

To use a variables in the event condition expression

- Available variables are Integer type (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)
- Array variables are not available
- Local variables are not available
- If a variable value cannot satisfy the event condition for more than 0.01 seconds, the system cannot retrieve the change in variables.
- Up to 64 can wait for variables in one system (including the ones used in the event condition expressions such as Wait). If it is over 64, an error occurs during the project build.
- If you specify Byref to a waiting variable on any function call, an error occurs.
- When a variable is included in the right side member of the event condition expression, the value is calculated when setting the Trap condition. We recommend not using variables in an integer expression to avoid making unintended conditions.

See Also

Call, GoTo, Xqt, SF_GetStatus

Trap Statement 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> Using global variable as event condition

```
Global Integer gi

Function main
   Trap 1, gi = 5 GoTo THandle
   Xqt sub
   Wait 100
   Exit Function

THandle:
   Print "IN Trap ", gi

Fend

Function sub
   For gi = 0 To 10
      Print gi
      Wait 0.5
   Next
Fend
```

Trap Statement (System status trigger)

Defines interrupts and what should happen when they occur.
With the Trap statement, you can jump to labels or call functions when the event occurs.
Trap statement has 2 types as below:
- 4 Traps that interrupts by the user defined input status
- 7 Traps that interrupts by the system status

Trap with system status triggers is explained here.

Syntax

Trap {*Emergency* | *Error* | *Pause* | *SGOpen* | *SGClose* | *Abort* | *Finish* } *Xqt funcname*
Trap {*Emergency* | *Error* | *Pause* | *SGOpen* | *SGClose* | *Abort* | *Finish* }

Parameters

Emergency	In the emergency stop status, executes the specified function.
Error	In the error status, executes the specified function.
Pause	In the pause status, executes the specified function.
SGOpen	When safeguard is open, executes the specified function.
SGClose	When safeguard is closed, executes the specified function.
Abort	All tasks except the background tasks stops (such as when a statement corresponding to the Abort All is executed or Pause button is pressed) by the user or system, executes the specified function.
Finish	All tasks except the background tasks are completed, executes the specified function. It cannot be executed in the condition which executes the Trap Abort.
funcname	Function of interrupt processing task for which Xqt is executed when the system status is completed. Functions with argument cannot be specified. However, three parameters can be specified if "Error" is specified for the parameter.

Note

Trap *** Call function of EPSON RC+4.x or before is replaced to Trap *** Xqt in EPSON RC+ 7.0.

Description

When the system status completes, the specified interrupt processing task is executed.

Even if you execute an interrupt processing task, the Trap settings cannot be cleared.
To clear the Trap setting, omit the *funcname* and execute the Trap statement.
Example : Trap Emergency clears Trap Emergency
After all normal tasks are completed and the controller is in the Ready status, all Trap settings are cleared.

You cannot execute more tasks using the Xqt from an interrupt processing.

⚠ CAUTION	■ Forced flag You can turn On/Off the I/O outputs even in the Emergency Stop status, Safeguard Open status, Teach mode, or error status by specifying the Forced flag to the I/O output statement such as On and Off statements. DO NOT connect the external devices which can move machines such as actuators with the I/O outputs which specifies the Forced flag. It is extremely dangerous and it can lead the external devices to move in the Emergency Stop status, Safeguard Open status, Teach mode, or error status. I/O outputs which specifies the Forced flag is supposed to be connected with the external device such as LED as the status display which cannot move machines.

If Emergency is specified
When the Emergency Stop is activated, the specified function is executed in the NoEmgAbort task attribute.
The commands executable from the interrupt processing tasks can execute the NoEmgAbort task.
When the interrupt processing of Emergency Stop is completed, finish the task promptly. Otherwise, the controller cannot be in the Ready status. You cannot reset the Emergency Stop automatically by executing the Reset command from the interrupt processing task.
When the task executes I/O On/Off from the interrupt processing task, uncheck the [Outputs off during emergency stop] check box in the [Controller]-[Preferences] page. If this check box is checked, the execution order of turn Off by the controller and turn On using the task are not guaranteed.

If Error is specified
When the Error is activated, the specified function is executed in the NoEmgAbort task attribute.
The commands executable from the interrupt processing tasks can execute the NoEmgAbort task.
When the interrupt processing of Emergency Stop is completed, finish the task promptly. Otherwise, the controller cannot be in the Ready status.
The three omittable parameters (*errNumber, robotNumber, jointNumber*) can be specified to the user function. If you want to use these parameters, add three byval integer parameters to the trap function.
If a motion error occurs, *errNumber, robotNumber,* and *jointNumber* are set.
If an error other than the motion error occurs, '0' will be set to *robotNumber,* and *jointNumber.*

If Pause is specified
When the Pause is activated, the specified function is executed in the NoEmgAbort task attribute.

If SGOpen is specified
When the Safeguard is open, the specified function is executed in the NoEmgAbort task attribute.

If SGClose is specified
When the safeguard is closed and latched, the specified function is executed in the NoEmgAbort task attribute.
If you execute the Cont statement from the interrupt processing tasks, an error occurs.

If Abort is specified
All tasks except background tasks stop (such as when a statement corresponding to the Abort All is executed or Pause button is pressed) by the user or system, executes the specified function in the NoPause attribute.
When the interrupt processing of Pause is completed, finish the task promptly. Otherwise, the controller cannot be in the Ready status. Although a task executed with the Trap Abort has an error, the Trap Error processing task is not executed.
If the Shutdown or Restart statements are aborted, processing tasks of neither the Trap Abort nor Trap Finish is executed.

If Finish is specified
All tasks except the background tasks stops (such as when a statement corresponding to the Abort All is executed or Pause button is pressed) by the user or system, executes the specified function in the NoPause attribution. It cannot be executed in the condition which executes the Trap Abort processing task.
When the shutdown and interrupt processing are completed, finish the tasks promptly. Otherwise, the controller cannot be in the Ready status.

See Also
Era, Erl, Err, Ert, ErrMsg$, OnErr, Reset, Restart, SysErr, Xqt

Trap Statement Example

```
Function main
     :
  Trap Error Xqt suberr
     :
Fend

Function suberr
  Print "Error =", Err
  On ErrorSwitch
Fend

Function main

  Trap Error Xqt trapError

FEnd


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
```

Trim$ Function

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

Returns the status of the Wait, WaitNet, and WaitSig commands.

Syntax

TW

Return Values

Returns False if Wait condition is satisfied within the time interval.
Returns True 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 False (Wait condition was satisfied) or a True (time interval has elapsed).

See Also

TMOut, Waitt, Hand_TW

TW Function Example

```
Wait Sw(0) = On, 5      'Waits up to 5 seconds for input bit 0 On
If TW = True Then
    Print “Time Up”     'Displays “Time UP” after 5 seconds
EndIf
```

UBound Function

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

Redim

UBound Function Example

```
Integer i, a(10)

For i=0 to UBound(a)
    a(i) = i
Next
```

UByte Statement

Declares variables of UByte type. (unsigned variable type, size: 2 bytes).

Syntax

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

Parameters

varName	Variable name which the user wants to declare as UByte type.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

UByte is used to declare variables as UByte type. Variables of UByte type can contain values from 0 to 255. 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, Int32, Int64, Integer, Long, Real, Short, String, UInt32, UInt64, UShort

UByte Statement Example

The following example shows a simple program that declares some variables as UByte type and assigns values to the variables.
The program monitors whether the top bit of “test_ok” is 1 or 0. The result will be displayed on the screen. (Since the value 15 is assigned to the variable, the bit with higher “test_ok” value is always set in this example.)

```
Function Test
    UByte A(10)          'Single dimension array of UByte type
    UByte B(10, 10)      'Two dimension array of UByte type
    UByte C(5, 5, 5)     'Three dimension array of UByte type
    UByte 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"
    EndIf
Fend
```

UCase$ Function

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$ Function Example

```
str$ = "Data"
str$ = UCase$(str$)   ' str$ = "DATA"
```

UInt32 Statement

Declares variables of UInt32 type. (unsigned 4-byte integer variable).

Syntax

UInt32 *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameters

varName	Variable name which the user wants to declare.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

UInt32 is used to declare variables as integer type. Variables of integer type can contain values from 0 to 4294967295. 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, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt64, UShort

UInt32 Statement Example

The following example shows a simple program that declares some variables as integer type using UInt32.

```
Function uint32test
    UInt32 A(10)          'Single dimension array of UInt32 type
    UInt32 B(10, 10)      'Two dimension array of UInt32 type
    UInt32 C(5, 5, 5)     'Three dimension array of UInt32 type
    UInt32 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
```

UInt64 Statement

Declares variables of Uint64 type. (unsigned 8-byte integer variable).

Syntax

Uint64 *varName* [**(***subscripts***)**] [, *varName* [**(***subscripts***)**]...]

Parameters

varName	Variable name which the user wants to declare.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows.

(ubound1, [ubound2], [ubound3])

ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension.

The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1.

When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

UInt64 is used to declare variables as integer type. Variables of integer type can contain values from 0 to 18446744073709551615. 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, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UShort

UInt64 Statement Example

The following example shows a simple program that declares some variables as integer type using UInt64.

```
Function uint64test
    UInt64 A(10)          'Single dimension array of UInt64 type
    UInt64 B(10, 10)      'Two dimension array of UInt64 type
    UInt64 C(5, 5, 5)     'Three dimension array of UInt64 type
    UInt64 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
```

UOpen Statement

Opens a file for read / write access.

Syntax

UOpen *fileName* **As** *#fileNumber*

.

.

Close *#fileNumber*

Parameters

fileName	String expression that specifies path and file name. If path is omitted, the file in the current directory is specified See ChDisk for the details.
fileNumber	Integer expression representing values from 30 to 63.

Description

Opens the specified file by the specified file number. This statement is used for writing and loading data in the specified file.

Note

A network path is available.

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.

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.

Close closes the file and releases the file number.

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 #, Input#, AOpen, BOpen, ROpen, WOpen, FreeFile, Seek

UOpen Statement Example

```
Integer fileNum, i, j

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

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
Seek #fileNum, 10
Input #fileNum, j
Print "data = ", j
Close #fileNum
```

UpdateDB Statement

Updates the data in the table which is retrieved in the opened data base.

Syntax

UpdateDB *#DBNumber, item, value*

Parameters

DBNumber	Integer expression (501 to 508) representing the data base number specified by OpenDB.
item	Item name of the table to update.
value	Value to be updated.

Description

Updates the data in the table which is retrieved in the opened data base with the specified value.
Before updating the data, it is required to issue SelectDB and select the record to be updated.

Note

- Connection of PC with installed RC+ is required.

See Also

OpenDB,CloseDB, SelectDB, DeleteDB

UpdateDB Statement Example

Example using SQL database
Following is an example to register the data to the table “Employees” in the sample database “Northwind” of SQL server 2000, and update the items in the registered data.

```
Integer count, i, eid
String Lastname$, Firstname$, Title$

OpenDB #501, SQL, "(LOCAL)", "Northwind"
count = SelectDB(#501, "Employees", "TitleOfCourtesy = 'Mr.'")
Print #501, "Epson", "Taro", "Engineer", "Mr."
count = SelectDB(#501, "Employees", "LastName = 'Epson' and
FirstName = 'Taro'")
Input #501, eid, Lastname$, Firstname$, Title$
Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
UpdateDB #501, "Title", "Chief Engineer"
count = SelectDB(#501, "Employees", "LastName = 'Epson' and
FirstName = 'Taro'")
Input #501, eid, Lastname$, Firstname$, Title$
Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
CloseDB #501
```

UShort Statement

Declares variables of UShort type. (unsigned 2-byte integer variable).

Syntax

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

Parameters

varName	Variable name which the user wants to declare.
subscripts	Optional. Dimensions of an array variable; up to 3 dimensions may be declared. The subscripts syntax is as follows (ubound1, [ubound2], [ubound3]) ubound1, ubound2, ubound3 each specify the maximum upper bound for the associated dimension. The elements in each dimension of an array are numbered from 0 and the available number of array elements is the upper bound value + 1. When specifying the upper bound value, make sure the number of total elements is within the range shown below:

Local variable	2,000
Global Preserve variable	4,000
Global variable and module variable	100,000

Description

UShort is used to declare variables as integer type. Integer variables can contain values from 0 to 65535. 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, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64

UShort Statement Example

The following example shows a simple program that declares some variables as integer type using UShort.

```
Function ushorttest
    UShort A(10)          'Single dimension array of UShort type
    UShort B(10, 10)      'Two dimension array of UShort type
    UShort C(5, 5, 5)     'Three dimension array of UShort type
    UShort 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
```

Val Function

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 Function 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 Command window.

```
> Print Val("25.999")
25.999
>
```

VSD Statement

Sets the variable speed CP motion function for SCARA robot.

Syntax

VSD { ON | Off }

Parameters

On | Off	On: Enables the variable speed CP motion function of SCARA robot. Off: Disables the variable speed CP motion function of SCARA robot.

Description

VSD is available for following commands.
    Move, Arc, Arc3
This command is available only for SCARA robots.
For other than SCARA robots, use AvoidSingularity SING_VSD.

The variable speed CP motion function prevents the acceleration error and overspeed error from occurring when SCARA robot is executing CP motion. This function automatically controls the joint speed while keeping the trajectory. If the joint speed is controlled, the tool center point speed specified by SpeedS will not be kept. However, the original speed setting will be returned when the joint speed gets below the limit. If constant velocity is prioritized, set AccelS, DecelS, and SpeedS smaller and eliminate the error occurrence.

If the acceleration and overspeed errors occur even when the VSD statement is used, set AccelS, DecelS, and SpeedS smaller.

If the VSD parameter is changed, the current setting is effective until the next controller startup.
VSD is set off when the startup of the controller.

See Also

VSD Function

VSD Statement Example

```
VSD On 'Enable the variable speed CP motion and execute the motion
  Move P1
  Move P2
VSD Off
```

VSD Function

Returns the setting of the variable speed CP motion function for SCARA robot.

Syntax

VSD

Return Values

On	= Enables the variable speed CP motion function
Off	= Disables the variable speed CP motion function

See Also

VSD

VSD Function Example

```
If VSD = Off Then
    Print "Variable Speed Drive is off"
EndIf
```

VxCalib Statement

This command is only for use with external vision systems and cannot be used with Vision Guide.
Creates calibration data for an external vision system.

Syntax

(1) **VxCalib** *CalNo*
(2) **VxCalib** *CalNo*, *CamOrient*, *P(pixel_st : pixel_ed)*, *P(robot_st : robot_ed)* [, *TwoRefPoints*]
(3) **VxCalib** *CalNo*, *CamOrient*, *P(pixel_st : pixel_ed)*, *P(robot_st : robot_ed)*, *P(ref0)* [, *P(ref180)*]

Parameters

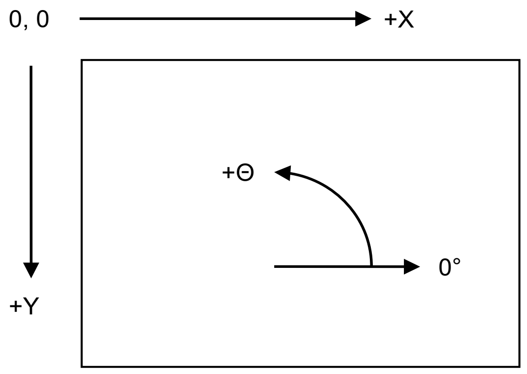
CalNo	Integer expression that specifies the calibration data number. The range is from 0 to 15; up to 16 calibrations may be defined.
CamOrient	Integer expression that specifies the camera mounting direction using the following values: 1 to 3: Available only for syntax (2). 4 to 7: Available only for syntax (3). 1: Standalone 2: Fixed downward 3: Fixed upward 4: Mobile on Joint #2 5: Mobile on Joint #4 6: Mobile on Joint #5 7: Mobile on Joint #6
P(pixel_st : pixel_ed)	Specifies the Pixel coordinates (X, Y only) using the continuous point data.
P(robot_st : robot_ed)	Specifies the robot coordinates using the continuous point data. The point data varies with mounting directions of the camera specified by CamOrient. If CamOrient = 1 to 3: The robot coordinates must be set to the current TOOL and ARM values. If CamOrient = 4 to 7: The robot coordinates must be set as TOOL: 0, ARM: 0.
TwoRefPoints	Available for syntax (1). True, when using two measuring points. False, when using one measuring point. Specifying two measuring points makes the calibration more accurate. Optional. Default: False
P(ref0)	Available for syntax (3). Specifies the robot coordinates of the reference point using the point data.
P(ref180)	Available for syntax (3). Specifies the robot coordinates of the second reference point using the point data. Specifying two reference points makes the calibration more accurate. Optional.

Description

The VxCalib command calculates the vision calibration data for the specified calibration number using the specified camera orientation, pixel coordinates, robot coordinates, and reference points (Mobile camera only) given by the parameter.

When you specify only *CalNo*, the point data and other settings you defined are displayed (only from the Command Window).

The following figure shows the coordinates system of the pixel coordinates. (Units: pixel)



For the pixel coordinates and robot coordinates, set the top left position of the window as Point 1 and set the bottom right position as Point 9 according to the order in the table below.
It is classified into the four categories by the parameter CamOrient and TwoRefPoints.

1) CamOrient = 1 to 3 (Standalone, Fixed Downward, Fixed Upward), TwoRefPoints = False

Data order	Position	Pixel coordinates	Robot coordinates
1	Top left	Detection coordinates 1	Measuring point coordinates 1
2	Top center	Detection coordinates 2	Measuring point coordinates 2
3	Top right	Detection coordinates 3	Measuring point coordinates 3
4	Center right	Detection coordinates 4	Measuring point coordinates 4
5	Center	Detection coordinates 5	Measuring point coordinates 5
6	Center left	Detection coordinates 6	Measuring point coordinates 6
7	Bottom left	Detection coordinates 7	Measuring point coordinates 7
8	Bottom center	Detection coordinates 8	Measuring point coordinates 8
9	Bottom right	Detection coordinates 9	Measuring point coordinates 9

2) CamOrient = 2 (Fixed Downward), TwoRefPoints = True
Note: When the tool is exactly defined, TwoRefPoints is not necessary and should be set to False.

By setting TwoRefPoints to True, two measuring points are used for each calibration position, which makes the calibration more accurate. 18 robot points with U axis: 0 degree / 180 degrees are required.
After setting 1 to 9 measuring points coordinates, turn the U axis by 180 degrees and set the measuring point coordinates 10 to 18 where the hand (such as the rod) is positioned at the calibration target position.

Data order	Position	Pixel coordinates	Robot coordinates	U axis
1	Top left	Detection coordinates 1	Measuring point coordinates 1	0 degree
2	Top center	Detection coordinates 2	Measuring point coordinates 2	
3	Top right	Detection coordinates 3	Measuring point coordinates 3	
4	Center right	Detection coordinates 4	Measuring point coordinates 4	
5	Center	Detection coordinates 5	Measuring point coordinates 5	
6	Center left	Detection coordinates 6	Measuring point coordinates 6	
7	Bottom left	Detection coordinates 7	Measuring point coordinates 7	
8	Bottom center	Detection coordinates 8	Measuring point coordinates 8	
9	Bottom right	Detection coordinates 9	Measuring point coordinates 9	
10	Top left	- - -	Measuring point coordinates 10	180 degrees
11	Top center	- - -	Measuring point coordinates 11	
12	Top right	- - -	Measuring point coordinates 12	
13	Center right	- - -	Measuring point coordinates 13	
14	Center	- - -	Measuring point coordinates 14	
15	Center left	- - -	Measuring point coordinates 15	
16	Bottom left	- - -	Measuring point coordinates 16	
17	Bottom center	- - -	Measuring point coordinates 17	
18	Bottom right	- - -	Measuring point coordinates 18	

3) CamOrient = 3 (Fixed Upward), TwoRefPoints = True
Note: When the tool is exactly defined, TwoRefPoints is not necessary and should be set to False.

By setting TwoRefPoints to True, two detection points are used, which makes the calibration more accurate. For only the pixel coordinates, 18 points of U axis: 0 degree / 180 degrees are required.
After setting 1 to 9 detection coordinates at the each measuring point coordinates at 0 degrees, set the detection coordinates for points 10 to 18 at 180 degrees.

Data order	Position	Pixel coordinates	Robot coordinates	U axis
1	Top left	Detection coordinates 1	Measuring point coordinates 1	0 degree
2	Top center	Detection coordinates 2	Measuring point coordinates 2	
3	Top right	Detection coordinates 3	Measuring point coordinates 3	
4	Center right	Detection coordinates 4	Measuring point coordinates 4	
5	Center	Detection coordinates 5	Measuring point coordinates 5	
6	Center left	Detection coordinates 6	Measuring point coordinates 6	
7	Bottom left	Detection coordinates 7	Measuring point coordinates 7	
8	Bottom center	Detection coordinates 8	Measuring point coordinates 8	
9	Bottom right	Detection coordinates 9	Measuring point coordinates 9	
10	Top left	Detection coordinates 10	- - -	180 degrees
11	Top center	Detection coordinates 11	- - -	
12	Top right	Detection coordinates 12	- - -	
13	Center right	Detection coordinates 13	- - -	
14	Center	Detection coordinates 14	- - -	
15	Center left	Detection coordinates 15	- - -	
16	Bottom left	Detection coordinates 16	- - -	
17	Bottom center	Detection coordinates 17	- - -	
18	Bottom right	Detection coordinates 18	- - -	

4) CamOrient = 4 to 7

Data order	Position	Pixel coordinates	Robot coordinates
1	Top left	Detection coordinates 1	Measuring point coordinates 1
2	Top center	Detection coordinates 2	Measuring point coordinates 2
3	Top right	Detection coordinates 3	Measuring point coordinates 3
4	Center right	Detection coordinates 4	Measuring point coordinates 4
5	Center	Detection coordinates 5	Measuring point coordinates 5
6	Center left	Detection coordinates 6	Measuring point coordinates 6
7	Bottom left	Detection coordinates 7	Measuring point coordinates 7
8	Bottom center	Detection coordinates 8	Measuring point coordinates 8
9	Bottom right	Detection coordinates 9	Measuring point coordinates 9

Note

In addition to the tables above, specify the robot coordinates of the reference points.
Using the two reference points makes the calibration more accurate. In this case, it needs two points of U axis: 0 degree / 180 degrees.

After setting the first reference points coordinates, turn the U axis by 180 degrees and set the second reference points coordinates where the hand (such as the rod) is positioned at the calibration target position. When the tool is exactly defined, the two reference points are not necessary.

See Also

VxTrans Function, VxCalInfo Function, VxCalDelete, VxCalSave, VxCalLoad

VxCalib Statement Example

```
Function MobileJ2

   Integer i
   Double d(8)

   Robot 1
   LoadPoints "MobileJ2.pts"

   VxCalib 0, 4, P(21:29), P(1:9), P(0)

   If (VxCalInfo(0, 1) = True) Then
         For i = 0 To 7
               d(i) = VxCalInfo(0, i + 2)
         Next i
         Print "Calibration result:"
         Print d(0), d(1), d(2), d(3), d(4), d(5), d(6), d(7)

         P52 = VxTrans(0, P51, P50)
         Print "Coordinates conversion result:"
         Print P52
         SavePoints "MobileJ2.pts"
         VxCalSave "MobileJ2.caa"
   Else
         Print "Calibration failed"
   EndIf

Fend
```

VxCalDelete Statement

This command is only for use with external vision systems and cannot be used with Vision Guide.
Deletes the calibration data for an external vision system calibration.

Syntax

VxCalDelete *CalNo*

Parameters

CalNo	Integer expression that specifies the calibration data number. The range is from 0 to 15; up to 16 calibrations may be defined.

Description

Deletes the calibration data defined by the specified calibration number.

See Also

VxCalib, VxTrans Function, VxCalInfo Function, VxCalSave, VxCalLoad

VxCalDelete Statement Example

```
VxCalDelete "MobileJ2.caa"
```

VxCalLoad Statement

This command is only for use with external vision systems and cannot be used with Vision Guide.
Loads the calibration data for an external vision system calibration from a file.

Syntax

VxCalLoad *FileName*

Parameters

FileName	Specifies the file name from which the calibration data is loaded using a string expression. The file extension is ".caa". If omitted, ".caa" is automatically added. For extensions other than ".caa", they are automatically changed to ".caa".

Description

Loads the calibration data from the specified file in the current project.

See Also

VxCalib, VxTrans Function, VxCalInfo Function, VxCalDelete, VxCalSave

VxCalLoad Statement Example

```
VxCalLoad "MobileJ2.caa"
```

VxCalInfo Function

This command is only for use with external vision systems and cannot be used with Vision Guide.
Returns the calibration completion status and the calibration data.

Syntax

VxCalInfo (*CalNo*,*CalData*)

Parameters

CalNo	Integer expression that specifies the calibration data number. The range is from 0 to 15; up to 16 calibrations may be defined.
CalData	Specifies the calibration data type to acquire using the integer values in the table below.

CalData	Calibration Data Type
1	CalComplete
2	X Avg Error [mm]
3	X Max error [mm]
4	X mm per pixel [mm]
5	X tilt
6	Y Avg error [mm]
7	Y Max error [mm]
8	Y mm per pixel [mm]
9	Y tilt

Return Values

Returns the specified calibration data. For CalData = 1, the data type is Boolean. For all other data, the data type is Double.

Description

You can check which calibration has defined calibration data.
Also, you can retrieve the calibration data values.

See Also

VxCalib, VxTrans Function, VxCalDelete, VxCalSave, VxCalLoad

VxCalInfo Function Example

```
Print VxCalInfo(0, 1)
```

VxCalSave Statement

This command is only for use with external vision systems and cannot be used with Vision Guide.
Saves the calibration data for an external vision system calibration to a file.

Syntax

VxCalSave *FileName*

Parameters

FileName	Specifies the file name from which the calibration data is loaded using a string expression. The extension is ".caa". If omitted, ".caa" is automatically added. For extensions other than ".caa", they are automatically changed to ".caa".

Description

Saves the calibration data with the specified file name. The file is saved in the current project. If the file name is already existed, the calibration data is overwritten.

See Also

VxCalib, VxTrans Function, VxCalInfo Function, VxCalDelete, VxCalLoad

VxCalSave Statement Example

```
VxCalSave "MobileJ2.caa"
```

VxTrans Function

This command is only for use with external vision systems and cannot be used with Vision Guide.
Converts pixel coordinates to robot coordinates and returns the converted point data.

Syntax

VxTrans (*CalNo*, *P(pixel)* [, *P(camRobot)*]) As Point

Parameters

CalNo	Integer expression that specifies the calibration data number. The range is from 0 to 15; up to 16 calibrations may be defined.
P(pixel)	Specifies the vision pixel coordinates (X, Y, U only) using point data.
P(camRobot)	Optional. For a mobile camera, this is the position where the robot was located when the image was acquired. If not specified, then the current robot position is used. The point should be in TOOL: 0 and ARM: 0.

Return Values

Returns the calculated robot coordinates using the point data.

Description

This command converts pixel coordinates to robot coordinates using the calibration data of the specified calibration number.

When using a mobile camera, specify *P(camRobot)* if the robot has been moved from the position where the image was acquired. Ensure that *P(camRobot)* is in TOOL: 0 and ARM: 0. The Joint #4 and Joint #6 angles of the set robot coordinates are used for the calculation.

See Also

VxCalib, VxCalInfo Function, VxCalDelete, VxCalSave, VxCalLoad

VxTrans Function Example

```
P52 = VxTrans(0, P51, P50)
```

Wait Statement

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.) Also waits for the values of global variables to change.

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 0.01 seconds.
inputcondition	The following syntax can be used to specify the input condition:

[*Event*] Comparative operator (=, <>, >=, >, <, <=) [Integer expression]

The following functions and variables can be used in the *Event*.

Functions :	Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, MCalComplete, Motor, LOF, ErrorOn, SaftyOn, EstopOn, TeachOn , Cnv_QueLen, WindowsStatus, AtHome, LatchState, WorkQue_Len, PauseOn, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus
Operators :	Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort global preserve variables, global variables, module variables

In addition, using the following operators you can specify multiple input conditions.

Operator : And, Or, Xor, Mask

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 Event 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 Event 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.)

Waiting for variable with Wait

- Available variables are Integer type (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)
- Array variables are not available
- Local variables are not available
- If variables value cannot satisfy the event condition for more than 0.01 seconds, the change in variables may not be retrieved.
- Up to 64 can wait for variables in one system (including ones used in the event condition expressions such as Till). If it is over 64, an error occurs during the project build.
- If you specify Byref to a waiting variable on any function call, an error occurs.
- When a variable or a functions are included in the right side member of the event condition expression, the value is calculated when setting the Trap condition. We recommend not using variables or functions in an integer expression to avoid making unintended conditions.

When Using PC COM port (1001 to 1008)

- You cannot use Lof Function for Wait instruction.

When the program is paused while Wait is executing

The Wait instruction does not stop even when the program is paused while the Wait instruction is executing. The Wait instruction ends when an event condition is satisfied or the specified time has passed. If the time is set by the Wait parameter, the passed time is reset and the program waits for the specified time when the program is restarted by selecting Run Window Continue.

See Also

AtHome, Cnv_QueLen, Ctr, ErrorOn, EstopOn, GetRobotInsideBox, GetRobotInsidePlane, In, InW, LatchState, LOF, Mask, MCalComplete, MemIn, MemInW, MemSw, Motor, Oport, Out, OutW, PauseOn, SaftyOn, Sw, TeachOn, TMOut, WindowsStatus, Tw, WorkQue_Len, SF_GetStatus

Wait 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.

```
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

' Waits until input 0 turns on
Wait Sw(0) = On

' Waits 60.5 secs and then continue execution
Wait 60.5

' Waits until input 0 is off and input 1 is on
Wait Sw(0) = Off And Sw(1) = On

' Waits until memory bit 0 is on or memory bit 1 is on
Wait MemSw(0) = On Or MemSw(1) = On

' Waits one second, then turn output 1 on
Wait 1; On 1

' Waits for the lower 3 bits of input port 0 to equal 1
Wait In(0) Mask 7 = 1

' Waits until the global Integer type variable giCounter is over 10
Wait giCounter > 10

' Waits ten seconds, until the global Long type variable glCheck is 30000
Wait glCheck = 30000, 10
```

WaitNet Statement

Wait for TCP/IP port connection to be established.

Syntax

WaitNet *#portNumber* [, *timeOut*]

Parameters

portNumber	Integer expression for TCP/IP port number to connect. Range is 201 to 216
timeOut	Optional. Maximum time to wait for connection.

See Also

OpenNet, CloseNet

WaitNet Statement Example

For this example, two controllers have their TCP/IP settings configured as follows:

Controller #1:
Port: #201
Host Name: 192.168.0.2
TCP/IP Port: 1000

```
Function tcpip
  OpenNet #201 As Server
  WaitNet #201
  Print #201, "Data from host 1"
Fend
```

Controller #2:
Port: #201
Host Name: 192.168.0.1
TCP/IP Port: 1000

```
Function tcpip
  String data$
  OpenNet #201 As Client
  WaitNet #201
  Input #201, data$
  Print "received '", data$, "' from host 1"
Fend
```

WaitPos Statement

Waits for robot to decelerate and stop at position before executing the next statement while path motion is active.

Syntax

WaitPos

Description

Normally, when path motion is active (CP On or CP parameter specified), the motion command starts the next statement as deceleration starts.
Use the 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   ' waits for robot to decelerate
On 1
CP Off
```

WaitSig Statement

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 63.
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 Statement Example

```
Function Main
  Xqt SubTask
  Wait 1
  Signal 1
  .
  .
Fend

Function SubTask
  WaitSig 1
  Print "signal received"
  .
Fend
```

Weight Statement

Specifies or displays the weight setting for the robot arm.

Syntax

Weight [*payloadWeight* [*, distance* | *S* | *T*]]
Weight

Parameters

payloadWeight	Optional (It is not possible to omit only *payloadWeight*). The weight of the end effector to be carried in Kg unit.
distance	Optional. 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 (including RS series).
S	Load weight against the additional S axis in kg to 2 decimal places)
T	Load weight against the additional T axis in kg to 2 decimal places)

Return Values

Displays the current Weight settings when parameters are omitted.
When [distance] is omitted, the entered [payloadWeight] will be set and the default value [distance] will be set.
It is not possible to omit only [payloadWeight].

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 robots (including RS series). It is the distance from the second arm rotation joint centerline to the hand/work piece combined center of gravity.
If the robot has the additional axis, the loads on the additional axis must be set with the S, T parameters.
If the equivalent value work piece weight calculated from specified parameters exceeds the maximum allowable payload, an error occurs.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.
You can also set by following "Weight, Inertia, and Eccentricity/offset Measurement Utility".
The following manual describes the details.
EPSON RC+ 7.0 User's Guide 6.18.12 Weight, Inertia, and Eccentricity/offset Measurement Utility

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. Once the value is set, the value is memorized in the controller.
When nothing is changed, it will remain at the previously set value.

See Also

Accel, Inertia

For details of hand, refer to the *Hand Function Manual.*

Weight Statement Example

This Weight instruction on the Command window displays the current setting.

```
> weight
2.000, 200.000
>
```

Sets the hand weight (3 kg) with the Weight statement

```
Weight 3.0
```

Sets the load weight on the additional S axis (30 kg) with the Weight statement

```
Weight 30.0, S
```

Weight Function

Returns a Weight parameter.

Syntax

Weight(*paramNumber*)

Parameters

paramNumber	Integer expression containing one of the values below:
	1: Payload weight
	2: Arm length
	3: Load on the additional S axis
	4: Load on the additional T axis

Return Values

Real number containing the parameter value.

See Also

Inertia, Weight

For details of Hand, refer to the *Hand Function Manual.*

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. Default is Local 0.

See Also

Joint, PList, Pulse

Where Statement Example

The display type can be different depending on the robot type and existence of additional axes.
The following example is for Scara robot without the additional axis.

```
>where
WORLD: X: 350.000 mm Y: 0.000 mm  Z: 0.000 mm  U: 0.000 deg V: 0.000 deg
W: 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
WORLD: X: 250.000 mm Y:-100.000 mm  Z: 0.000 mm U: 0.000 deg V: 0.000 deg
W: 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
```

WindowsStatus Function

Returns the Windows startup status.

Syntax

WindowsStatus

Return Values

Integer value representing the current Windows startup status. The Windows startup status is returned in a bit image and shows the following status.

Function name	System reservation	RC+ enabled	PC enabled
Bit number	15 to 2	1	0
Details of available functions		Vision Guide (Frame grabber type) RC+ API Fieldbus master	PC file PC RS-232C Data base access DLL call

Description

This function is used to check the controller startup status when the controller configuration is set to "Independent mode". When the controller configuration is set to "Cooperative mode", programs cannot be started until both RC+ function and PC function turn available.

Note

About the Controllers to use

It cannot be used with T/VT series.

WindowsStatus Function Example

```
Print "The current PC Booting up Status is: ", WindowsStatus
```

WOpen Statement

Opens a file for writing.

Syntax

WOpen *fileName* **As** *#fileNumber*
 .
 .
Close *#fileNumber*

Parameters

fileName	A string expression containing the path and file name. If path is omitted, the file in the current directory is specified. See ChDisk for the details.
fileNumber	Integer expression from 30 to 63

Description

Opens the specified file using the specified *fileNumber*. This statement is used to open and write data to the specified file. (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.

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#, Write, Seek, and Close.

Close closes the file and releases the file number.

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.

Notes

A network path is available.

File write buffering

File writing is buffered. The buffered data can be written with Flush statement. Also, when closing a file with Close statement, the buffered data can be written.

See Also

AOpen, BOpen, Close, Print#, ROpen, UOpen, FreeFile

WOpen Statement Example

```
Integer fileNum, i, j

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

fileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print "data = ", j
Next i
Close #fileNum
```

WorkQue_Add Statement

Adds the work queue data (point data and user data) to the specified work queue.

Syntax

WorkQue_Add *WorkQueNum*, *pointData* [, *userData*]

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
pointData	The point data to be added to the work queue.
userData	Optional. Real expression used to register the user data along with the point data. If omitted, 0 (real number) is registered as the user data.

Description

pointData and *useData* are added to the end of the work queue. When the Sort type is specified by WorkQue_Sort, however, they are registered according to the specified Sort type.

When the double registration prevention distance is set by WorkQue_Reject, the distance from the previously registered point data is calculated. If the point data is less than that distance, the point data and the user data are not added to the work queue. In this case, an error does not occur.

The upper limit of the work queue data is 1000. The work queue data is deleted by WorkQue_Remove when it is finished being used.

See Also

WorkQue_AutoRemove, WorkQue_Len, WorkQue_Reject, WorkQue_Remove, WorkQue_Sort

WorkQueAdd Statement Example

```
Integer x, y
Real u

P0 = XY(300, 300, 300, 90, 0, 180)
P1 = XY(200, 280, 150, 90, 0, 180)
P2 = XY(200, 330, 150, 90, 0, 180)
P3 = XY(-200, 280, 150, 90, 0, 180)

Pallet 1, P1, P2, P3, 10, 10
x = 1
y = 1
u = 5.3
WorkQue_Add 1, Pallet(1, x, y), u
```

WorkQue_AutoRemove Statement

Sets the auto delete function to the specified work queue.

Syntax

WorkQue_AutoRemove *WorkQueNum* ,{True | False}

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
True | False	True: Enables the auto delete function. False: Disables the auto delete function.

Description

Sets the auto delete function to the work queue. When the auto delete is enabled, the point data and the user data are automatically deleted from the work queue when the point data is obtained from the work queue by WorkQue_Get.
When the auto delete is disabled, the point data and the user data are not deleted. To delete them, use WorkQue_Remove.
The user data obtained by WorkQue_UserData are not deleted automatically.

Auto delete function can be set to each work queue.

See Also

WorkQue_AutoRemove Function, WorkQue_Get

WorkQue_AutoRemove Statement Example

```
WorkQue_AutoRemove 1, True
```

WorkQue_AutoRemove Function

Returns the state of the auto delete function set to the work queue.

Syntax

WorkQue_AutoRemove (*WorkQueNum*)

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.

Return Values

True if the auto delete function of the specified work queue is enabled, otherwise False.

See Also

WorkQue_AutoRemove, WorkQue_Get

WorkQue_AutoRemove Function Example

```
Boolean autoremove

autoremove = WorkQue_AutoRemove(1)
```

WorkQue_Get Function

Returns the point data from the specified work queue.

Syntax

WorkQue_Get(*WorkQueNum* [, *index*])

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
index	Optional. Integer expression that represents the index of the queue data to acquire. (the beginning of the index number is 0)

Return Values

The point data is returned from the specified work queue.

Description

Use WorkQue_Get to acquire the point data from the work queue. If the index is omitted, the first data of the queue data is returned. If the index is specified, the point data of the specified index is returned.

When the auto delete function is enabled by WorkQue_AutoRemove, the point data and the user data are deleted by WorkQue_Get.
When the auto delete is disabled, the point data and the user data are not deleted. To delete them, use WorkQue_Remove.

See Also

WorkQue_AutoRemove, WorkQue_Len, WorkQue_Reject, WorkQue_Remove, WorkQue_Sort

WorkQue_Get Function Example

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

WorkQue_Len Function

Returns the number of the valid work queue data registered to the specified work queue.

Syntax

WorkQue_Len(*WorkQueNum*)

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.

Return Values

The integer expression representing the number of registered valid work queue data.

Description

Returns the number of registered valid work queue data.

You can also use WorkQue_Len as an argument to the Wait statement.

See Also

WorkQue_Add, WorkQue_Get, WorkQue_Remove

WorkQue_Len Function Example

```
Do
  Do While WorkQue_Len(1) > 0
    WorkQue_Remove 1, 0
  Loop
  If WorkQue_Len(1) > 0 Then
    Jump WorkQue_Get(1, 0) C0
    On gripper
    Wait .1
    WorkQue_Remove 1, 0
    Jump place
    Off gripper
    Jump idlePos
  EndIf
Loop
```

WorkQue_List Statement

Displays the work queue data list (point data and user data) of the specified work queue.

Syntax

WorkQue_List *WorkQueNum* [, *numOfItems*]

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
numOfItems	Optional. Integer expression to specify how many items to display. If omitted, all items are displayed.

Note

This command will only work in the command window.

See Also

WorkQue_Add, WorkQue_Get, WorkQue_Remove

WorkQue_List Statement Example

From the command window:

```
> WorkQue_List 1
Queue 0    = XY(     1.000,    1.000,    0.000,    0.000 )  /R /0 (   0.000)
Queue 1    = XY(     3.000,    1.000,    0.000,    0.000 )  /R /0 (   2.000)
Queue 2    = XY(     4.000,    1.000,    0.000,    0.000 )  /R /0 (   3.000)
Queue 3    = XY(     5.000,    1.000,    0.000,    0.000 )  /R /0 (   4.000)
Queue 4    = XY(     6.000,    1.000,    0.000,    0.000 )  /R /0 (   5.000)
```

WorkQue_Reject Statement

Sets and displays the minimum distance for double registration prevention to the specified work queue.

Syntax

WorkQue_Reject *WorkQueNum* [, *rejectDistance*]

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
rejectDistance	Optional when being executed from the command window. Real expression specifying the minimum distance between parts allowed in the queue in millimeters. If omitted, the current *rejectDistance* is displayed.

Description

Use WorkQue_Reject to specify the minimum distance between parts to prevent double registration of the point data. The work queue cannot be registered when the point data less than the minimum distance is registered by WorkQue_Add. WorkQue_Reject helps the system filter out double registration. The default is 0 mm.

WorkQue_Reject should be executed before adding the work queue data (point data and user data) by WorkQue_Add.

Double registration prevention can be set for each work queue.

See Also

WorkQue_Add, WorkQue_Reject Function

WorkQue_Reject Statement Example

```
WorkQue_Reject 1, 2.5
```

WorkQue_Reject Function

Returns the distance of the double registration prevention set to the specified work queue.

Syntax

WorkQue_Reject (*WorkQueNum*)

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.

Return Values

Real value in millimeters

See Also

WorkQue_Add, WorkQue_Reject

WorkQue_Reject Function Example

```
Real rejectDist

RejectDist = WorkQue_Reject(1)
```

WorkQue_Remove Statement

Deletes the work queue data (point data and user data) from the specified work queue.

Syntax

WorkQue_Remove *WorkQueNum* [, *index* | All]

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
index	Optional. Integer expression that represents the index of the queue data to delete. (the beginning of the index number is 0). Specify All when deleting all the queue data from the work queue.

Description

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

See Also

WorkQue_Add

WorkQue_Remove Statement Example

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

' Remove the data from the WorkQueue
WorkQue_Remove 1
```

WorkQue_Sort Statement

Sets and displays the Sort type of the specified work queue.

Syntax

WorkQue_Sort *WorkQueNum [, SortMethod]*

Parameters

WorkQueNum Integer expression (1 to 16) that specifies the work queue number.

SortMethod Specify the Sort method with an integer expression or with the following constant. This can be omitted if executed from the command window. If omitted, the current Sort method is displayed.

Constant	Value	Description
QUE_SORT_NONE	0	No sorting (registration order to work queue)
QUE_SORT_POS_X	1	X coordinate ascending order
QUE_SORT_INV_X	2	X coordinate descending order
QUE_SORT_POS_Y	3	Y coordinate ascending order
QUE_SORT_INV_Y	4	Y coordinate descending order
QUE_SORT_POS_USER	5	User data (real value) ascending order
QUE_SORT_INV_USER	6	User data (real value) descending order

Description

Sets the Sort method to the work queue. When the point data and the user data are added by WorkQue_Add, they are registered to the work queue according to the specified Sort method.

When the user data is set again by WorkQue_UserData, the order of the work queues is changed according to the specified Sort method.

WorkQue_Sort should be executed before adding the work queue data (point data and user data) to the work queue data by WorkQue_Add.

WorkQue_Sort should be executed before setting the user data again by WorkQue_UserData.

Sort method can be set for each work queue.

See Also

WorkQue_Add, WorkQue_UserData

WorkQue_Sort Statement Example

```
WorkQue_Sort 1, QUE_SORT_POS_X
```

WorkQue_Sort Function

Returns the Sort method of the specified work queue.

Syntax

WorkQue_Sort (*WorkQueNum*)

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.

Return Values

An integer expression representing the Sort method set to the work queue.

4	= No sorting (registration order to work queue)
5	= X coordinate ascending order
6	= X coordinate descending order
7	= Y coordinate ascending order
8	= Y coordinate descending order
9	= User data (real value) ascending order
10	= User data (real value) descending order

See Also

WorkQue_Add, WorkQue_Sort, WorkQue_UserData

WorkQue_Sort Function Example

```
Integer quesort

quesort = WorkQue_Sort(1)
```

WorkQue_UserData Statement

Resets and displays the user data (real number) registered to the specified work queue.

Syntax

WorkQue_UserData *WorkQueNum* [, *index*] [, *userData*]

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
index	Integer expression that represents the index of the work queue data. (the beginning of the index number is 0). Optional when executing from the command window.
userData	Integer expression that represents the user data to be set again. This can be omitted when executed from the command window. If omitted, the current user data (real expression) is displayed.

Description

Resets and displays the user data currently registered to the work queue.

If the Sort method is specified by WorkQue_Sort, the order of the work queue data is changed according to the specified Sort method.

QUE_SORT_POS_USER : User data (real expression) ascending order
QUE_SORT_INV_USER : User data (real expression) descending order

See Also

WorkQue_UserData Function

WorkQue_UserData Example

```
WorkQue_UserData 1, 1, angle
```

WorkQue_UserData Function

Returns the user data (real value) registered to the specified work queue.

Syntax

WorkQue_UserData (*WorkQueNum* [, *index*])

Parameters

WorkQueNum	Integer expression (1 to 16) that specifies the work queue number.
index	Optional. Integer expression that represents the index of the work queue data. (the first index number is 0).

Return Values

Real value.

See Also

WorkQue_UserData

WorkQue_UserData Function Example

```
' Remove from queue
angle = WorkQue_UserData(1)   ' default to queue index of 0
Jump WorkQue_Get(1) :U(angle)
WorkQue_Remove 1
```

Wrist Statement

Sets the wrist orientation of a point.

Syntax

(1) **Wrist** *point* [, *Flip* | *NoFlip*]
(2) **Wrist**

Parameters

point	**P***number* or **P**(*expr*) or point label.
Flip | *NoFlip*	Representing wrist orientation.

Return Values

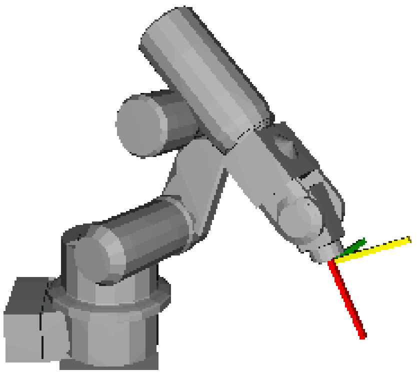
When both parameters are omitted, the wrist orientation is displayed for the current robot position.
If *Flip* | *NoFlip* is omitted, the wrist orientation for the specified point is displayed.

See Also

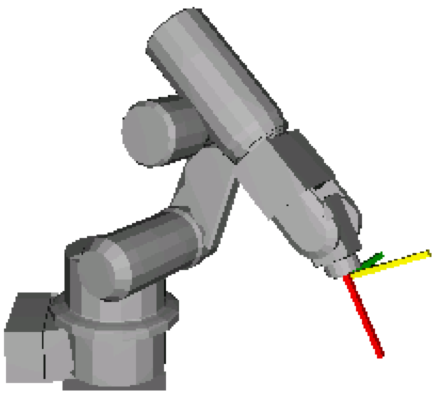
Elbow, Hand, J4Flag, J6Flag, Wrist Function

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

Returns the wrist orientation of a point.

Syntax

Wrist [(*point*)]

Parameters

point	Optional. **P***number* or **P**(*expr*) or point label or point expression. If *point* is omitted, 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

Writes characters to a file or communication port without end of line terminator.

Syntax

Write *#portNumber*, *string*

Parameters

portNumber	ID number that specifies the file or communications port. File number can be specified in ROpen, WOpen, AOpen statements. Communication port number can be specified in OpenCom (RS-232C) or OpenNet (TCP/IP) statements.
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.

Note

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

File write buffering

File writing is buffered. The buffered data can be written with Flush statement. Also, when closing a file with Close statement, the buffered data can be written.

See Also

Print, Read, WriteBin

Write Statement Example

```
OpenCom #1
For i = 1 to 10
    Write #1, data$(i)
Next i
CloseCom #1
```

WriteBin Statement

Writes binary data to a file or communications port.

Syntax

WriteBin *#portNumber*, *data*
WriteBin *#portNumber*, *array(), count*

Parameters

portNumber	ID number that specifies the file or communications port File number can be specified in BOpen statements. Communication port number can be specified in OpenCom (RS-232C) or OpenNet (TCP/IP) statements.
data	Integer expression containing the data to be written.
array()	Name of a byte, integer, or long array variable that contains the data bytes to be written. Specify a one dimension array variable.
count	Specifies the number of bytes to be written. The specified count has to be less than or equal to the number of array elements and also smaller than 256 bytes. If the communication port (TCP/IP) is the subject, the count has to be less than or equal to the number of array and also smaller than 1024 bytes.

Note

About the Controllers to use

For T/VT series, an error will occur at operation when RS-232C port of the Controller is specified.

See Also

ReadBin, Write

WriteBin Statement Example

```
Integer i, data(100)

OpenCom #1
For i = 0 To 100
  WriteBin #1, i
Next I
WriteBin #1, data(), 100
CloseCom #1
```

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.

Result

Returns a result of bitwise Xor operation.

Description

The Xor operator performs the bitwise Xor operation on the values of the operands. Each bit of the result is the Xored 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

Initiates execution of a task from within another task.

Syntax

Xqt [*taskNumber,*] *funcName* [**(***argList***)**] [, *Normal* | *NoPause* | *NoEmgAbort*]

Parameters

taskNumber	Optional. The task number for the task to be executed. The range of the task number is 1 to 32. For background tasks, specifies integer value from 65 to 80.
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.
taskType	Optional. Usually omitted. For background tasks, specifying a task type means nothing.
Normal	Executes a normal task.
NoPause	Executes a task that does not pause at Pause statement or Pause input signal occurrence or Safety Door Open.
NoEmgAbort	Executes a task that continues processing at Emergency Stop or error occurrence.

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

Task Type

Specify *NoPause* or *NoEmgAbort* as a task type to execute a task that monitors the whole controller.
However, be sure to use these special tasks based on the understanding of the task motion using SPEL$^+$ or restriction of special tasks.
For details of special tasks, refer to the section *Special Tasks* in the *EPSON RC+ User's Guide*.

Background task

When executing Xqt in a background task, the generated task is also the background task.
To execute the main function from a background task, use the StartMain statement.
The details of the background task is explained in the *EPSON RC+ Users Guide manual: 6.20 Special Task*.

Unavailable Commands in NoEmgAbort Task and background task

The following commands cannot be executed in NoEmgAbort task and background task.

A Accel
AccelR
AccelS
AIO_TrackingStart
AIO_TrackingEnd
Arc
Arc3
Arch
Arm
ArmCalib
ArmCalibCLR
ArmCalibSET
ArmClr
ArmSet
AutoLJM
AutoOrientationFlag
AvoidSingularity
B Base
BGo
BMove
Box
BoxClr
Brake
C Calib
Cnv_AbortTrack
Cnv_Accel
Cnv_AccelLim
Cnv_Adjust
Cnv_AdjustClear
Cnv_AdjustGet
Cnv_AdjustSet
Cnv_DownStream
Cnv_Fine
Cnv_Mode
Cnv_OffsetAngle
Cnv_QueAdd
Cnv_QueMove
Cnv_QueReject
Cnv_QueRemove
Cnv_QueUserData
Cnv_Trigger
Cnv_UpStream
CollisionDetect
CP
CP_Offset
Curve
CVMove
E ECP
ECPClr
ECPSet
F Find
Fine
FineDist
Force_Calibrate
Force_ClearTrigger
Force_Sensor
Force_SetTrigger
G Go
H Hand_On
Hand_Off
Home
HomeClr
HomeSet
Hordr
I Inertia
J JTran
Jump
Jump3
Jump3CP
JRange
L LatchEnable
LimitTorque
LimZ
LimZMargin
Local
LocalClr
M MCal
MCordr
Motor
Move
O OLAccel
P Pass
PerformMode
Pg_LSpeed
Pg_Scan
Plane
PlaneClr
Power
PTPBoost
Pulse
Q QP
QPDecelR
QPDecelS
R Range
Reset *1
Restart *2
S Sense
SetLatch
SFree
SingularityAngle
SingularityDist
SingularitySpeed
SLock
SoftCP
Speed
SpeedFactor
SpeedR
SpeedS
SyncRobots
T TC
TGo
Till
TLSet
TLClr
TMove
Tool
Trap
V VCal
VcalPoints
VCls
VCreateCalibration
VCreateObject
VCreateSequence
VDefArm
VDefGetMotionRange
VDefLocal
VDefSetMotionRange
VDefTool
VDeleteCalibration
VDeleteObject
VDeleteSeuence
VEditWindow
VGet
VGoCenter
VLoad
VLoadModel
VRun
VSave
VSaveImage
VSaveModel
VSet
VShowModel
VStasShow
VStatsReset
VStatsResetAll
VStatsSave
VSD
VStatsShow
VTeach
VTrain
W WaitPos
Weight
WorkQue_Add
WorkQue_Reject
WorkQue_Remove
WorkQue_Sort
WorkQue_UserData
X Xqt *3
XYLim

*1 Reset Error can be executed
*2 Executable from the Trap Error processing task
*3 Executable from the background tasks

DO NOT use XQT command repeatedly in Loop statements.

Do not use XQT command repeatedly in Loop statements such as Do…Loop.
The controller may freeze up. If you use Loop statements repeatedly, make sure to add Wait command (Wait 0.1).

See Also

Function/Fend, Halt, Resume, Quit, Startmain, Trap

Xqt Statement Example

```
Function main
    Xqt flash               '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

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 (including N series). Real expression representing the V coordinate.
w	Optional for 6-Axis robots (including N series). Real expression representing the W coordinate.

Return Values

A point constructed from the specified coordinates.

Description

When you don’t use the additional ST axis, there are nothing in particular to be care of.
You can move the manipulator to the specified coordinate with XY function like below:

```
  Go XY(60,30,-50,45)
```

When you use the additional ST axis, you need to be careful.
XY function returns the only robot point data, not including the additional axis.
If you use XY function lick this: Go XY(60,30,-50,45), the manipulator will move to the specified coordinate but the additional axis will not move. If you want to move the additional axis as well, specify like this: Go XY(60,30,-50,45) : ST(10,20).
For the details of additional axis, refer to *EPSON RC+ Users Guide: 21. Additional Axis*.

See Also

JA, Point Expression, ST Function

XY Function Example

```
P10 = XY(60, 30, -50, 45) + P20
```

XYLim Statement

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*.)

Result

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 monitor method configured in XYLimMode command. For details of monitor method, refer to XYLimMode statement.

Robot parameter data is stored in compact flash in controller. Therefore, writing to command flash occurs when executing this command. Frequent writing to compact flash affect to lifetime of compact flash. We recommend to use this command minimally.

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

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 by using the XYZ Limits page on the Robot Manager .

See Also

Range, XYLimMode

XYLim Statement Example

This simple example from the command 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

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

When "1" is specified for reference data, returns X axis lower limit position specified in XYLim as X of point data, Y axis lower limit position as Y, and Z axis lower limit position as Z.
When "2" is specified for reference data, returns X axis upper limit position specified in XYLim as X of point data, Y axis upper limit position as Y, and Z axis upper limit position as Z.

See Also

XYLim Statement

XYLim Function Example

```
P1 = XYLim(1)
P2 = XYLim(2)
```

XYLimClr Statement

Clears the XYLim definition.

Syntax

XYLimClr

See Also

XYLim, XYLimDef

XYLimClr Function Example

This example uses the XYLimClr function in a program:

```
Function ClearXYLim

   If XYLimDef = True Then
      XYLimClr
   EndIf
Fend
```

XYLimDef Function

Returns whether XYLim has been defined or not.

Syntax

XYLimDef

Return Values

True if XYLim has been defined, otherwise False.

See Also

XYLim, XYLimClr

XYLimDef Function Example

This example uses the XYLimDef function in a program:

```
Function ClearXYLim

   If XYLimDef = True Then
      XYLimClr
   EndIf
Fend
```

XYLimMode Statement

Sets or displays monitor method of XYLim.

Syntax

(1) XYLimMode *monitor method*
(2) XYLimMode

Parameters

Monitor method Integer expression represents monitor method of using XYLim.

Constant	Value	Description
XYLIM_STANDARD	0	Applies XYLim to endpoint of motion command. (There is no effect on Pulse.)
XYLIM_STRICT	1	Applies XYLim to monitor method of XYLIM_STANDARD, trajectory, and pulse motion.

Result

Displays monitor method of XYLim that currently configured when used without parameters.

Description

XYLimMode sets monitor method of XYLim for spcified robot.
When XYLIM_STANDARD is specified, the motion range set in XYLim is effective only for endpoint of motion command. From start point of motion to trajectory of endpoint is not applicable. Therefore, during operation, the arm may pass outside of the area that set in XYLim. In this mode, XYLim is not applied to pulse motion.
When XYLIM_STRICT is specified, the motion range set in XYLim is applied to endpoint of motion command and start point of motion to trajectory of endpoint. Therefore, during operation, if the arm tried to move out of the range set in the XYLim, an error will occur. In this mode, XYLim is applied to pulse motion. However, moving from outside the XYLim range to within the range, such as “the start point is outside the XYLim range and the endpoint is within the XYLim range”, it is possible to move without an outside XYLim range error.
To prevent robot interferes other devices, using with XYLIM_STRICT is recommended.

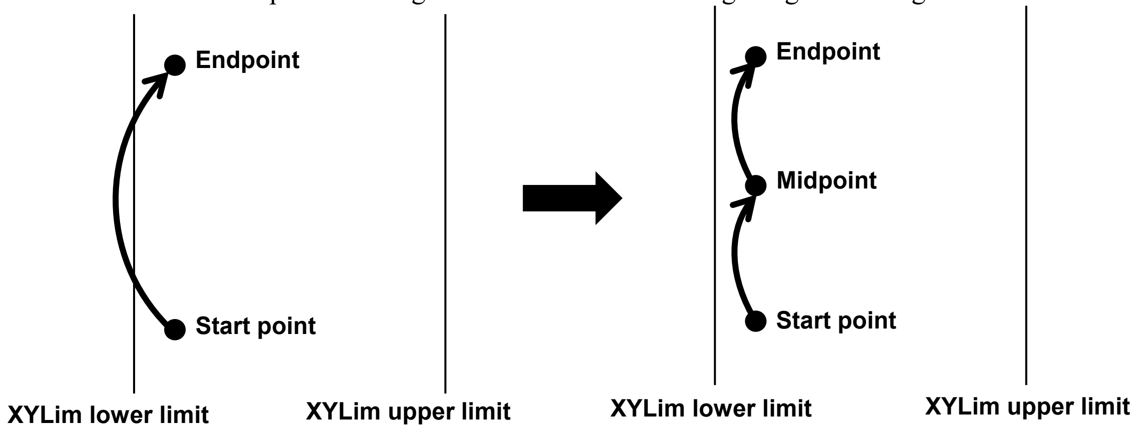
It is possible to change settings of default value of monitoring method when Controller start up, in the Controller preferences of EPSON RC+. The value confihured in XYLimMode command is enabled until Controller restart. When restarted Controller, the monitoring method of XYLim will be reset to the method specified in the Controller preferences.

⚠ CAUTION	■ For XYLIM_STANDARD, the arm may pass outside of the area that set in XYLim. Therefore, note that the robot does not interfere other devices. For example, setting extra space to XYLim and actual obstacle, operating near XYLim area, check trajectory at low speed.

Potential Errors

When executed PTP near XYLim area.

When executing PTP motion such as Go motion, start point and endpoint moves like the arrow of image on the left. Therefore, in XYLIM_STRICT, target coordinate is in the range but trajectory is out of range so an error will occur. In this case, avoid it so that the motion trajectory does not exceed XYLim by adding midPoint between start point and target coordinate like following image on the right.



When using past project:

If a project used with Controller FW Ver 7.5.2.0 or earlier than 7.5.52.0 is applied to a Controller that XYLim monitoring method is XYLIM_STRICT, the trajectory may become an out-of-range error. In this case, change the program to not exceed XYLim by adding midpoint.

See Also

XYLim

XYLimMode Command Example

This example uses XYLimMode. Moving current point to P1 with XYLIM_STANDARD and moving P1 to P2 with XYLIM_STRICT.

```
Function XYLimMode_sample
  Motor On
  XYLimMode XYLIM_STANDARD
  Go P1                  'Applying XYLim only for endpoint.
  XYLimMode XYLIM_STRICT
  Go P2                  'Applying XYLim to endpoint and trajectory.
Fend
```

This is an example using XYLimMode in command window. Displays current monitor method of XYLim.

```
> XYLimMode
1
```

XYLimMode Function

Acquire the monitor method of configured XYLim.

Syntax

XYLimMode

Return Values

Returns the monitor method of configured XYLim.
0 = Applying XYLim to endpoint of motion command. (not applying to pulse motion.)
1 = Applying XYLim to endpoint of motion command and trajectory. (applying to pulse motion.)

See Also

XYLimMode

XYLimMode Function Example

This is an example of using XYLimMode function. This is a program acquires monitor method of XYLim for function and displays it.

```
Function XYLimMode_sample
   Integer iVar

   iVar = XYLimMode
   Print iVar

Fend
```

Appendix A: SPEL+ Command Use Condition List

Command window	Command can be used in the command window.
Program	Command can be used as a statement in the SPEL+ program.
Function	Command can be used as a function.

Command		Command window		Program	Function
		RC+	TP3		
A	AbortMotion	✓	✓	✓	-
	Abs	-	-	✓	✓
	Accel	✓	✓	✓	✓
	AccelMax	-	-	✓	✓
	AccelR	✓	✓	✓	✓
	AccelS	✓	✓	✓	✓
	Acos	-	-	✓	✓
	Agl	-	-	✓	✓
	AglToPls	-	-	✓	✓
	AIO_In	✓	✓	-	✓
	AIO_InW	✓	✓	-	✓
	AIO_Out	✓	✓	✓	✓
	AIO_OutW	✓	✓	✓	✓
	AIO_Set	✓	✓	✓	✓
	AIO_TrackingSet	✓	✓	✓	-
	AIO_TrackingStart	-	-	✓	-
	AIO_TrackingEnd	-	-	✓	-
	AIO_TrackingOn	✓	✓	✓	✓
	Align	-	-	✓	✓
	AlignECP	-	-	✓	✓
	And	-	-	✓	-
	AOpen	✓	✓	✓	-
	Arc	✓	✓	✓	-
	Arc3	✓	✓	✓	-
	Arch	✓	✓	✓	✓
	AreaCorrection	✓	✓	✓	✓
	AreaCorrectionClr	✓	✓	✓	-
	AreaCorrectionDef	✓	✓	✓	✓
	AreaCorrectionInv	✓	✓	✓	✓
	AreaCorrectionOffset	✓	✓	✓	✓
	AreaCorrectionSet	✓	✓	✓	-
	Arm	✓	✓	✓	✓
	ArmClr	✓	✓	✓	-
	ArmDef	-	-	✓	✓
	ArmSet	✓	✓	✓	✓
	Asc	-	-	✓	✓
	ArmCalib	✓	✓	✓	✓
	ArmCalibClr	✓	✓	✓	-
	ArmCalibDef	-	-	✓	✓
	ArmCalibSet	✓	✓	✓	✓
	Asin	-	-	✓	✓
	Atan	-	-	✓	✓
	Atan2	-	-	✓	✓
	ATCLR	✓	✓	✓	-
	AtHome	-	-	✓	✓
	ATRQ	✓	✓	✓	✓
	AutoLJM	✓	✓	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	AvoidSingularity	✓	✓	✓	✓
B	Base	✓	✓	✓	✓
	BClr	-	-	✓	✓
	BClr64	-	-	✓	✓
	BGo	✓	✓	✓	-
	BMove	✓	✓	✓	-
	Boolean	-	-	✓	-
	BOpen	✓	✓	✓	-
	Box	✓	✓	✓	✓
	BoxClr	✓	✓	✓	-
	BoxDef	-	-	✓	✓
	Brake	✓	✓	✓ (Function only)	✓
	BSet	-	-	✓	✓
	BSet64	-	-	✓	✓
	BTst	-	-	✓	✓
	BTst64	-	-	✓	✓
	Byte	-	-	✓	-
C	Calib	✓	✓	✓	-
	Call	-	-	✓	-
	CalPls	✓	✓	✓	✓
	ChDir	✓	✓	✓	-
	ChDisk	✓	✓	✓	-
	ChDrive	✓	✓	✓	-
	ChkCom	-	-	✓	✓
	ChkNet	-	-	✓	✓
	Chr$	-	-	✓	✓
	ClearPoints	✓	✓	✓	-
	Close	✓	✓	✓	-
	CloseCom	✓	✓	✓	-
	CloseDB	✓	✓	✓	-
	CloseNet	✓	✓	✓	-
	Cls	✓	✓	✓	-
	Cnv_AbortTrack	✓	✓	✓	-
	Cnv_Accel	✓	✓	✓	✓
	Cnv_AccelLim	✓	✓	✓	✓
	Cnv_Adjust	✓	✓	✓	-
	Cnv_AdjustClear	✓	✓	✓	-
	Cnv_AdjustGet	✓	✓	✓	✓
	Cnv_AdjustSet	✓	✓	✓	-
	Cnv_Downstream	✓	✓	✓	✓
	Cnv_Fine	✓	✓	✓	✓
	Cnv_LPulse	-	-	✓	✓
	Cnv_Mode	✓	✓	✓	✓
	Cnv_Name$	-	-	✓	✓
	Cnv_Number	-	-	✓	✓
	Cnv_OffsetAngle	✓	✓	✓	✓
	Cnv_Point	-	-	✓	✓
	Cnv_PosErr	-	-	✓	✓
	Cnv_PosErrOffset	-	-	✓	✓
	Cnv_Pulse	-	-	✓	✓
	Cnv_QueAdd	✓	✓	✓	-
	Cnv_QueGet	-	-	✓	✓
	Cnv_QueLen	-	-	✓	✓
	Cnv_QueList	✓	✓	-	-
	Cnv_QueMove	✓	✓	✓	-
	Cnv_QueReject	✓	✓	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	Cnv_QueRemove	✓	✓	✓	-
	Cnv_QueUserData	✓	✓	✓	✓
	Cnv_RobotConveyor	-	-	✓	✓
	Cnv_Speed	-	-	✓	✓
	Cnv_Trigger	✓	✓	✓	-
	Cnv_Upstream	✓	✓	✓	✓
	CollisionDetect	✓	✓	✓	✓
	Cont	✓	-	✓	-
	Copy	✓	✓	✓	-
	Cos	-	-	✓	✓
	CP	✓	✓	✓	✓
	Ctr	-	-	✓	✓
	CTReset	✓	✓	✓	-
	CtrlDev	-	-	✓	✓
	CtrlInfo	-	-	✓	✓
	CurDir$	-	-	✓	✓
	CurDisk$	-	-	✓	✓
	CurDrive$	-	-	✓	✓
	CurPos	-	-	✓	✓
	Curve	✓	✓	✓	-
	CVMove	✓	✓	✓	-
	CP_Offset	✓	✓	✓	✓
	CR	✓	✓	✓	✓
	CS	✓	✓	✓	✓
	CT	✓	✓	✓	✓
	CU	✓	✓	✓	✓
	CV	✓	✓	✓	✓
	CW	✓	✓	✓	✓
	CX	✓	✓	✓	✓
	CY	✓	✓	✓	✓
	CZ	✓	✓	✓	✓
D	Date	✓	✓	✓	-
	Date$	-	-	✓	✓
	Declare	-	-	✓	-
	DegToRad	-	-	✓	✓
	Del	✓	✓	✓	-
	DeleteDB	✓	✓	✓	-
	DiffPoint	✓	✓	✓	✓
	DispDev	✓	✓	✓	✓
	Dist	-	-	✓	✓
	Do...Loop	-	-	✓	-
	Double	-	-	✓	-
E	ECP	✓	✓	✓	✓
	ECPClr	✓	✓	✓	-
	ECPDef	-	-	✓	✓
	ECPSet	✓	✓	✓	✓
	ElapsedTime	-	-	✓	✓
	Elbow	✓	✓	✓	✓
	Eof	-	-	✓	✓
	Era	-	-	✓	✓
	EResume	✓	✓	✓	-
	Erf$	-	-	✓	✓
	Erl	-	-	✓	✓
	Err	-	-	✓	✓
	Errb	✓	✓	✓	✓
	ErrMsg$	-	-	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	Error	✓	✓	✓	-
	ErrorOn	-	-	✓	✓
	Ert	-	-	✓	✓
	EStopOn	-	-	✓	✓
	Eval	-	-	✓	✓
	Exit	-	-	✓	-
	ExportPoints	✓	✓	✓	-
F	FbusIO_GetBusStatus	-	-	✓	✓
	FbusIO_GetDeviceStatus	-	-	✓	✓
	FbusIO_SendMsg	✓	✓	✓	-
	FileDataTime$	-	-	✓	✓
	FileExists	-	-	✓	✓
	FileLen	-	-	✓	✓
	Find	✓	✓	✓	-
	FindPos	-	-	✓	✓
	Fine	✓	✓	✓	✓
	FineDist	✓	✓	✓	✓
	FineStatus	✓	✓	✓	✓
	Fix	-	-	✓	✓
	Flush	✓	✓	✓	-
	FmtStr	✓	✓	✓	-
	FmtStr$	-	-	✓	✓
	FolderExists	-	-	✓	✓
	For...Next	-	-	✓	-
	Force_Calibrate	✓	✓	✓	-
	Force_ClearTrigger	✓	✓	✓	-
	Force_GetForce	-	-	✓	✓
	Force_GetForces	✓	✓	✓	-
	Force_Sensor	✓	✓	✓	✓
	Force_SetTrigger	✓	✓	✓	-
	FreeFile	-	-	✓	✓
	Function...Fend	-	-	✓	-
G	GClose	✓	✓	✓	-
	GetCurrentUser$	-	-	✓	✓
	GetRobotInsideBox	-	-	✓	✓
	GetRobotInsidePlane	-	-	✓	✓
	GGet	✓	✓	✓	-
	Global	-	-	✓	-
	Go	✓	✓	✓	-
	Gosub...Return	-	-	✓	-
	Goto	-	-	✓	-
	GSet	✓	✓	✓	-
	GShow	✓	✓	✓	-
	GShowDialog	-	-	✓	✓
H	Halt	-	-	✓	-
	Hand	✓	✓	✓	✓
	HealthCalcPeriod	✓	✓	✓	✓
	HealthCtrlAlarmOn	✓	✓	✓	✓
	HealthCtrlInfo	✓	✓	✓	✓
	HealthCtrlRateOffset	✓	✓	✓	-
	HealthCtrlReset	✓	✓	✓	-
	HealthCtrlWarningEnable	✓	✓	✓	✓
	HealthRateCtrlInfo	✓	✓	✓	✓
	HealthRateRBInfo	✓	✓	✓	✓
	HealthRBAlarmOn	✓	✓	✓	✓
	HealthRBAnalysis	✓	✓	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	HealthRBDistance	✓	✓	✓	✓
	HealthRBInfo	✓	✓	✓	✓
	HealthRBRateOffset	✓	✓	✓	-
	HealthRBReset	✓	✓	✓	-
	HealthRBSpeed	✓	✓	✓	✓
	HealthRBStart	✓	✓	✓	-
	HealthRBStop	✓	✓	✓	-
	HealthRBTRQ	✓	✓	✓	✓
	HealthRBWarningEnable	✓	✓	✓	✓
	Here	✓	✓	✓	✓
	Hex$	-	-	✓	✓
	Hofs	✓	✓	✓	✓
	HofsJointAccuracy	✓	✓	✓	-
	Home	✓	✓	✓	-
	HomeClr	✓	✓	✓	-
	HomeDef	-	-	✓	✓
	HomeSet	✓	✓	✓	✓
	Hordr	✓	✓	✓	✓
	Hour	✓	✓	✓	✓
I	If...Then..Else…EndIf	-	-	✓	-
	ImportPoints	✓	✓	✓	-
	In	-	-	✓	✓
	InBCD	-	-	✓	✓
	Inertia	✓	✓	✓	✓
	InPos	-	-	✓	✓
	Input	✓	✓	✓	-
	Input #	✓	✓	✓	-
	InputBox	✓	✓	✓	-
	InReal	-	-	✓	✓
	InsideBox	-	-	✓	✓
	InsidePlane	-	-	✓	✓
	InStr	-	-	✓	✓
	Int	-	-	✓	✓
	Int32	-	-	✓	-
	Integer	-	-	✓	-
	InW	-	-	✓	✓
	IODef	-	-	✓	✓
	IOLabel$	-	-	✓	✓
	IONumber	-	-	✓	✓
J	J1Angle	✓	✓	✓	✓
	J4Angle	✓	✓	✓	✓
	J1Flag	✓	✓	✓	✓
	J2Flag	✓	✓	✓	✓
	J4Flag	✓	✓	✓	✓
	J6Flag	✓	✓	✓	✓
	JA	-	-	✓	✓
	Joint	✓	✓	✓	-
	JointAccuracy	✓	✓	✓	✓
	JRange	✓	✓	✓	✓
	JS	-	-	✓	✓
	JT	-	-	✓	✓
	JTran	✓	✓	✓	-
	Jump	✓	✓	✓	-
	Jump3	✓	✓	✓	-
	Jump3CP	✓	✓	✓	-
	JumpTLZ	✓	✓	✓	-

Command		Command window		Program	Function
		RC+	TP3		
L	LatchEnable	✓	✓	✓	-
	LatchPos	-	-	✓	✓
	LatchState	-	-	✓	✓
	LCase$	-	-	✓	✓
	Left$	-	-	✓	✓
	Len	-	-	✓	✓
	LimitTorque	✓	✓	✓	✓
	LimitTorqueLP	✓	✓	✓	✓
	LimitTorqueStop	✓	✓	✓	✓
	LimitTorqueStopLP	✓	✓	✓	✓
	LimZ	✓	✓	✓	✓
	LimZMargin	✓	✓	✓	✓
	Line Input	✓	✓	✓	-
	Line Input #	✓	✓	✓	-
	LJM	-	-	✓	✓
	LoadPoints	✓	✓	✓	-
	Local	✓	✓	✓	✓
	LocalClr	✓	✓	✓	-
	LocalDef	-	-	✓	✓
	Lof	-	-	✓	✓
	LogIn	-	-	✓	✓
	Long	-	-	✓	-
	LSet$	-	-	✓	✓
	LShift	-	-	✓	✓
	LShift64	-	-	✓	✓
	LTrim$	-	-	✓	✓
M	Mask	-	-	✓	-
	MCal	✓	✓	✓	-
	MCalComplete	-	-	✓	✓
	MCordr	✓	✓	✓	✓
	MemIn	-	-	✓	✓
	MemInW	-	-	✓	✓
	MemOff	✓	✓	✓	-
	MemOn	✓	✓	✓	-
	MemOut	✓	✓	✓	-
	MemOutW	✓	✓	✓	-
	MemSw	-	-	✓	✓
	Mid$	-	-	✓	✓
	MHour	-	-	✓	✓
	MkDir	✓	✓	✓	-
	Mod	-	-	✓	-
	Motor	✓	✓	✓	✓
	Move	✓	✓	✓	-
	MsgBox	✓	✓	✓	✓
	MyTask	-	-	✓	✓
N	Next	-	-	✓	-
	Not	-	-	✓	-
O	Off	✓	✓	✓	-
	OLAccel	✓	✓	✓	✓
	OLRate	✓	✓	✓	✓
	On	✓	✓	✓	-
	OnErr	-	-	✓	-
	OpBCD	✓	✓	✓	-
	OpenCom	✓	✓	✓	✓
	OpenDB	✓	✓	✓	-
	OpenNet	✓	✓	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	Oport	-	-	✓	✓
	Or	-	-	✓	-
	Out	✓	✓	✓	✓
	OutReal	✓	✓	✓	✓
	OutW	✓	✓	✓	✓
P	P#	✓	✓	✓	-
	PAgl	-	-	✓	✓
	Pallet	✓	✓	✓	✓
	PalletClr	✓	✓	✓	-
	ParseStr	✓	✓	✓	✓
	Pass	✓	✓	✓	-
	Pause	-	-	✓	-
	PauseOn	-	-	✓	✓
	PDescription	✓	✓	✓	-
	PDescription$	✓	✓	-	✓
	PDef	-	-	✓	✓
	PDel	✓	✓	✓	-
	PerformMode	✓	✓	✓	✓
	PG_FastStop	✓	✓	✓	-
	PG_LSpeed	✓	✓	✓	✓
	PG_Scan	✓	✓	✓	-
	PG_SlowStop	✓	✓	✓	-
	PLabel	✓	✓	✓	-
	PLabel$	-	-	✓	✓
	Plane	✓	✓	✓	✓
	PlaneClr	✓	✓	✓	-
	PlaneDef	-	-	✓	✓
	PList	✓	✓	✓	-
	PLocal	✓	✓	✓	✓
	Pls	-	-	✓	✓
	PNumber	-	-	✓	✓
	PosFound	-	-	✓	✓
	Power	✓	✓	✓	✓
	PPls	-	-	✓	✓
	Preserve	-	-	✓	-
	Print	✓	✓	✓	-
	Print #	✓	✓	✓	-
	PTCLR	✓	✓	✓	-
	PTPBoost	✓	✓	✓	✓
	PTPBoostOK	-	-	✓	✓
	PTPTime	-	-	✓	✓
	PTran	✓	✓	✓	-
	PTRQ	✓	✓	✓	✓
	Pulse	✓	✓	✓	✓
Q	QP	✓	✓	✓	-
	QPDecelR	✓	✓	✓	✓
	QPDecelS	✓	✓	✓	✓
	Quit	-	-	✓	-
R	RadToDeg	-	-	✓	✓
	Randmize	✓	✓	✓	-
	Range	✓	✓	✓	-
	Read	✓	✓	✓	-
	ReadBin	✓	✓	✓	-
	Real	-	-	✓	-
	RealAccel	-	-	✓	✓
	RealPls	-	-	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	RealPos	-	-	✓	✓
	RealTorque	-	-	✓	✓
	Recover	✓	-	✓	✓
	RecoverPos	-	-	✓	✓
	Redim	✓	✓	✓	-
	Rename	✓	✓	✓	-
	RenDir	✓	✓	✓	-
	Reset	✓	✓	✓	-
	ResetElapsedTime	✓	✓	✓	-
	Restart	✓	-	✓	-
	Resume	-	-	✓	-
	Return	-	-	✓	-
	Right$	-	-	✓	✓
	RmDir	✓	✓	✓	-
	Rnd	-	-	✓	✓
	Robot	✓	✓	✓	✓
	RobotInfo	-	-	✓	✓
	RobotInfo$	-	-	✓	✓
	RobotModel$	-	-	✓	✓
	RobotName$	-	-	✓	✓
	RobotSerial$	-	-	✓	✓
	RobotType	-	-	✓	✓
	ROpen	✓	✓	✓	-
	ROTOK	✓	✓	✓	✓
	RSet$	-	-	✓	✓
	RShift64	-	-	✓	✓
	RShift	-	-	✓	✓
	RTrim$	-	-	✓	✓
	RunDialog	-	-	✓	-
S	SafetyOn	-	-	✓	✓
	SavePoints	✓	✓	✓	-
	Seek	✓	✓	✓	-
	Select...Send	-	-	✓	-
	SelectDB	✓	✓	✓	✓
	Sense	✓	✓	✓	-
	SetCom	✓	✓	✓	-
	SetIn	✓	✓	✓	-
	SetInReal	✓	✓	✓	-
	SetInW	✓	✓	✓	-
	SetLatch	✓	✓	✓	-
	SetNet	✓	✓	✓	-
	SetSw	✓	✓	✓	-
	SF_GetParam	○	○	○	○
	SF_GetParam$	○	○	○	○
	SF_GetStatus	○	○	○	○
	SF_LimitSpeedS	○	○	○	○
	SF_LimitSpeedSEnable	○	○	○	○
	SF_RealSpeedS	○	○	○	○
	SF_PeakSpeedS	○	○	○	○
	SF_PeakSpeedSClear	○	○	○	-
	SFree	✓	✓	✓	✓
	Sgn	-	-	✓	✓
	Short	-	-	✓	-
	Shutdown	✓	✓	✓	✓
	Signal	✓	✓	✓	-

Command		Command window		Program	Function
		RC+	TP3		
	SimGet	-	-	✓	-
	SimSet	✓	✓	✓	-
	Sin	-	-	✓	✓
	SingularityAngle	✓	✓	✓	✓
	SingularityDist	✓	✓	✓	✓
	SingularitySpeed	✓	✓	✓	✓
	SLock	✓	✓	✓	-
	SoftCP	✓	✓	✓	✓
	Space$	-	-	✓	✓
	Speed	✓	✓	✓	✓
	SpeedFactor	✓	✓	✓	✓
	SpeedR	✓	✓	✓	✓
	SpeedS	✓	✓	✓	✓
	SPELCom_Event	✓	✓	✓	-
	Sqr	-	-	✓	✓
	ST	-	-	✓	✓
	StartMain	-	-	✓	-
	Stat	-	-	✓	✓
	Str$	-	-	✓	✓
	String	-	-	✓	-
	Sw	-	-	✓	✓
	SyncLock	-	-	✓	-
	SyncUnlock	-	-	✓	-
	SyncRobots	✓	✓	✓	✓
	SysConfig	✓	✓	-	-
	SysErr	-	-	✓	✓
T	Tab$	-	-	✓	✓
	Tan	-	-	✓	✓
	TargetOK	-	-	✓	✓
	TaskDone	-	-	✓	✓
	TaskInfo	-	-	✓	✓
	TaskInfo$	-	-	✓	✓
	TaskState	✓	✓	✓	✓
	TaskWait	✓	✓	✓	-
	TC	✓	✓	✓	-
	TCLim	✓	✓	✓	✓
	TCPSpeed	-	-	✓	✓
	TCSpeed	✓	✓	✓	✓
	TeachOn	-	-	✓	✓
	TGo	✓	✓	✓	-
	Till	✓	✓	✓	-
	TillOn	-	-	✓	✓
	Time	✓	✓	✓	✓
	Time$	-	-	✓	✓
	TLClr	✓	✓	✓	-
	TLDef	-	-	✓	✓
	TLSet	✓	✓	✓	✓
	TMOut	✓	✓	✓	-
	TMove	✓	✓	✓	-
	Tmr	-	-	✓	✓
	TmReset	✓	✓	✓	-
	Toff	✓	✓	✓	-
	Ton	✓	✓	✓	-
	Tool	✓	✓	✓	✓
	Trap	-	-	✓	-
	Trim$	-	-	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	TW	-	-	✓	✓
U	UBound	-	-	✓	✓
	UByte	-	-	✓	-
	UCase$	-	-	✓	✓
	UInt32	-	-	✓	-
	UOpen	✓	✓	✓	-
	UpdateDB	✓	✓	✓	-
	UShort	-	-	✓	-
V	Val	-	-	✓	✓
	VCal	✓	✓	✓	-
	VCalPoints	✓	✓	✓	-
	VCls	-	-	✓	-
	VCreateCalibration	-	-	✓	-
	VCreateObject	-	-	✓	-
	VCreateSequence	-	-	✓	-
	VDefArm	-	-	✓	-
	VDefGetMotionRange	✓	-	✓	-
	VDefLocal	-	-	✓	-
	VDefSetMotionRange	✓	-	✓	-
	VDefTool	-	-	✓	-
	VDeleteCalibration	-	-	✓	-
	VDeleteObject	-	-	✓	-
	VDeleteSequence	-	-	✓	-
	VGet	-	-	✓	-
	VGoCenter	-	-	✓	-
	VisCalib	-	-	-	-
	VisCalInfo	-	-	-	✓
	VisCalLoad	-	-	-	-
	VisCalSave	-	-	-	-
	VisTrans	-	-	-	✓
	VLoad	-	-	✓	-
	VLoadModel	-	-	✓	-
	VRun	-	-	✓	-
	VSave	-	-	✓	-
	VSaveImage	-	-	✓	-
	VSaveModel	-	-	✓	-
	VSD	✓	✓	✓	✓
	VSet	-	-	✓	-
	VShowModel	✓	-	✓	-
	VStatsReset	-	-	✓	-
	VStatsResetAll	-	-	✓	-
	VStatsSave	-	-	✓	-
	VStatsShow	✓	✓	✓	-
	VTeach	-	-	✓	-
	VTrain	✓	-	✓	-
	VxCalib	✓	✓	✓	-
	VxCalDelete	✓	✓	✓	-
	VxCalLoad	✓	✓	✓	-
	VxCalInfo	-	-	✓	✓
	VxCalSave	✓	✓	✓	-
	VxTrans	-	-	✓	✓
W	Wait	✓	✓	✓	-
	WaitNet	✓	✓	✓	-
	WaitPos	✓	✓	✓	-
	WaitSig	✓	✓	✓	-
	Weight	✓	✓	✓	✓

Command		Command window		Program	Function
		RC+	TP3		
	Where	✓	✓	-	-
	WindowStatus	-	-	✓	✓
	WorkQue_Add	✓	✓	✓	-
	WorkQue_AutoRemove	✓	✓	✓	✓
	WorkQue_Get	✓	✓	✓	✓
	WorkQue_Len	✓	✓	✓	✓
	WorkQue_List	✓	✓	-	-
	WorkQue_Reject	✓	✓	✓	✓
	WorkQue_Remove	✓	✓	✓	-
	WorkQue_Sort	✓	✓	✓	✓
	WorkQue_UserData	✓	✓	✓	✓
	WOpen	✓	✓	✓	-
	Wrist	✓	✓	✓	✓
	Write	✓	✓	✓	-
	WriteBin	✓	✓	✓	-
X	Xor	-	-	✓	-
	Xqt	-	-	✓	-
	XY	-	-	✓	✓
	XYLim	✓	✓	✓	✓
	XYLimClr	✓	✓	✓	-
	XYLimDef	-	-	✓	✓
	XYLimMode	✓	✓	✓	✓

Appendix B: Precaution of Compatibility

B-1: Precaution of EPSON RC+ 6.0 Compatibility

Overview

This section contains information for customers using EPSON RC+ 7.0 with RC700 series Controller that have already used EPSON RC+ 6.0 with RC620.
EPSON RC+ 7.0 and EPSON RC+ 6.0 differ in such as hardware, adaptable manipulators, number of joint allowance, and software execution environment. Please read this section and understand the contents for the safety use of the Robot system.
EPSON RC+ 7.0 is improved software that has compatibility with products before EPSON RC+ 7.0 and designed to innovate advanced software technologies. However, some parts do not have compatibility with EPSON RC+ 6.0 or have been deleted to specialize in the robot controller and for ease of use.
The following compatibility is indicated based on EPSON RC+ 6.0 compared to EPSON RC+ 7.0.

General Differences

General differences of EPSON RC+ 6.0 and EPSON RC+ 7.0 are as follows.

Item	EPSON RC+ 7.0	EPSON RC+ 6.0
Number of task	Up to 32 tasks (Background task : Up to 16 tasks)	Up to 32 tasks (Background task : Up to 16 tasks)
Type of task	Able to specify NoPouse task Able to specify NoEmgAbort task Able to specify Background task	Able to specify NoPouse task Able to specify NoEmgAbort task Able to specify Background task
Special TRAP such as TRAP ERROR	Supported	Supported
Task starts by TRAP number	Dedicated task number	Dedicated task number
Multi Manipulator	Supported	Supported
Robot number	1 to 16	1 to 16
Number of significant figure for Real type	6 digits	6 digits
Number of significant figure for Double type	14 digits	14 digits
Array elements number	Other than string variable Local variable 2,000 Global variable 100,0000 Module variable 100,0000 Global Preserve variable 4,000	Other than string variable Local variable 2,000 Global variable 100,0000 Module variable 100,0000 Global Preserve variable 4,000
	String variable Local variable 200 Global variable 10,000 Module variable 10,000 Global Preserve variable 400	String variable Local variable 200 Global variable 10,000 Module variable 10,000 Global Preserve variable 400
Device number	21:PC 22:REMOTE 24:TP 20:TP3	21:PC 22:REMOTE 24:TP 28:LCD

Item	EPSON RC+ 7.0	EPSON RC+ 6.0
Control device	Remote I/O PC REMOTE COM REMOTE Ethernet TP3	Remote I/O PC
Timer number range	0 to 63	0 to 63
Program capacity	8 MB	8 MB
Signal No range for SyncLock, SyncUnlock	0 to 63	0 to 63
Signal No range for WaitSig, Signal	0 to 63	0 to 63
Memory I/O port	1024	1024
I/O port number	Common with EPSON RC+ 6.0	
Port No of Ethernet	201 to 216	201 to 216
Remote I/O assignment	Assigned as default	Default: --
Port No of RS-232C communication	1 to 8, 1001 to 1008	1to 8, 1001, 1002
OpenCom execution of RS-232C communication port	Mandatory	Mandatory
Input/output to files	Supported	Supported
File number	30 to 63	30 to 63
Access number for the database	501 to 508	501 to 508
Vision Guide	Network camera type Frame grubber type	Network camera type Frame grubber type
Conveyor tracking	Supported	Supported
PG robot	Supported	Supported
OCR	Supported	Supported
Security	Supported	Supported
VBGuide 6.0 (RC+ API 7.0)	Supported	Supported
Fieldbus I/O	Use normal I/O commands	Use normal I/O commands
Fieldbus master	Response is not guaranteed	Response is not guaranteed
Fieldbus slave	Response is guaranteed	Response is guaranteed
GUI Builder	Supported	Supported
Error number	Common with EPSON RC+ 6.0	

Compatibility List of Commands

+	Function expansion / function changes have been made with upper compatibility.
–	No changes.
!	Pay attention. Function changes or syntax changes have been made.
!!	Pay attention. Significant changes have been made.
×	Deleted.

Command		Compatibility	Note
A	Abs Function	–	
	Accel Statement	–	
	Accel Function	–	
	AccelMax Statement	–	
	AccelR Statement	–	
	AccelR Function	–	
	AccelS Statement	–	
	AccelS Function	–	
	Acos Function	–	
	AglToPls Function	–	
	Agl Function	–	
	AlignECP Function	–	
	Align Function	–	
	And Statement	–	
	Arc Statement	–	
	Arc3 Statement	–	
	Arch Statement	–	
	Arch Function	–	
	Arm Statement	–	
	Arm Function	–	
	ArmClr Statement	–	
	ArmDef Function	–	
	ArmSet Statement	–	
	ArmSet Function	–	
	Asc Function	–	
	Asin Function	–	
	Atan Function	–	
	Atan2 Function	–	
	ATCLR Statement	–	
	ATRQ Statement	–	
	ATRQ Function	–	
B	Base Statement	–	
	Base Function	–	
	BClr Function	–	
	BGo Statement	+	Added PerformMode parameter.
	BMove Statement	–	
	Boolean Statement	–	
	Box Statement	+	Added the remote output logic designation
	Box Function	–	
	BoxClr Function	–	

Command		Compatibility	Note
	BoxDef Function	–	
	Brake Statement	–	
	Brake Function	–	
	BSet Function	–	
	BTst Function	–	
	Byte Statement	–	
C	Call Statement	–	
	ChkCom Function	–	
	ChkNet Function	–	
	Chr$ Function	–	
	ClearPoints Statement	–	
	CloseCom Statement	–	
	CloseNet Statement	–	
	Cls Statement	–	
	Cos Function	–	
	CP Statement	–	
	CP Function	–	
	CTReset Statement	–	
	Ctr Function	–	
	CtrlDev Function	–	
	CtrlInfo Function	–	
	CurPos Function	–	
	Curve Statement	–	
	CVMove Statement	–	
	CX to CW Statement	–	
	CX to CW Function	–	
D	Date Statement	–	
	Date$ Function	–	
	DegToRad Function	–	
	DispDev Statement	–	
	DispDev Function	–	
	Dist Function	–	
	Do...Loop Statement	–	
	Double Statement	–	
E	ECP Statement	–	
	ECP Function	–	
	ECPClr Statement	–	
	EcpDef Function	–	
	ECPSet Statement	–	
	ECPSet Function	–	
	Elbow Statement	–	
	Elbow Function	–	
	Era Function	–	
	Erase Statement	–	
	EResume Statement	–	
	Erf$ Function	–	
	Erl Function	–	

Command		Compatibility	Note
	Err Function	–	
	ErrMsg$ Function	–	
	Error Statement	–	
	ErrorOn Function	–	
	Ert Function	–	
	EStopOn Function	–	
	Exit Statement	–	
F	Find Statement	–	
	FindPos Function	–	
	Fine Statement	–	
	Fine Function	–	
	Fix Function	–	
	FmtStr$ Statement	–	
	For...Next	–	
	Function...Fend	–	
G	Global Statement	+	Added PerformMode parameter.
	Go Statement	–	
	Gosub...Return	–	
	Goto Statement	–	
H	Halt Statement	–	
	Hand Statement	–	
	Hand Function	–	
	Here Statement	–	
	Here Function	–	
	Hex$ Function	–	
	Home Statement	–	
	HomeClr Statement	–	
	HomeDef Function	–	
	HomeSet Statement	–	
	HomeSet Function	–	
	HOrdr Statement	–	
	HOrdr Function	–	
	Hour Statement	–	
	Hour Function	–	
I	If...EndIf	–	
	In Function	–	
	InBCD Function	–	
	Inertia Statement	–	
	Inertia Function	–	
	InPos Function	–	
	Input Statement	–	
	Input# Statement	–	
	InsideBox Function	–	
	InsidePlane Function	–	
	InStr Function	–	
	Int Function	–	
	Integer Statement	–	

Command		Compatibility	Note
	InW Function	–	
	IOLabel$ Function	–	
	IONumber Function	–	
	IONumber Function	–	
J	J1Flag Statement	–	
	J1Flag Function	–	
	J2Flag Statement	–	
	J2Flag Function	–	
	J4Flag Statement	–	
	J4Flag Function	–	
	J6Flag Statement	–	
	J6Flag Function	–	
	JA Function	–	
	Joint	–	
	JRange Statement	–	
	JRange Function	–	
	JS Function	–	
	JT Function	–	
	JTran Statement	–	
	Jump Statement	+	Added PerformMode parameter.
	Jump3 Statement	–	
	Jump3CP Statement	–	
L	LCase$ Function	–	
	Left$ Function	–	
	Len Function	–	
	LimZ Statement	–	
	LimZ Function	–	
	Line Input Statement	–	
	Line Input# Statement	–	
	LJM Function	–	
	LoadPoints	–	
	Local Statement	–	
	Local Function	–	
	LocalClr Statement	–	
	LocalDef Function	–	
	Lof Function	–	
	Long Statement	–	
	LSet$ Function	–	
	LShift Function	–	
	LTrim$ Function	–	
M	Mask Operator	–	
	MemIn Function	–	
	MemInW Function	–	
	MemOff Statement	–	
	MemOn Statement	–	
	MemOut Statement	–	
	MemOutW Statement	–	

Command		Compatibility	Note
	MemSw Function	–	
	Mid$ Function	–	
	Mod Operator	–	
	Motor Statement	–	
	Motor Function	–	
	Move Statement	–	
	MyTask Function	–	
N	Not Operator	–	
O	Off Statement	–	
	OLAccel Statement	–	
	OLAccel Function	–	
	OLRate Statement	–	
	OLRate Function	–	
	On Statement	–	
	OnErr	–	
	OpBCD Statement	–	
	OpenCom Statement	–	
	OpenNet Statement	–	
	Oport Function	–	
	Or Operator	–	
	Out Statement	–	
	Out Function	–	
	OutW Statement	–	
	OutW Function	–	
P	PAgl Function	–	
	Pallet Statement	+	Added coordinate value designation
	Pallet Function	–	
	ParsStr Statement	–	
	ParsStr Function	–	
	Pass Statement	–	
	Pause Statement	–	
	PauseOn Function	–	
	PDef Function	–	
	PDel	–	
	PLabel$ Function	–	
	PLabel Statement	–	
	Plane Statement	–	
	Plane Function	–	
	PlaneClr Statement	–	
	PlaneDef Function	–	
	PList Statement	–	
	PLocal Statement	–	
	PLocal Function	–	
	Pls Function	–	
	PNumber Function	–	
	PosFound Function	–	
	Power Statement	–	

Command		Compatibility	Note
	Power Function	–	
	PPls Function	–	
	Print Statement	–	
	Print# Statement	–	
	PTCLR Statement	–	
	PTPBoost Statement	–	
	PTPBoost Function	–	
	PTPBoostOK Function	–	
	PTPTime Function	–	
	PTran Statement	–	
	PTRQ Statement	–	
	PTRQ Function	–	
	Pulse Statement	–	
	Pulse Function	–	
Q	QP Statement	–	
	Quit Statement	–	
R	RadToDeg Function	–	
	Randmize Statement	–	
	Range Statement	–	
	Read Statement	–	
	ReadBin Statement	–	
	Real Statement	–	
	RealPls Function	–	
	RealPos Function	–	
	RealTorque Statement	–	
	Redim Statement	–	
	Reset Statement	–	
	Resume Statement	–	
	Return Statement	–	
	RobotInfo Function	–	
	RobotInfo$ Function	–	
	RobotModel$ Function	–	
	RobotName$ Function	–	
	RobotSerial$ Function	–	
	RobotType Function	–	
	RSet$ Function	–	
	RShift Function	–	
	RTrim$ Function	–	
S	SafetyOn Function	–	
	SavePoints Statement	–	
	Select...Send Statement	–	
	Sense Statement	–	
	SetCom Statement	–	
	SetInW Statement	–	
	SetIn Statement	–	
	SetNet Statement	–	
	SetSw Statement	–	

Command		Compatibility	Note
	SFree Statement	–	
	SFree Function	–	
	Sgn Function	–	
	Signal Statement	–	
	Sin Function	–	
	SLock Statement	–	
	SoftCP Statement	–	
	SoftCP Function	–	
	Space$ Function	–	
	Speed Statement	–	
	Speed Function	–	
	SpeedR Statement	–	
	SpeedR Function	–	
	SpeedS Statement	–	
	SpeedS Function	–	
	SPELCom_Event Statement	–	
	Sqr Function	–	
	Stat Function	–	
	Str$ Function	–	
	String Statement	–	
	Sw Function	–	
	SyncLock Statement	–	
	SyncUnlock Statement	–	
	SysConfig Statement	–	
	SysErr Function	–	
T	Tab$ Function	–	
	Tan Function	–	
	TargetOK Function	–	
	TaskDone Function	–	
	TaskInfo Function	–	
	TaskInfo$ Function	–	
	TaskState Statement	–	
	TaskState Function	–	
	TaskWait Statement	–	
	TC Statement	–	
	TCLim Statement	–	
	TCLim Function	–	
	TCSpeed Statement	–	
	TCSpeed Function	–	
	TGo Statement	+	Added PerformMode parameter.
	TillOn Function	–	
	Time Command	–	
	Time Function	–	
	Time$ Function	–	
	TLClr Statement	–	
	TlDef Function	–	

Command		Compatibility	Note
	TLSet Statement	–	
	TLSet Function	–	
	TMOut Statement	–	
	TMove Statement	–	
	Tmr Function	–	
	TmReset Statement	–	
	Toff Statement	–	
	Ton Statement	–	
	Tool Statement	–	
	Tool Function	–	
	Trap Statement	–	
	Trim$ Function	–	
	Tw Function	–	
U	UBound Function	–	
	UCase$ Function	–	
V	Val Function	–	
W	Wait Statement	–	
	WaitNet Statement	–	
	WaitPos Statement	–	
	WaitSig Statement	–	
	Weight Statement	–	
	Weight Function	–	
	Where Statement	–	
	Wrist Statement	–	
	Wrist Function	–	
	Write Statement	–	
	WriteBin Statement	–	
X	Xor Operator	–	
	Xqt Statement	–	
	XY Function	–	
	XYLim Statement	–	
	XYLim Function	–	
	XYLimClr Statement	–	
	XYLimDef Statement	–	
	XYLimDef Function	–	

B-2: Precaution of EPSON RC+ 5.0 Compatibility

Overview

This section contains information for customers using EPSON RC+ 7.0 with RC700 series and RC90 series Controllers that have already used EPSON RC+ 5.0 with RC180.
EPSON RC+ 7.0 and EPSON RC+ 5.0 differ in such as hardware, adaptable manipulators, number of joint allowance, and software execution environment. Please read this section and understand the contents for the safety use of the Robot system.
EPSON RC+ 7.0 is improved software that has compatibility with products before EPSON RC+ 7.0 and designed to innovate advanced software technologies. However, some parts do not have compatibility with EPSON RC+ 5.0 or have been deleted to specialize in the robot controller and for ease of use.
The following compatibility is indicated based on EPSON RC+ 5.0 compared to EPSON RC+ 7.0.

General Differences

General differences of EPSON RC+ 5.0 and EPSON RC+ 7.0 are as follows.

Item	EPSON RC+ 7.0	EPSON RC+ 5.0
Number of task	Up to 32 tasks (Background task : Up to 16 tasks)	Up to 16 tasks
Type of task	Able to specify NoPouse task Able to specify NoEmgAbort task Able to specify Background task	Able to specify NoPouse task Able to specify NoEmgAbort task
Special TRAP such as TRAP ERROR	Supported	Not supported
Task starts by TRAP number	Dedicated task number	Dedicated task number
Multi Manipulator	Supported	Not supported
Robot number	1 to 16	1
Number of significant figure for Real type	6 digits	6 digits
Number of significant figure for Double type	14 digits	14 digits
Array elements number	Other than string variable Local variable 2,000 Global variable 100,000 Module variable 100,000 Global Preserve variable 4,000	Other than string variable Local variable 1,000 Global variable 0,000 Module variable 0,000 Global Preserve variable 1,000
	String variable Local variable 200 Global variable 10,000 Module variable 10,000 Global Preserve variable 400	String variable Local variable 100 Global variable 1,000 Module variable 1,000 Global Preserve variable 100
Device number	21:PC 22:REMOTE 24:TP 20:TP3	21:PC 22:REMOTE 23:OP 24:TP
Control device	Remote I/O PC REMOTE COM REMOTE Ethernet TP3	Remote I/O PC OP1 REMOTE Ethernet

Item	EPSON RC+ 7.0	EPSON RC+ 5.0
Timer number range	0 to 63	0 to 15
Program capacity	8 MB	4 MB
Signal No range for SyncLock, SyncUnlock	0 to 63	0 to 15
Signal No range for WaitSig, Signal	0 to 63	0 to 5
Memory I/O port	1024	256
I/O port number	Common with EPSON RC+ 5.0	
Port No of Ethernet	201 to 216	201 to 208
Remote I/O assignment	Assigned as default	Assigned as default
Port No of RS-232C communication	1 to 8, 1001 to 1008	1 to 8
OpenCom execution of RS-232C communication port	Mandatory	Mandatory
Input/output to files	Supported	Not supported
File number	30 to 63	Not supported
Access number for the database	501 to 508	Not supported
Vision Guide	Network camera type Frame grubber type	Network camera type
Conveyor tracking	Supported	Not supported
PG robot	Supported	Not supported
OCR	Supported	Not supported
Security	Supported	Not supported
VBGuide 5.0 (RC+ API 7.0)	Supported	VBGuide Lite is supported
Fieldbus I/O	Use normal I/O commands	Use normal I/O commands
Fieldbus master	Response is not guaranteed	Not supported
Fieldbus slave	Response is guaranteed	Response is guaranteed
GUI Builder	Supported	Not supported
Error number	Common with EPSON RC+ 5.0	

Compatibility List of Commands

+	Function expansion / function changes have been made with upper compatibility.
–	No changes.
!	Pay attention. Function changes or syntax changes have been made.
!!	Pay attention. Significant changes have been made.
×	Deleted.

Command		Compatibility	Note
A	Abs Function	–	
	Accel Statement	–	
	Accel Function	–	
	AccelMax Statement	–	
	AccelR Statement	–	
	AccelR Function	–	
	AccelS Statement	–	
	AccelS Function	–	
	Acos Function	–	
	AglToPls Function	–	
	Agl Function	–	
	AlignECP Function	–	
	Align Function	–	
	And Statement	–	
	Arc Statement	–	
	Arc3 Statement	–	
	Arch Statement	–	
	Arch Function	–	
	Arm Statement	–	
	Arm Function	–	
	ArmClr Statement	–	
	ArmDef Function	–	
	ArmSet Statement	–	
	ArmSet Function	–	
	Asc Function	–	
	Asin Function	–	
	Atan Function	–	
	Atan2 Function	–	
	ATCLR Statement	–	
	ATRQ Statement	–	
	ATRQ Function	–	
B	Base Statement	–	
	Base Function	–	
	BClr Function	–	
	BGo Statement	+	Added PerformMode parameter.
	BMove Statement	–	
	Boolean Statement	–	
	Box Statement	+	Added the robot number designation
	Box Function	+	Added the robot number designation
	BoxClr Function	+	Added the robot number designation
	BoxDef Function	+	Added the robot number designation
	Brake Statement	–	

Command		Compatibility	Note
	Brake Function	–	
	BSet Function	–	
	BTst Function	–	
	Byte Statement	–	
C	Call Statement	+	DLL function Call is supported
	ChkCom Function	–	
	ChkNet Function	–	
	Chr$ Function	–	
	ClearPoints Statement	–	
	CloseCom Statement	–	
	CloseNet Statement	–	
	Cls Statement	–	
	Cos Function	–	
	CP Statement	–	
	CP Function	–	
	CTReset Statement	–	
	Ctr Function	–	
	CtrlDev Function	!	Changed device ID
	CtrlInfo Function	–	Changed the obtaining contents
	CurPos Function	–	
	Curve Statement	–	
	CVMove Statement	–	
	CX to CW Statement	+	Added CR, CS, CT
	CX to CW Function	+	Added CR, CS, CT
D	Date Statement	!	Only displays
	Date$ Function	–	
	DegToRad Function	–	
	DispDev Statement	–	
	DispDev Function	–	
	Dist Function	–	
	Do...Loop Statement	–	
	Double Statement	–	
E	ECP Statement	–	
	ECP Function	–	
	ECPClr Statement	–	
	EcpDef Function	–	
	ECPSet Statement	–	
	ECPSet Function	–	
	ElapsedTime Function	–	
	Elbow Statement	–	
	Elbow Function	–	
	Era Function	–	
	Erase Statement	×	
	EResume Statement	–	
	Erf$ Function	–	
	Erl Function	–	
	Err Function	–	
	ErrMsg$ Function	–	
	Error Statement	–	

Command		Compatibility	Note
	ErrorOn Function	−	
	Ert Function	−	
	EStopOn Function	−	
	Exit Statement	−	
F	Find Statement	−	
	FindPos Function	−	
	Fine Statement	−	
	Fine Function	−	
	Fix Function	−	
	FmtStr$ Statement	−	
	For...Next	−	
	Function...Fend	−	
G	Global Statement	−	
	Go Statement	+	Added PerformMode parameter.
	Gosub...Return	−	
	Goto Statement	−	
H	Halt Statement	−	
	Hand Statement	−	
	Hand Function	−	
	Here Statement	−	
	Here Function	−	
	Hex$ Function	−	
	Home Statement	−	
	HomeClr Statement	−	
	HomeDef Function	−	
	HomeSet Statement	−	
	HomeSet Function	−	
	HOrdr Statement	−	
	HOrdr Function	−	
	Hour Statement	−	
	Hour Function	−	
I	If...EndIf	−	
	In Function	−	
	InBCD Function	−	
	Inertia Statement	−	
	Inertia Function	−	
	InPos Function	−	
	Input Statement	−	
	Input# Statement	+	Added the device number
	InsideBox Function	!	Added the designation of robot number and All Cannot use with Wait statement
	InsidePlane Function	!	Added the designation of robot number and All Cannot use with Wait statement
	InStr Function	−	
	Int Function	−	
	Integer Statement	−	
	InW Function	−	

Command		Compatibility	Note
	IOLabel$ Function	–	
	IONumber Function	–	
	IONumber Function	–	
J	J1Flag Statement	–	
	J1Flag Function		
	J2Flag Statement		
	J2Flag Function		
	J4Flag Statement		
	J4Flag Function	–	
	J6Flag Statement	–	
	J6Flag Function	–	
	JA Function	–	
	Joint	–	
	JRange Statement	–	
	JRange Function	–	
	JS Function	–	
	JT Function	–	
	JTran Statement	–	
	Jump Statement	+	Added PerformMode parameter.
	Jump3 Statement	+	
	Jump3CP Statement	+	
L	LCase$ Function	–	
	Left$ Function	–	
	Len Function	–	
	LimZ Statement	–	
	LimZ Function	–	
	Line Input Statement	–	
	Line Input# Statement	+	Added the device number
	LJM Function	–	
	LoadPoints	–	
	Local Statement	–	
	Local Function	–	
	LocalClr Statement	–	
	LocalDef Function	–	
	Lof Function	–	
	Long Statement	–	
	LSet$ Function	–	
	LShift Function	–	
	LTrim$ Function	–	
M	Mask Operator	–	
	MemIn Function	–	
	MemInW Function	–	
	MemOff Statement	–	
	MemOn Statement	–	
	MemOut Statement	–	
	MemOutW Statement	–	
	MemSw Function	–	
	Mid$ Function	–	
	Mod Operator	–	

Command		Compatibility	Note
	Motor Statement	–	
	Motor Function	–	
	Move Statement	–	
	MyTask Function	–	
N	Not Operator	–	
O	Off Statement	–	
	OLAccel Statement	–	
	OLAccel Function	–	
	OLRate Statement	–	
	OLRate Function	–	
	On Statement	–	
	OnErr	–	
	OpBCD Statement	–	
	OpenCom Statement	–	
	OpenNet Statement	–	
	Oport Function	–	
	Or Operator	–	
	Out Statement	–	
	Out Function	–	
	OutW Statement	–	
	OutW Function	–	
P	PAgl Function	–	
	Pallet Statement	–	Added the coordinate designation
	Pallet Function	–	
	ParsStr Statement	–	
	ParsStr Function	–	
	Pass Statement	+	
	Pause Statement	–	
	PauseOn Function	–	
	PDef Function	–	
	PDel	–	
	PLabel$ Function	–	
	PLabel Statement	–	
	Plane Statement	+	Added the robot number designation
	Plane Function	+	Added the robot number designation
	PlaneClr Statement	+	Added the robot number designation
	PlaneDef Function	+	Added the robot number designation
	PList Statement	!	Changed the display type
	PLocal Statement	–	
	PLocal Function	–	
	Pls Function	–	
	PNumber Function	–	
	PosFound Function	–	
	Power Statement	–	
	Power Function	–	
	PPls Function	–	
	Print Statement	–	
	Print# Statement	+	Changed the device number
	PTCLR Statement	–	

Command		Compatibility	Note
	PTPBoost Statement	–	
	PTPBoost Function	–	
	PTPBoostOK Function	–	
	PTPTime Function	–	
	PTran Statement	–	
	PTRQ Statement	–	
	PTRQ Function	–	
	Pulse Statement	–	
	Pulse Function	–	
Q	QP Statement	–	
	Quit Statement	–	
R	RadToDeg Function	–	
	Randmize Statement	–	
	Range Statement	–	
	Read Statement	–	
	ReadBin Statement	–	
	Real Statement	–	
	RealPls Function	–	
	RealPos Function	–	
	RealTorque Statement	–	
	Redim Statement	–	
	Reset Statement	–	
	ResetElapsedTime Statement	—	
	Resume Statement	–	
	Return Statement	–	
	RobotInfo Function	+	Added the information
	RobotInfo$ Function	+	Added the display of default point file name
	RobotModel$ Function	–	
	RobotName$ Function	–	
	RobotSerial$ Function	–	
	RobotType Function	–	
	RSet$ Function	–	
	RShift Function	–	
	RTrim$ Function	–	
S	SafetyOn Function	–	
	SavePoints Statement	–	
	Select...Send Statement	–	
	Sense Statement	–	
	SetCom Statement	–	
	SetInW Statement	–	
	SetIn Statement	–	
	SetNet Statement	–	
	SetSw Statement	–	
	SFree Statement	–	
	SFree Function	–	
	Sgn Function	–	
	Signal Statement	–	
	Sin Function	–	

Command		Compatibility	Note
	SLock Statement	–	
	SoftCP Statement	–	
	SoftCP Function	–	
	Space$ Function	–	
	Speed Statement	–	
	Speed Function	–	
	SpeedR Statement	–	
	SpeedR Function	–	
	SpeedS Statement	–	
	SpeedS Function	–	
	SPELCom_Event Statement	–	
	Sqr Function	–	
	Stat Function	+	Added the information
	Str$ Function	–	
	String Statement	–	
	Sw Function	–	
	SyncLock Statement	!	Error occurs by executing SyncLock repeatedly
	SyncUnlock Statement	–	
	SysConfig Statement	+	Added the information
	SysErr Function	+	Added the function to retrieve the warnings
T	Tab$ Function	–	
	Tan Function	–	
	TargetOK Function	–	
	TaskDone Function	–	
	TaskInfo Function	–	
	TaskInfo$ Function	–	
	TaskState Statement	+	Added the display of background task
	TaskState Function	–	
	TaskWait Statement	–	
	TC Statement	–	
	TCLim Statement	–	
	TCLim Function	–	
	TCSpeed Statement	–	
	TCSpeed Function	–	
	TGo Statement	+	Added PerformMode parameter.
	TillOn Function	–	
	Time Command	!	Only displays
	Time Function	–	
	Time$ Function	–	
	TLClr Statement	–	
	TlDef Function	–	
	TLSet Statement	–	
	TLSet Function	–	
	TMOut Statement	–	
	TMove Statement	–	
	Tmr Function	–	
	TmReset Statement	–	

Command		Compatibility	Note
	Toff Statement	–	
	Ton Statement	–	
	Tool Statement	–	
	Tool Function	–	
	Trap Statement	!	Added the Trap that interrupts the controller status
	Trim$ Function	–	
	Tw Function	–	
U	UBound Function	–	
	UCase$ Function	–	
V	Val Function	–	
W	Wait Statement	!	Added the global variables and others as the wait condition
	WaitNet Statement	–	
	WaitPos Statement	–	
	WaitSig Statement	–	
	Weight Statement	+	Added the designation of S, T
	Weight Function	+	Added the designation of S, T
	Where Statement	–	
	Wrist Statement	–	
	Wrist Function	–	
	Write Statement	–	
	WriteBin Statement	–	
X	Xor Operator	–	
	Xqt Statement	–	
	XY Function	–	
	XYLim Statement	–	
	XYLim Function	–	
	XYLimClr Statement	–	
	XYLimDef Statement	–	
	XYLimDef Function	–	

Commands from EPSON RC+ Ver.4.* (Not supported in EPSON RC+ 5.0)

Aopen Statement
BOpen Statement
Calib Statement
CalPls Statement
ChDir Statement
ChDrive Statement
Close Statement
Cnv_AbortTrack Statement
Cnv_Downstream Statement
Cnv_Fine Statement
Cnv_Fine Function
Cnv_Name$ Function
Cnv_Number Function
Cnv_Point Function
Cnv_PosErr Function
Cnv_Pulse Function
Cnv_QueAdd Statement
Cnv_QueGet Function
Cnv_QueLen Function
Cnv_QueList Statement
Cnv_QueMove Statement
Cnv_QueReject Statement
Cnv_QueReject Function
Cnv_QueRemove Statement
Cnv_QueUserData Statement
Cnv_QueUserData Function
Cnv_RobotConveyor Function
Cnv_Speed Function
Cnv_Trigger Statement
Cnv_Upstream Function
Cont Statement
Copy Statement
CurDir$ Function
CurDrive$ Function
Declare Statement
Del Statement
Eof Function
Eval Function
FbusIO_GetBusStatus Function
FbusIO_GetDeviceStatus Function
FbusIO_SendMsg Statement
FileDateTime$ Function
FileExists Function
FileLen Function
FolderExists Function
FreeFile Function
GetCurrentUser$ Statement
Hofs Statement
Hofs Function
ImportPoints Statement
InputBox Statement
LogIn Function
MCalComplete Function
MCal Statement
MCordr Statement
MCordr Function
MKDir Statement
MsgBox Statement
Recover Function
Rename Statement
RenDir Statement
Restart Statement
RmDir Statement
Robot Statement
Robot Function
ROpen Statement
RunDialog Statement
Seek Statement
Shutdown Statement
UOpen Statement
WOpen Statement

B-3: Precaution of EPSON RC+ Ver.4.* Compatibility

Overview

This section contains information for customers using EPSON RC+ 7.0 with RC700 series Controller that have already used EPSON RC+ Ver.4.* with RC520 or RC420.
EPSON RC+ 7.0 and EPSON RC+ Ver.4.* differs in such as hardware, adaptable manipulators, number of joint allowance, and software execution environment. Please read this section and understand the contents for the safety use of the Robot system.
EPSON RC+ 7.0 is improved software that has compatibility with products before EPSON RC+ 7.0 and designed to innovate advanced software technologies. However, some parts do not have compatibility with EPSON RC+ Ver.4.* or have been deleted to specialize in the robot controller and for ease of use.
The following compatibility is indicated based on EPSON RC+ Ver.4.* compared to EPSON RC+ 7.0.

General Differences

General differences of EPSON RC+ Ver.4.* and EPSON RC+ 7.0 are as follows.

Item	EPSON RC+ 7.0	EPSON RC+ Ver.4.*
Number of task	Up to 32 tasks (Background task : Up to 16 tasks)	Up to 32 tasks
Type of task	Able to specify NoPouse task Able to specify NoEmgAbort task Able to specify Background task	Able to specify NoPouse task
Special TRAP such as TRAP ERROR	Supported	Supported
Task starts by TRAP number	Dedicated task number	Task number only using 1 to 32
Multi manipulator	Supported	Supported
Robot number	1 to 16	1 to 16
Number of significant figure for Real type	6 digits	7 digits
Number of significant figure for Double type	14 digits	15 digits
Array elements number	Other than string variable Local variable 2000 Global variable 1,000,000 Module variable 1,000,000 Global Preserve variable 4,000	As far as the memory remains
	String variable Local variable 200 Global variable 10,000 Module variable 10,000 Global Preserve variable 400	
Line number	Not supported	Supported
Device number	21:PC 22:REMOTE 24:TP 20:TP3	1:Controller 2:REMOTE 3:OP
Control device	Remote I/O PC REMOTE COM REMOTE Ethernet	Remote I/O PC OP500RC

Item	EPSON RC+ 7.0	EPSON RC+ Ver.4.*
	TP3	
Timer number range	0 to 63	0 to 63
Program capacity	8 MB	4 MB
Signal No range for SyncLock, SyncUnlock	0 to 63	1 to 32
Signal No range for WaitSig, Signal	0 to 63	0 to 127
Memory I/O port	1024	512
I/O port number	Different with EPSON RC+4.0	
Port No of Ethernet	201 to 216	128 to 147
Remote I/O assignment	Assigned as default	Default: --
Port No of RS-232C communication	1 to 8, 1001, 1008	1 to 16
OpenCom execution of RS-232C communication port	Mandatory	Optional
Input/output to files	Supported	Supported
File number for the file access	30 to 63	30 to 63
Access number for the database	501 to 508	Not supported
Vision Guide	Network camera type Frame grubber type	Frame grubber type
Conveyor tracking	Supported	Supported
PG robot	Supported	Supported
OCR	Supported	Supported
Security	Supported	Supported
VBGuide (RC+ API 7.0)	Supported	Supported
Fieldbus I/O	Use normal I/O commands	Use special commands
Fieldbus master	Response is not guaranteed	Response is not guaranteed
Fieldbus slave	Response is guaranteed	Response is not guaranteed
GUI Builder	Supported	Not supported
Group in the project	Not supported	Supported
Error number	Different with EPSON RC+ Ver.4.*	

Compatibility List of Commands

+	Function expansion / function changes have been made with upper compatibility.
–	No changes.
!	Pay attention. Function changes or syntax changes have been made.
!!	Pay attention. Significant changes have been made.
×	Deleted.

Command		Compatibility	Note
A	Abs Function	–	
	Accel Statement	+	Able to specify more than 100 for some robots
	Accel Function	–	
	AccelR Statement	–	
	AccelR Function	–	
	AccelS Statement	–	
	AccelS Function	–	
	Acos Function	+	Argument range check has been added
	Agl Function	–	
	AglToPls Function	–	
	And Operator	–	
	AOpen Statement	×	
	Arc Statement	–	
	Arc3 Statement	–	
	Arch Statement	–	
	Arch Function	–	
	Arm Statement	–	
	Arm Function	–	
	ArmClr Statement	–	
	ArmSet Statement	–	
	ArmSet Function	–	
	Asc Function	–	
	Asin Function	+	Argument range check has been added
	Atan Function	–	
	Atan2 Function	–	
	ATCLR Statement	–	
	ATRQ Statement	–	
	ATRQ Function	–	
B	Base Statement	–	
	BClr Function	+	Argument range check has been added
	Beep Statement	×	
	BGo Statement	+	Added PerformMode parameter.
	BMove Statement	–	
	Boolean Statement	–	
	BOpen Statement	–	
	Brake Statement	–	
	BSet Function	+	Argument range check has been added
	BTst Function	+	Argument range check has been added
	Byte Statement	–	
C	Calib Statement	–	

Command		Compatibility	Note
	Call Statement	–	
	CalPls Statement	–	
	CalPls Function	–	
	Chain Statement	×	
	ChDir Statement	–	
	ChDrive Statement	–	
	ChkCom Function	–	
	ChkNet Function	–	
	Chr$ Function	–	
	Clear Statement	!	Renamed to ClearPoints
	Close Statement	–	
	CloseCom Statement	–	
	CloseNet Statement	+	Able to specify All
	ClrScr Statement	!	Renamed to Cls Device ID can be specified for arguments
	Cnv_**	–	
	Cont Statement	!	Able to execute by the setting
	Copy Statement	–	
	Cos Function	–	
	CP Statement	–	
	CP Function	–	
	Ctr Function	–	
	CTReset Statement	–	
	CtrlDev Statement	×	
	CtrlDev Function	!	Changed device ID
	CtrlInfo Function	!!	Changed the obtaining contents
	CurDir$ Function	–	
	CurDrive$ Function	–	
	CurPos Function	–	
	Curve Statement	–	
	CVMove Statement	–	
	CX to CW Statement	+	Added CR, CS, CT
	CX to CW Function	+	Added CR, CS, CT
D	Date Statement	!	Only displays
	Date$ Function	–	
	Declare Statement	!	The processing is slow
	DegToRad Function	–	
	Del Statement	–	
	Dir Statement	–	
	Dist Function	–	
	Do...Loop Statement	–	
	Double Statement	!	Significant figure is 14 digits
E	EClr Statement	×	
	ECP Statement	–	
	ECP Function	–	
	ECPClr Statement	–	
	ECPSet Statement	–	
	ECPSet Function	–	

Command		Compatibility	Note
	Elbow Statement	–	
	Elbow Function	–	
	ENetIO_****	×	
	Eof Function	–	
	EPrint Statement	×	
	Era Function	–	
	Erase Statement	×	
	EResume Statement	–	
	Erf$ Function	+	Able to omit the task number
	Erl Function	+	Able to omit the task number
	Err Function	–	
	ErrHist Statement	×	
	ErrMsg$ Function	!	Argument has language ID
	Error Statement	+	Able to specify task number for arguments
	Ert Function	–	
	EStopOn Function	+	Able to specify Wait
	Eval Function	!	Differences in the error output
	Exit Statement	–	
F	FbusIO_****	×	Normal I/O command available
	FileDateTime$ Function	–	
	FileExists Function	–	
	FileLen Function	–	
	Find Statement	–	
	FindPos Function	–	
	Fine Statement	–	
	Fine Function	–	
	Fix Function	–	
	FmtStr$ Statement	!!	Function is limited significantly
	FoldrExist Function	–	
	For...Next	–	
	FreeFile Function	–	
	Function...Fend	–	
G	GetCurrentUser$ Functio n	–	
	Global Statement	–	
	Go Statement	+	Added PerformMode parameter.
	Gosub...Return	–	
	Goto Statement	–	
H	Halt Statement	–	
	Hand Statement	–	
	Hand Function	–	
	Here Statement	–	
	Here Function	–	
	Hex$ Function	–	
	Hofs Statement	–	
	Hofs Function	–	
	Home Statement	–	

Command		Compatibility	Note
	HomeSet Statement	–	
	HomeSet Function	–	
	HOrdr Statement	–	
	HOrdr Function	–	
	Hour Statement	–	
	Hour Function	–	
	HTest Statement	×	
	HTest Function	×	
I	If...EndIf	–	
	ImportPoints Statement	!	Extension ".pnt" has changed to ".pts"
	In Function	–	
	In($n) Statement	×	Replaced to MemIn
	InBCD Function	–	
	Inertia Statement	–	
	Inertia Function	–	
	InPos Function	–	
	Input Statement	–	
	Input# Statement	+	Input is available from devices
	InputBox Statement	–	
	InStr Function	–	
	Int Function	–	
	Integer Statement	–	
	InW Function	–	
	InW($n) Statement	×	Replaced to MemInW
	IONumber Function	–	
J	J4Flag Statement	–	
	J4Flag Function	–	
	J6Flag Statement	–	
	J6Flag Function	–	
	JA Function	–	
	JRange Statement	–	
	JRange Function	–	
	JS Function	!	Returns True/False
	JT Function	–	
	JTran Statement	–	
	Jump Statement	+	Added PerformMode parameter.
	Jump3 Statement	–	
	Jump3CP Statement	–	
K	Kill Statement	×	Replaced with Del
L	LCase$ Function	–	
	Left$ Function	–	
	Len Function	–	
	LimZ Statement	–	
	LimZ Function	–	
	Line Input Statement	–	
	Line Input# Statement	+	Input is available from devices
	LoadPoints	!	Extension ".pnt" has changed to ".pts"

Command		Compatibility	Note
	Local Statement	!	Local number “0” is an error
	Local Function	!	Local number “0” is an error
	LocalClr Statement	–	
	Lof Function	–	
	LogIn Statement	!	Changed from a statement to a function
	Long Statement	–	
	LPrint Statement	×	
	LSet$ Function	–	
	LShift Function	+	Argument range check has been added
	LTrim$ Function	–	
M	Mask Operator	–	
	MCal Statement	–	
	MCalComplete Function	–	
	MCofs Statement	×	
	MCofs Function	×	
	MCordr Statement	–	
	MCordr Function	–	
	Mcorg Statement	×	
	MemIn Function	–	
	MemInW Function	–	
	MemOff Statement	–	
	MemOn Statement	–	
	MemOut Statement	–	
	MemOutW Statement	–	
	MemSw Function	–	
	Mid$ Function	–	
	MKDir Statement	–	
	Mod Operator	–	
	Motor Statement	–	
	Motor Function	–	
	Move Statement	–	
	MsgBox Statement	–	
	MyTask Function	–	
N	Not Operator	–	
O	Off Statement	–	
	Off$ Statement	×	Replaced to MemOff
	OLRate Statement	–	
	OLRate Function	–	
	On Statement	–	
	On$ Statement	×	Replaced to MemOn
	OnErr	–	
	OP_*	×	
	OpBCD Statement	–	
	OpenCom Statement	!	OpenCom is mandatory
	OpenNet Statement	–	
	Oport Function	–	
	Or Operator	–	

Command		Compatibility	Note
	Out Statement	–	
	Out Function	–	
	Out$ Statement	×	Replaced to MemOut
	OutW Statement	–	
	OutW Function	–	
	OutW$ Statement	×	Replaced to MemOutW
P	PAgl Function	–	
	Pallet Statement	–	
	Pallet Function	–	
	ParsStr Statement	–	
	ParsStr Function	–	
	Pass Statement	+	Able to specify continuous point
	Pause Statement	–	
	PauseOn Function	–	
	PDef Function	–	
	PDel	+	Argument check has been added
	PLabel$ Function	–	
	PLabel Statement	–	
	PList	!!	Changed the display type Argument check has been added Function of Plist* has been deleted
	PLocal Statement	–	
	PLocal Function	–	
	Pls Function	–	
	PNumber Function	–	
	Point Assignment	–	
	Point Expression	–	
	POrient Statement	×	
	POrient Function	×	
	PosFound Function	!	Returns True/False
	Power Statement	–	
	Power Function	–	
	PPls Function	–	
	Print Statement	!	Outputs all flags at point output Sets the output digit number of Double type and Real type to significant figure
	Print# Statement	!	Same as Print Statement Enables Print to each devices
	PTCLR Statement	–	
	PTPBoost Statement	–	
	PTPBoost Function	–	
	PTPBoostOK Function	!	Returns True/False
	PTPTime Function	–	
	PTran Statement	–	
	PTRQ Statement	–	
	PTRQ Function	–	
	Pulse Statement	–	
	Pulse Function	–	

Command		Compatibility	Note
Q	QP Statement	–	
	Quit Statement	–	
R	RadToDeg Function	–	
	Randmize Statement	+	Seed value can be specified
	Range Statement	–	
	Read Statement	–	
	ReadBin Statement	+	Able to read multiple bytes to array variable
	Real Statement	!	6 digit significant figure
	Recover Statement	!	Able to execute by the setting
	Redim Statement	!	Element number is limited Array called by reference cannot be executed
	Rename Statement	–	
	RenDir Statement	–	
	Reset Statement	–	
	Resume Statement	–	
	Restart Statement	–	
	Reset Statement	+	Added Reset Error
	Return Statement	–	
	Right$ Function	–	
	RmDir Statement	–	
	Rnd Function	–	
	Robot Statement	+	Added the RS series
	Robot Function	–	
	RobotModel$ Function	–	
	RobotType Function	–	
	ROpen Statement	×	
	RSet$ Function	–	
	RShift Function	+	Argument check has been added
	RTrim$ Function	–	
	RunDialog Statement	–	
S	SafetyOn Function	+	Able to specify Wait
	SavePoints Statement	!	Extension (.pnt) has changed to (.pts)
	Seek Statement	–	
	Select...Send	–	
	Sense	–	
	SetCom Statement	!	Cannot specify “56000” for the transfer rate Port with OpenCom cannot be executed
	SetNet Statement	–	
	SFree Statement	–	
	SFree Function	–	
	Sgn Function	–	
	Shutdown Statement	–	
	Signal Statement	–	
	Sin Function	–	
	SLock Statement	–	
	Space$ Function	–	

Command		Compatibility	Note
	Speed Statement	–	
	Speed Function	+	Argument optional
	SpeedR Statement	–	
	SpeedR Function	–	
	SpeedS Statement	–	
	SpeedS Function	–	
	SPELCom_Event Statement	–	
	SPELCom_Return Statement	×	
	Sqr Function	–	
	Stat Function	!	Some information cannot be retrieved
	Str$ Function	–	
	String Statement	–	
	Sw Function	–	
	Sw($) Function	×	Replaced to MemSw
	SyncLock Statement	!	Error occurs by executing SyncLock repeatedly Lock is released when the task is completed
	SyncUnlock Statement	–	
T	Tab$ Function	–	
	Tan Function	–	
	TargetOK Function	!	Returns True/False
	TaskDone Function	–	
	TaskState Function	!	6 specified tasks do not return while Wait statement execution
	TaskWait Statement	–	
	TGo Statement	+	Added PerformMode parameter.
	TillOn Function	–	
	Time Command	!	Only displays
	Time Function	–	
	Time$ Function	–	
	TLClr Statement	–	
	TLSet Statement	–	
	TLSet Function	–	
	TMOut Statement	–	
	TMove Statement	–	
	Tmr Function	–	
	TmReset Statement	–	
	Tool Statement	–	
	Tool Function	–	
	Trap Statement	!!	Compatibility with Trap Goto Trap Gosub abolished and replaced to Trap Call Trap Call is renamed to Trap Xqt Added Trap Finish
	Trim$ Function	–	
	Tw Function	!	Returns True/False
	Type Statement	–	

Command		Compatibility	Note
U	UBound Function	–	
	UCase$ Function	–	
	UOpen Statement	–	
V	Val Function	–	
	Ver Statement	×	Replaced to SysConfig
	Verinit Statement	×	
W	Wait Statement	+	Added the global variables and others as the wait condition
	WaitNet Statement	–	
	WaitPos Statement	–	
	WaitSig Statement	–	
	Weight Statement	+	Added the designation of S, T
	Weight Function	+	Added the designation of S, T
	Where Statement	!	Coordinate value always displays 6-axis
	While..Wend	×	Replaced to Do...Loop
	WOpen Statement	–	
	Wrist Statement	–	
	Wrist Function	–	
	Write Statement	–	
	WriteBin Statement	+	Multiple bytes can be listed from the array variable
X	Xor Operator	–	
	Xqt Statement	+	Able to specify NoEmgAbort
	XY Function	–	
	XYLim Statement	–	
	XYLim Function	–	
Z	ZeroFlg Function	×	

Appendix C: Commands of EPSON RC+7.0

C-1: List of Commands Added EPSON RC+4.0 or Later

AbortMotion Statement
AccelMax Function
AglToPls Function
AIO_Out Statement
AIO_Out Function
AIO_OutW Statement
AIO_OutW Function
AIO_Set Statement
AIO_Set Function
AIO_TrackingSet Statement
AIO_TrackingStart Statement
AIO_TrackingEnd Statement
AIO_TrackingOn Function
AIO_In Function
AIO_InW Function
Align Function
AlignECP Function
AreaCorrection Function
AreaCorrectionClr Statement
AreaCorrectionDef Function
AreaCorrectionInv Function
AreaCorrectionOffset Function
AreaCorrectionSet Statement
ArmCalib Statement
ArmCalib Function
ArmCalibSet Statement
ArmCalibSet Function
ArmCalibClr Statement
ArmCalibDef Function
ArmDef Function
ATCLR Statement
AtHome Function
ATRQ Statement
ATRQ Function
AutoLJM Statement
AutoLJM Function
AvoidSingularity Statement
AvoidSingularity Function

BClr Function
BClr64 Function
Box Statement
Box Function
BoxClr Function
BoxDef Function
Brake Function
BSet Function
BSet64 Function
BTst Function
BTst64 Function

ChDisk Statement
ChkCom Function
ChkNet Function
CloseCom Statement
CloseDB Statement
CloseNet Statement
Cls Statement
CP Statement
CP Function
CP_Offset Statement
CP_Offset Function
CR Statement
CR Function CS Statement
CS Function
CT Statement
CT Function
CtrlDev Function
Curve Statement
CVMove Statement
Cnv_Accel Statement
Cnv_Accel Function
Cnv_AccelLim Statement
Cnv_AccelLim Function
Cnv_Adjust Statement
Cnv_AdjustClear Statement
Cnv_AdjustGet Statement
Cnv_AdjustSet Statement
Cnv_DownStream Statement
Cnv_Mode Statement
Cnv_Mode Function
Cnv_OffsetAngle Statement
Cnv_OffsetAngle Function
Cnv_PosErrOffset Statement
Cnv_Upstream Statement
CollisionDetect Statement
CollisionDetect Function

DegToRad Function
DeleteDB Statement
DiffPoint Function
DispDev Statement
DispDev Function
Dist Function

EcpDef Function
ElapsedTime Function
EResume Statement
Errb Function
ErrorOn Function
Error Statement
EStopOn Function
Exit Statement
ExportPoints Statement

FindPos Function
Find Statement
FineDist Statement
FineDist Function
FineStatus Function
Fix Function
Flush Statement
Fmtstr Statement
FunctionHere Statement

GetRobotInsideBox Function
GetRobotInsidePlane Statement

Hand_On Statement
Hand_On Function、
Hand_Off Statement
Hand_Off Function
Hand_TW Function
Hand_Def Function
Hand_Type Function
Hand_Label$ Function
Hand_Number Function
HealthCalcPeriod Statement
HealthCalcPeriod Function
HealthCtrlAlarmOn Function
HealthCtrlInfo Statement
HealthCtrlInfo Function
HealthCtrlRateOffset Statement
HealthCtrlReset Statement
HealthCtrlWarningEnable Statement
HealthCtrlWarningEnable Function
HealthRateCtrlInfo Function
HealthRateRBInfo Function
HealthRBAlarmOn Function
HealthRBAnalysis Statement
HealthRBAnalysis Function
HealthRBDistance Statement
HealthRBDistance Function
HealthRBInfo Statement
HealthRBInfo Function
HealthRBRateOffset Statement
HealthRBReset Statement
HealthRBSpeed Statement
HealthRBSpeed Function
HealthRBStart Statement
HealthRBStop Statement
HealthRBTRQ Statement
HealthRBTRQ Function
HealthRBWarningEnable Statement
HealthRBWarningEnable Function

Here Function
Hex$ Function
HofsJointAccuracy Statement
HomeClr Statement
HomeDef Function

InReal Function
InsideBox Function
InsidePlane Function
InStr Function
IODef Function
IOLabel$ Function
IONumber Function

J1Angle Statement
J1Angle Function
J4Angle Statement
JA Function
Joint Statement
JointAccuracy Statement
JointAccuracy Function
JumpTLZ Statement
JTran Statement

LatchEnable Statement
LatchState Function
LatchPos Function
LimZMargin Statement
LimZMargin Function
LimitTorque Statement
LimitTorque Function
LimitTorqueLP Statement
LimitTorqueLP Function
LimitTorqueStop Statement
LimitTorqueStop Function
LimitTorqueStopLP Statement
LimitTorqueStopLP Function
LJM Function
LocalDef Function
LShift64 Function

MemInW Function
MemOutW Statement
MHour Function

OLAccel Statement
OLAccel Function
OpenCom Statement
OpenCom Function
OpenDB Statement
OpenNet Statement
OpenNet Function
OutReal
OutReal Function

P# Statement
PalletClr Statement
PauseOn Function
PDef Function
PDel Statement
PDescription Statement
PDescription Function
PerformMode Statement
PerformMode Function
PG_FastStop Statement
PG_LSpeed Statement
PG_LSpeed Function
PG_Scan Statement
PG_SlowStop Statement
PLabel Statement
PLabel$ Function
PlaneClr Statement
PlaneDef Statement
Plane Statement
Plane Function
PList Statement
PLocal Statement
PLocal Function
PNumber Function
PosFound Function
PTCLR Statement
PTPBoostOK Function
PTPTime Function
PTran Statement
PTRQ Statement
PTRQ Function

QPDECELR Statement
QPDECELR Function
QPDECELS Statement
QPDECELS Function

RadToDeg Function
Randomize Statement
ReadBin Statement
Read Statement
RealAccel Function
RealPls Function
RealPos Function
RealTorque Function
RecoverPos Function
Recover Statement
Redim Statement
Rnd Function
ResetElapsedTime Statement
Rnd Function
RobotInfo Function
RobotInfo$ Function
RobotModel$ Function
RobotName$ Function
RobotSerial$ Function
RobotType Function
ROTOK Function
RShift64 Function

SafetyOn Function
SelectDB Statement
SetCom Statement
SetInW Statement
SetIn Statement
SetNet Statement
SetSw Statement
SF_GetParam Function
SF_GetParam$ Function
SF_GetStatus Function
SF_LimitSpeedS Statement
SF_LimitSpeedS Function
SF_LimitSpeedEnable Statement
SF_LimitSpeedEnable Function
SF_PeakSpeedS Statement
SF_PeakSpeedS Function
SF_PeakSpeedSClear Statement
SF_RealSpeedS Statement
SF_RealSpeedS Function
Shutdown Function
SimGet Statement
SimSet Statement
SysErr Function

SingularityAngle Statement
SingularityAngle Function
SingularityDist Statement
SingularityDist Function
SingularitySpeed Statement
SingularitySpeed Function
SoftCP Statement
SoftCP Function
SpeedFactor Statement
SpeedFactor Function
StartMain Statement
SyncRobots Statement
SyncRobots Function

Tab$ Function
TargetOK Function
TaskDone Function
TaskInfo Function
TaskInfo$ Function
TaskState Statement
TaskState Function
TaskWait Statement
TC Statement
TCLim Statement
TCLim Function
TCSpeed Statement
TCSpeed Function
TeachOn Function
TillOn Function
TlDef Function
Toff Statement
Ton Statement

UBound Function
UpdateDB Statement

VDefArm Statement
VDefLocal Statement
VDefSetMotionRange Statement
VDefGetMotionRange Statement
VDefTool Statement
VGoCenter Statement
VSD Statement
VSD Function
VxCalib Statement
VxCalDelete Statement
VxCalLoad Statement
VxCalInfo Function
VxCalSave Statement
VxTrans Function

WaitNet Statement
WaitPos Statement
Where Statement
WindosStatus Function
WriteBin Statement
Write Statement
WorkQue_Add Statement
WorkQue_AutoRemove Statement
WorkQue_AutoRemove Function
WorkQue_Get Function
WorkQue_Len Function
WorkQue_List Statement
WorkQue_Reject Statement
WorkQue_Reject Function
WorkQue_Remove Statement
WorkQue_Sort Statement
WorkQue_Sort Function
WorkQue_UserData Statement
WorkQue_UserData Function

XYLimClr Statement
XYLimDef Statement
XY Function

C-2: List of Commands Added for Each Version of EPSON RC+ 7.0

Common with EPSON RC+ 6.0, 5.0, and 4.0

Version of EPSON RC+7.0	New Commands	
Ver.7.5.4	Cnv_AccelLim	SF_LimitSpeedEnable
	Cnv_AccelLim Function	SF_LimitSpeedEnable Function
	SF_GetParam Function	SF_PeakSpeedS
	SF_GetParam$ Function	SF_PeakSpeedS Function
	SF_GetStatus Function	SF_PeakSpeedSClear
	SF_LimitSpeedS	SF_RealSpeedS
	SF_LimitSpeedS Function	SF_RealSpeedS Function
Ver.7.5.3	AreaCorrection Function	AreaCorrectionOffset Function
	AreaCorrectionClr	AreaCorrectionSet
	AreaCorrectionDef Function	Cnv_PosErrOffset
	AreaCorrectionInv Function	
Ver.7.5.2	XYLimMode	
	XYLimMode Function	
Ver.7.5.1	ArmCalib	Hand_TW Function
	ArmCalib Function	Hand_Def Function
	ArmCalibSet	Hand_Type Function
	ArmCalibSet Function	Hand_Label$ Function
	ArmCalibClr	Hand_Number Function
	ArmCalibDef Function	HofsJointAccuracy
	JointAccuracy	Cnv_Adjust
	JointAccuracy Function	Cnv_AdjustClear
	Hand_On Statement	Cnv_AdjustGet
	Hand_On Function	Cnv_AdjustSet
	Hand_Off Statement	DiffPoint Function
	Hand_Off Function	ROTOK Function
Ver.7.4.3	AIO_TrackingSet	AIO_TrackingEnd
	AIO_TrackingStart	AIO_TrackingOn Function
Ver.7.4.1	AutoOrientationFlag	
	AutoOrientationFlag Function	
Ver.7.3.4	SimGet	
	SimSet	
Ver.7.3.3	HealthCtrlWarningEnable Statement	HealthRBWarningEnable Statement
	HealthCtrlWarningEnable Function	HealthRBWarningEnable Function
Ver.7.3.2	PDescription Statement	PDescription Function
Ver.7.3.1	AIO_Out Statement	AIO_Set Function
	AIO_Out Function	AIO_In Function
	AIO_OutW Statement	AIO_InW Function
	AIO_OutW Function	HealthCalcPeriod Statement
	AIO_Set Statement	HealthCalcPeriod Function
Ver.7.3.0	VDefArm Statement	VDefSetMotionRange Statement
	VDefGetMotionRange Statement	VDefTool Statement
	VDefLocal Statement	VGoCenter Statement
Ver.7.2.0	CP_Offset Function	HealthRBReset HealthRBSpeed Function
	CP_Offset Statement	HealthRBSpeed Statement
	HealthCtrlAlarmOn Statement	HealthRBStart Function
	HealthCtrlInfo Function	HealthRBStop Function
	HealthCtrlInfo Statement	HealthRBTRQ Function

	HealthCtrlRateOffset Function HealthCtrlReset Function HealthRateCtrlInfo Statement HealthRateRBInfo Statement HealthRBAlarmOn Statement HealthRBAnalysis Function HealthRBAnalysis Statement HealthRBDistance Function HealthRBDistance Statement HealthRBInfo Function HealthRBInfo Statement HealthRBRateOffset Function	HealthRBTRQ Statement J4Angle Function JumpTLZ Function LimitTorqueLP Function LimitTorqueLP Statement LimitTorqueStop Function LimitTorqueStop Statement LimitTorqueStopLP Function LimitTorqueStopLP Statement VSD Function VSD Statement
Ver.7.1.4	CollisionDetectStatement Function	
Ver.7.1.3	CollisionDetect Function MHourStatement Function	
Ver.7.1.2	SingularityDist Function SingularityDist Statement	ExportPoints Function
Ver.7.1.0	BClr64 Statement BSet64 Statement BTst64 Statement FineDist Function FineDist Statement FineStatus Statement Fmtstr Function IODef Statement LShift64 Statement RealAccel Statement RShift64 Statement WorkQue_Add Function WorkQue_AutoRemove Function	WorkQue_AutoRemove Statement WorkQue_Get Statement WorkQue_Len Statement WorkQue_List Function WorkQue_Reject Function WorkQue_Reject Statement WorkQue_Remove Function WorkQue_Sort Function WorkQue_Sort Statement WorkQue_UserData Function WorkQue_UserData Statement
Ver.7.0.3	PerformMode Function	PerformMode Statement

NOTE For EPSON RC+7.0 Ver.7.0.0, new commands are different from EPSON RC+ 6.0 and EPSON RC+ 5.0.

Added commands for EPSON RC+ 6.0, 5.0.

Version of EPSON RC+7.0	EPSON RC+ 6.0	EPSON RC+ 5.0
Ver.7.0.0		AbortMotion Function
	AutoLJM Function	AutoLJM Function
	AutoLJM Statement	AutoLJM Statement
	AvoidSingularity Function	AvoidSingularity Function
	AvoidSingularity Statement	AvoidSingularity Statement
		ChDisk Function
		CloseDB Function
	Cnv_Accel Function	Cnv_Accel Function
	Cnv_Accel Statement	Cnv_Accel Statement
	Cnv_Down Stream Function	Cnv_DownStream Function
	Cnv_Mode Function	Cnv_Mode Function
	Cnv_Mode Statement	Cnv_Mode Statement
	Cnv_Upstream Function	Cnv_Upstream Function
		CR Function
		CR Statement
		CS Function
		CS Statement
		CT Function
		CT Statement
	DeleteDB Function	DeleteDB Function
	ElapsedTime Statement	
	Errb Statement	Errb Statement
		Flush Function
		GetRobotInsideBox Statement
		GetRobotInsidePlane Statement
		J1Angle Function
		J1Angle Statement
	LimZMargin Function	LimZMargin Function
	LimZMargin Statement	LimZMargin Statement
	LimitTorque Function	LimitTorque Function
	LimitTorqueStatement	LimitTorque Statement
		OpenDB Function
	PalletClr Function	PalletClr Function
		PG_FastStop Function
		PG_LSpeed Function
		PG_LSpeed Statement
		PG_Scan Function
		PG_SlowStop Function
		QPDECELR Function
		QPDECELR Statement
		QPDECELS Function
		QPDECELS Statement
		RecoverPos Statement
		Recover Function
	ResetElapsedTime Function	
		SelectDB Function
		Shutdown Statement

Version of EPSON RC+7.0	EPSON RC+ 6.0	EPSON RC+ 5.0
Ver.7.0.0	SingularityAngle Function SingularityAngle Statement SingularitySpeed Function SingularitySpeed Statement SpeedFactor Function SpeedFactor Statement UpdateDB Function	SingularityAngle Function SingularityAngle Statement SingularitySpeed Function SingularitySpeed Statement SpeedFactor Function SpeedFactor Statement StartMain Function SyncRobots Function SyncRobots Statement TeachOnStatement Function UpdateDB Function WindosStatus Statement

Added commands for EPSON RC+ 4.0.

Version of EPSON RC+7.0	EPSON RC+ 4.0	
Ver.7.0.0	AbortMotion Function AccelMax Statement AglToPls Statement Align Statement AlignECP Statement ArmDef Statement ATCLR Function AtHome Statement ATRQ Function ATRQ Statement AutoLJM Function AutoLJM Statement AvoidSingularity Function AvoidSingularity Statement BClr Statement Box Function Box Statement BoxClr Statement BoxDef Statement Brake Statement BSet Statement BTst Statement ChDisk Function ChkCom Statement ChkNet Statement CloseCom Function CloseDB Function CloseNet Function Cls Function CP Function CP Statement	CR Function CR Statement CS Function CS Statement CT Function CT Statement CtrlDev Statement Curve Function CVMove Function Cnv_Accel Function Cnv_Accel Statement Cnv_DownStream Function Cnv_Mode Function Cnv_Mode Statement Cnv_OffsetAngle Function Cnv_OffsetAngle Statement Cnv_Upstream Function DegToRad Statement DeleteDB Function DispDev Function DispDev Statement Dist Statement EcpDef Statement EResume Function Errb Statement ErrorOn Statement Error Function EStopOn Statement Exit Function

Version of EPSON RC+7.0	EPSON RC+ 4.0	
Ver.7.0.0	FindPos Statement	P# Function
	Find Function	PalletClr Function
	FineStatus Statement	PauseOn Statement
	Fix Statement	PDef Statement
	Flush Function	PDel Function
		PG_FastStop Function
	GetRobotInsideBox	PG_LSpeed Function
	Statement	PG_LSpeed Statement
	GetRobotInsidePlane	PG_Scan Function
	Statement	PG_SlowStop Function
		PLabel Function
	Here Function	PLabel$ Statement
	Here Statement	PlaneClr Function
	Hex$ Statement	PlaneDef Function
	HomeClr Function	Plane Function
	HomeDef Statement	Plane Statement
		PList Function
	InReal Statement	PLocal Function
	InsideBox Statement	PLocal Statement
	InsidePlane Statement	PNumber Statement
	InStr Statement	PosFound Statement
	IOLabel$ Statement	PTCLR Function
	IONumber Statement	PTPBoostOK Statement
		PTPTime Statement
	J1Angle Function	PTran Function
	J1Angle Statement	PTRQ Function
	JA Statement	PTRQ Statement
	Joint Function	
	JTran Function	QPDECELR Function
		QPDECELR Statement
	LatchEnable Function	QPDECELS Function
	LatchState Statement	QPDECELS Statement
	LatchPos Statement	
	LimZMargin Function	RadToDeg Statement
	LimZMargin Statement	Randomize Function
	LimitTorque Function	ReadBin Function
	LimitTorque Statement	Read Function
	LJM Statement	RealPls Statement
	LocalDef Statement	RealPos Statement
		RealTorque Statement
	MemInW Statement	RecoverPos Statement
	MemOutW Function	Recover Function
	OLAccel Function	
	OLAccel Statement	
	OpenCom Function	
	OpenCom Statement	
	OpenDB Function	
	OpenNet Function	
	OpenNet Statement	
	OutReal Function	
	OutReal Statement	

Version of EPSON RC+7.0	EPSON RC+ 4.0	
Ver.7.0.0	Redim Function RobotInfo Statement RobotInfo$ Statement RobotModel$ Statement RobotName$ Statement RobotSerial$ Statement RobotType Statement SafetyOn Statement SelectDB Function SetCom Function SetInW Function SetIn Function SetNet Function SetSw Function Shutdown Statement SingularityAngle Function SingularityAngle Statement SingularitySpeed Function SingularitySpeed Statement SoftCP Function SoftCP Statement SpeedFactor Function SpeedFactor Statement StartMain Function SyncRobots Function SyncRobots Statement SysErr Statement Tab$ Function TargetOK Statement TaskDone Statement TaskInfo Statement TaskInfo$ Statement TaskState Function TaskState Statement TaskWait Function TC Function TCLim Function TCLim Statement TCSpeed Function TCSpeed Statement TeachOn Statement TillOn Statement TlDef Statement Toff Function Ton Function UBound Statement UpdateDB Function	VxCalib Function VxCalDelete Function VxCalLoad Function VxCalInfo Statement VxCalSave Function VxTrans Statement WaitNet Function WaitPos Function Where Function WindosStatus Statement WriteBin Function Write Function XYLimClr Function XYLimDef Function XY Statement

C-3: Deletion Commands (Sort by Version)

Deletion commands of EPSON RC+ 6.0, 5.0, and 4.0.

Version of EPSON RC+7.0	EPSON RC+ 6.0	EPSON RC+ 5.0	EPSON RC+ 4.0
Ver.7.1.2	SetLCD Function	SetLCD Function	SetLCD Function
Ver.7.0.0	Dir Function Type Function	-	Dir Function Type Function

Appendix D: Predefined Constants

There are several predefined constants for use in SPEL+ program. A project build time, the values for these constants are substituted for the constant name.

Constant name	Value	Use
TRUE	-1	Boolean expression
FALSE	0	Boolean expression
High	1	
Low	0	
Off	0	
On	1	
Above	1	
Below	2	
NoFlip	1	
Flip	2	
Righty	1	
Lefty	2	
J1	1	
J2	2	
J3	4	
J4	8	
J5	16	
J6	32	
J7	64	
CR	CHR$(13)	
CRLF	CHR$(13)+ CHR$(10)	
LF	CHR$(10)	
MB_OK	0	MsgBox flags
MB_OKCANCEL	1	MsgBox flags
MB_ABORTRETRYIGNORE	2	MsgBox flags
MB_YESNOCANCEL	3	MsgBox flags
MB_YESNO	4	MsgBox flags
MB_RETRYCANCEL	5	MsgBox flags
MB_ICONSTOP	16	MsgBox flags
MB_ICONQUESTION	32	MsgBox flags
MB_ICONEXCLAMATION	48	MsgBox flags
MB_ICONINFORMATION	64	MsgBox flags
MB_DEFBUTTON1	0	MsgBox flags
MB_DEFBUTTON2	256	MsgBox flags
IDOK	1	MsgBox return
IDCANCEL	2	MsgBox return
IDABORT	3	MsgBox return
IDRETRY	4	MsgBox return
IDIGNORE	5	MsgBox return
IDYES	6	MsgBox return
IDNO	7	MsgBox return
BACKCOLORMODE_VISUALSTYLE	0	For GUI Builder
BACKCOLORMODE_USER	1	For GUI Builder
BORDERSTYLE_NONE	0	For GUI Builder
BORDERSTYLE_FIXEDSINGLE	1	For GUI Builder
BORDERSTYLE_FIXED3D	2	For GUI Builder
CNV_QUELEN_ALL	0	Cnv_QueLen
CNV_QUELEN_UPSTREAM	1	Cnv_QueLen
CNV_QUELEN_PICKUPAREA	2	Cnv_QueLen
CNV_QUELEN_DOWNSTREAM	3	Cnv_QueLen
DEVID_SELF	21	CLS
DEVID_TP	24	CLS
DEVID_TP3	30	CLS
DIALOGRESULT_NOE	0	For GUI Builder
DIALOGRESULT_OK	1	For GUI Builder

Constant name	Value	Use
DIALOGRESULT_CANCEL	2	For GUI Builder
DLG_IOMON	102	RunDialog
DLG_ROBOTMNG	100	RunDialog
DLG_VGUIDE	110	RunDialog
DOCK_NONE	0	For GUI Builder
DOCK_TOP	1	For GUI Builder
DOCK_BOTTOM	2	For GUI Builder
DOCK_LEFT	3	For GUI Builder
DOCK_RIGHT	4	For GUI Builder
DOCK_FILL	5	For GUI Builder
DROPDOWNSTYLE_SIMPLE	0	For GUI Builder
DROPDOWNSTYLE_DROPDOWN	1	For GUI Builder
DROPDOWNSTYLE_DROPDOWNLIST	2	For GUI Builder
ERROR_DOINGMOTION	2999	For GUI Builder
ERROR_NOMOTION	2998	For GUI Builder
EVENTTASKTYPE_NORMAL	0	For GUI Builder
EVENTTASKTYPE_NOPAUSE	1	For GUI Builder
EVENTTASKTYPE_NOEMGABORT	2	For GUI Builder
FORCE_LESS	0	Force_SetTrigger
FORCE_GREATER	1	Force_SetTrigger
FORCE_XFORCE	2	Force_SetTrigger
FORCE_YFORCE	3	Force_SetTrigger
FORCE_ZFORCE	4	Force_SetTrigger
FORCE_XTORQUE	5	Force_SetTrigger
FORCE_YTORQUE	6	Force_SetTrigger
FORCE_ZTORQUE	7	Force_SetTrigger
FORMBORDERSTYLE_NONE	0	For GUI Builder
FORMBORDERSTYLE_FIXEDSINGLE	1	For GUI Builder
FORMBORDERSTYLE_FIXED3D	2	For GUI Builder
FORMBORDERSTYLE_FIXEDDIALOG	3	For GUI Builder
FORMBORDERSTYLE_SIZABLE	4	For GUI Builder
IMAGEALIGN_TOPLEFT	1	For GUI Builder
IMAGEALIGN_TOPCENTER	2	For GUI Builder
IMAGEALIGN_TOPRIGHT	3	For GUI Builder
IMAGEALIGN_MIDDLELEFT	4	For GUI Builder
IMAGEALIGN_MIDDLECENTER	5	For GUI Builder
IMAGEALIGN_MIDDLERIGHT	6	For GUI Builder
IMAGEALIGN_BOTTOMLEFT	7	For GUI Builder
IMAGEALIGN_BOTTOMCENTER	8	For GUI Builder
IMAGEALIGN_BOTTOMRIGHT	9	For GUI Builder
IOTYPE_INPUT	0	IOLabel function
IOTYPE_OUTPUT	1	IOLabel function
IOTYPE_MEMORY	2	IOLabel function
IOSIZE_BIT	1	IOLabel function
IOSIZE_BYTE	8	IOLabel function
IOSIZE_WORD	16	IOLabel function
LANGID_ENGLISH	0	ErrMsg$
LANGID_JAPANESE	1	ErrMsg$
LANGID_GERMAN	2	ErrMsg$
LANGID_FRENCH	3	ErrMsg$
LANGID_SIMPLIFIED_CHINESE	4	ErrMsg$
LANGID_TRADITIONAL_CHINESE	5	ErrMsg$
MODE_STANDARD	1	PerformMode
MODE_HIGH_SPEED	2	PerformMode
MODE_LOW_OSCILLATION	3	PerformMode
ORIENT_HORIZONTAL	0	For GUI Builder
ORIENT_VERTICAL	1	For GUI Builder
PROGRESSBAR_STYLE_BLOCKS	0	For GUI Builder
PROGRESSBAR_STYLE_CONT	1	For GUI Builder
PROGRESSBAR_STYLE_MARQUEE	2	For GUI Builder
SCROLLBARS_NONE	0	For GUI Builder

Constant name	Value	Use
SCROLLBARS_HORIZ	1	For GUI Builder
SCROLLBARS_VERT	2	For GUI Builder
SCROLLBARS_BOTH	3	For GUI Builder
SETLATCH_PORT_CU_0	24	SetLatch
SETLATCH_PORT_CU_1	25	SetLatch
SETLATCH_PORT_DU1_0	56	SetLatch
SETLATCH_PORT_DU1_1	57	SetLatch
SETLATCH_PORT_DU2_0	280	SetLatch
SETLATCH_PORT_DU2_1	281	SetLatch
SETLATCH_TRIGGERMODE_LEADINGEDGE	1	SetLatch
SETLATCH_TRIGGERMODE_TRAILINGEDGE	0	SetLatch
SHUTDOWN_ALL	0	Shutdown
SHUTDOWN_RESTART	1	Shutdown
SHUTDOWN_EPSONRC	2	Shutdown
SING_NONE	0	AvoidSingularity
SING_THRU	1	AvoidSingularity
SING_THRUROT	2	AvoidSingularity
SING_VSD	3	AvoidSingularity
SING_AUTO	4	AvoidSingularity
SIZEMODE_NORMAL	0	For GUI Builder
SIZEMODE_STRETCHIMAGE	1	For GUI Builder
SIZEMODE_AUTOSIZE	2	For GUI Builder
SIZEMODE_CENTERIMAGE	3	For GUI Builder
SIZEMODE_ZOOM	4	For GUI Builder
STARTPOSITION_MANUAL	0	For GUI Builder
STARTPOSITION_CENTERSCREEN	1	For GUI Builder
STARTPOSITION_CENTERPARENT	2	For GUI Builder
TEXTALIGN_LEFT	1	For GUI Builder
TEXTALIGN_CENTER	2	For GUI Builder
TEXTALIGN_RIGHT	3	For GUI Builder
TEXTALIGN_TOPLEFT	1	For GUI Builder
TEXTALIGN_TOPCENTER	2	For GUI Builder
TEXTALIGN_TOPRIGHT	3	For GUI Builder
TEXTALIGN_MIDDLELEFT	4	For GUI Builder
TEXTALIGN_MIDDLECENTER	5	For GUI Builder
TEXTALIGN_MIDDLERIGHT	6	For GUI Builder
TEXTALIGN_BOTTOMLEFT	7	For GUI Builder
TEXTALIGN_BOTTOMCENTER	8	For GUI Builder
TEXTALIGN_BOTTOMRIGHT	9	For GUI Builder
TICKSTYLE_NONE	0	For GUI Builder
TICKSTYLE_TOPLEFT	1	For GUI Builder
TICKSTYLE_BOTTOMRIGHT	2	For GUI Builder
TICKSTYLE_BOTH	3	For GUI Builder
VISION_SORT_NONE	0	For Vision Guide
VISION_SORT_PIXELX	1	For Vision Guide
VISION_SORT_PIXELY	2	For Vision Guide
VISION_SORT_PIXELXY	3	For Vision Guide
VISION_SORT_CAMERAX	4	For Vision Guide
VISION_SORT_CAMERAY	5	For Vision Guide
VISION_SORT_CAMERAXY	6	For Vision Guide
VISION_SORT_ROBOTX	7	For Vision Guide
VISION_SORT_ROBOTY	8	For Vision Guide
VISION_SORT_ROBOTXY	9	For Vision Guide
VISION_SIZETOFIND_ANY	0	For Vision Guide
VISION_SIZETOFIND_LARGEST	1	For Vision Guide
VISION_SIZETOFIND_SMALLEST	2	For Vision Guide
VISION_BACKCOLOR_NONE	0	For Vision Guide
VISION_BACKCOLOR_BLACK	1	For Vision Guide
VISION_BACKCOLOR_WHITE	2	For Vision Guide
VISION_CAMORIENT_STANDALONE	1	For Vision Guide
VISION_CAMORIENT_FIXEDDOWN	2	For Vision Guide

Constant name	Value	Use
VISION_CAMORIENT_FIXEDUP	3	For Vision Guide
VISION_CAMORIENT_MOBILEJ2	4	For Vision Guide
VISION_CAMORIENT_MOBILEJ4	5	For Vision Guide
VISION_CAMORIENT_MOBILEJ5	6	For Vision Guide
VISION_CAMORIENT_MOBILEJ6	7	For Vision Guide
VISION_FOUNDCOLOR_LIGHTGREEN	1	For Vision Guide
VISION_FOUNDCOLOR_DARKGREEN	2	For Vision Guide
VISION_GRAPHICS_ALL	1	For Vision Guide
VISION_GRAPHICS_POSONLY	2	For Vision Guide
VISION_GRAPHICS_NONE	3	For Vision Guide
VISION_OPERATION_OPEN	1	For Vision Guide
VISION_OPERATION_CLOSE	2	For Vision Guide
VISION_OPERATION_ERODE	3	For Vision Guide
VISION_OPERATION_DILATE	4	For Vision Guide
VISION_OPERATION_SMOOTH	5	For Vision Guide
VISION_OPERATION_SHARPEN1	6	For Vision Guide
VISION_OPERATION_SHARPEN2	7	For Vision Guide
VISION_OPERATION_HORIZEDGE	8	For Vision Guide
VISION_OPERATION_VERTEDGE	9	For Vision Guide
VISION_OPERATION_EDGEDETECT1	10	For Vision Guide
VISION_OPERATION_EDGEDETECT2	11	For Vision Guide
VISION_OPERATION_LAPLACE1	12	For Vision Guide
VISION_OPERATION_LAPLACE2	13	For Vision Guide
VISION_OPERATION_THIN	14	For Vision Guide
VISION_OPERATION_THICKEN	15	For Vision Guide
VISION_OPERATION_BINARIZE	16	For Vision Guide
VISION_OPERATION_ROTATE	17	For Vision Guide
VISION_OPERATION_FLIPHORIZ	18	For Vision Guide
VISION_OPERATION_FLIPVERT	19	For Vision Guide
VISION_OPERATION_FLIPBOTH	20	For Vision Guide
VISION_OPERATION_COLORFILTER	21	For Vision Guide
VISION_OPERATION_SUBTRACTABS	22	For Vision Guide
VISION_OPERATION_ZOOM	23	For Vision Guide
VISION_ACQUIRE_NONE	0	For Vision Guide
VISION_ACQUIRE_STATIONARY	1	For Vision Guide
VISION_ACQUIRE_STROBED	2	For Vision Guide
VISION_TRIGGERMODE_LEADINGEDGE	1	For Vision Guide
VISION_TRIGGERMODE_TRAILINGEDGE	2	For Vision Guide
VISION_THRESHCOLOR_BLACK	1	For Vision Guide
VISION_THRESHCOLOR_WHITE	2	For Vision Guide
VISION_OBJTYPE_CORRELATIO	1	For Vision Guide
VISION_OBJTYPE_BLOB	2	For Vision Guide
VISION_OBJTYPE_EDGE	3	For Vision Guide
VISION_OBJTYPE_POLAR	4	For Vision Guide
VISION_OBJTYPE_LINE	5	For Vision Guide
VISION_OBJTYPE_POINT	6	For Vision Guide
VISION_OBJTYPE_FRAME	7	For Vision Guide
VISION_OBJTYPE_IMAGEOP	8	For Vision Guide
VISION_OBJTYPE_OCR	9	For Vision Guide
VISION_OBJTYPE_CODEREADER	10	For Vision Guide
VISION_OBJTYPE_GEOMETRIC	11	For Vision Guide
VISION_DETAILLEVEL_MEDIUM	1	For Vision Guide
VISION_DETAILLEVEL_HIGH	2	For Vision Guide
VISION_DETAILLEVEL_VERYHIGH	3	For Vision Guide
VISION_IMAGESOURCE_CAMERA	1	For Vision Guide
VISION_IMAGESOURCE_FILE	2	For Vision Guide
VISION_CODETYPE_AUTO	0	For Vision Guide
VISION_CODETYPE_EAN13	2	For Vision Guide
VISION_CODETYPE_CODE39	3	For Vision Guide
VISION_CODETYPE_INTERLEAVED25	4	For Vision Guide
VISION_CODETYPE_CODE128	5	For Vision Guide
VISION_CODETYPE_CODABAR	6	For Vision Guide

Constant name	Value	Use
VISION_CODETYPE_PDF417	8	For Vision Guide
VISION_CODETYPE_QR	10	For Vision Guide
VISION_CODETYPE_EAN8	13	For Vision Guide
VISION_CODETYPE_UPCA	18	For Vision Guide
VISION_CODETYPE_UPCE	19	For Vision Guide
VISION_CODETYPE_UPC	20	For Vision Guide
VISION_EDGETYPE_SINGLE	1	For Vision Guide
VISION_EDGETYPE_PAIR	2	For Vision Guide
VISION_IMAGECOLOR_ALL	1	For Vision Guide
VISION_IMAGECOLOR_RED	2	For Vision Guide
VISION_IMAGECOLOR_GREEN	3	For Vision Guide
VISION_IMAGECOLOR_BLUE	4	For Vision Guide
VISION_IMAGECOLOR_GRAYSCALE	5	For Vision Guide
VISION_POINTTYPE_POINT	0	For Vision Guide
VISION_POINTTYPE_ENDPOINT	1	For Vision Guide
VISION_POINTTYPE_MIDPOINT	2	For Vision Guide
VISION_POINTTYPE_PERPTOLINE	3	For Vision Guide
VISION_POINTTYPE_STARTPOINT	4	For Vision Guide
VISION_POINTTYPE_PERPTOSTARTPOINT	5	For Vision Guide
VISION_POINTTYPE_PERPTOMIDPOINT	6	For Vision Guide
VISION_POINTTYPE_PERPTOENDPOINT	7	For Vision Guide
VISION_REFTYPE_TAUGHTPOINTS	1	For Vision Guide
VISION_REFTYPE_UPWARDCAMERA	2	For Vision Guide
VISION_IMAGESIZE_320X240	1	For Vision Guide
VISION_IMAGESIZE_640X480	2	For Vision Guide
VISION_IMAGESIZE_800X600	3	For Vision Guide
VISION_IMAGESIZE_1024X768	4	For Vision Guide
VISION_IMAGESIZE_1280X1024	5	For Vision Guide
VISION_IMAGESIZE_1600X1200	6	For Vision Guide
VISION_IMAGESIZE_2048X1536	7	For Vision Guide
VISION_IMAGESIZE_2560X1920	8	For Vision Guide
VISION_WINTYPE_RECTANGLE	1	For Vision Guide
VISION_WINTYPE_ROTATEDRECT	2	For Vision Guide
VISION_WINTYPE_CIRCLE	3	For Vision Guide
VISION_ORIENT_BOTH	1	For Vision Guide
VISION_ORIENT_HORIZ	2	For Vision Guide
VISION_ORIENT_VERT	3	For Vision Guide
VISION_DIRECTION_INSIDEOUT	1	For Vision Guide
VISION_DIRECTION_OUTSIDEIN	2	For Vision Guide
VISION_POLARITY_DARK	1	For Vision Guide
VISION_POLARITY_LIGHT	2	For Vision Guide
VISION_PASSTYPE_SOMEFOUND	1	For Vision Guide
VISION_PASSTYPE_ALLFOUND	2	For Vision Guide
VISION_PASSTYPE_SOMENOTFOUND	3	For Vision Guide
VISION_PASSTYPE_ALLNOTFOUND	4	For Vision Guide
WIN_IOMON	-1	For GUI Builder
WIN_TASKMGR	-2	For GUI Builder
WIN_FORCEMON	-3	For GUI Builder
WIN_SIMULATOR	-4	For GUI Builder
WINDOWSTATE_NORMAL	0	WindowsStatus
WINDOWSTATE_MINIMIZED	1	WindowsStatus
WINDOWSTATE_MAXIMIZED	2	WindowsStatus
WithMove	0	Recover
WithoutMove	1	Recover
DRYRUNOFF	1	SF_GetParam Function
SLS_1_HAND_EN	2	SF_GetParam Function
SLS_1_SPEED	3	SF_GetParam Function
SLS_1_ELBOW_EN	4	SF_GetParam Function
SLS_1_JOINT_EN	5	SF_GetParam Function
SLS_1_JOINTSPEED	6	SF_GetParam Function
SLS_1_WRIST_EN	7	SF_GetParam Function

Constant name	Value	Use
SLS_1_SHOULDER_EN	8	SF_GetParam Function
SLS_2_HAND_EN	9	SF_GetParam Function
SLS_2_SPEED	10	SF_GetParam Function
SLS_2_ELBOW_EN	11	SF_GetParam Function
SLS_2_JOINT_EN	12	SF_GetParam Function
SLS_2_JOINTSPEED	13	SF_GetParam Function
SLS_2_WRIST_EN	14	SF_GetParam Function
SLS_2_SHOULDER_EN	15	SF_GetParam Function
SLS_3_HAND_EN	16	SF_GetParam Function
SLS_3_SPEED	17	SF_GetParam Function
SLS_3_ELBOW_EN	18	SF_GetParam Function
SLS_3_JOINT_EN	19	SF_GetParam Function
SLS_3_JOINTSPEED	20	SF_GetParam Function
SLS_3_WRIST_EN	21	SF_GetParam Function
SLS_3_SHOULDER_EN	22	SF_GetParam Function
SLS_T2_HAND_EN	23	SF_GetParam Function
SLS_T2_SPEED	24	SF_GetParam Function
SLS_T2_ELBOW_EN	25	SF_GetParam Function
SLS_T2_JOINT_EN	26	SF_GetParam Function
SLS_T2_JOINTSPEED	27	SF_GetParam Function
SLS_T2_WRIST_EN	28	SF_GetParam Function
SLS_T2_SHOULDER_EN	29	SF_GetParam Function
SLS_T_SPEED	30	SF_GetParam Function
SLS_T_JOINT_EN	31	SF_GetParam Function
SLS_T_JOINTSPEED	32	SF_GetParam Function
SLS_HAND_OFS_X	33	SF_GetParam Function
SLS_HAND_OFS_Y	34	SF_GetParam Function
SLS_HAND_OFS_Z	35	SF_GetParam Function
SLS_1_DELAY	36	SF_GetParam Function
SLS_2_DELAY	37	SF_GetParam Function
SLS_3_DELAY	38	SF_GetParam Function
SLS_JOINT_POS_EN	39	SF_GetParam Function
SLS_JOINT_POS_ANGLE	40	SF_GetParam Function
SLP_A_XU_EN	41	SF_GetParam Function
SLP_A_XU_POS	42	SF_GetParam Function
SLP_A_XL_EN	43	SF_GetParam Function
SLP_A_XL_POS	44	SF_GetParam Function
SLP_A_YU_EN	45	SF_GetParam Function
SLP_A_YU_POS	46	SF_GetParam Function
SLP_A_YL_EN	47	SF_GetParam Function
SLP_A_YL_POS	48	SF_GetParam Function
SLP_A_ZU_EN	49	SF_GetParam Function
SLP_A_ZU_POS	50	SF_GetParam Function
SLP_A_ZL_EN	51	SF_GetParam Function
SLP_A_ZL_POS	52	SF_GetParam Function
SLP_B_XU_EN	53	SF_GetParam Function
SLP_B_XU_POS	54	SF_GetParam Function
SLP_B_XL_EN	55	SF_GetParam Function
SLP_B_XL_POS	56	SF_GetParam Function
SLP_B_YU_EN	57	SF_GetParam Function
SLP_B_YU_POS	58	SF_GetParam Function
SLP_B_YL_EN	59	SF_GetParam Function
SLP_B_YL_POS	60	SF_GetParam Function
SLP_B_ZU_EN	61	SF_GetParam Function
SLP_B_ZU_POS	62	SF_GetParam Function
SLP_B_ZL_EN	63	SF_GetParam Function
SLP_B_ZL_POS	64	SF_GetParam Function

Constant name	Value	Use
SLP_C_XU_EN	65	SF_GetParam Function
SLP_C_XU_POS	66	SF_GetParam Function
SLP_C_XL_EN	67	SF_GetParam Function
SLP_C_XL_POS	68	SF_GetParam Function
SLP_C_YU_EN	69	SF_GetParam Function
SLP_C_YU_POS	70	SF_GetParam Function
SLP_C_YL_EN	71	SF_GetParam Function
SLP_C_YL_POS	72	SF_GetParam Function
SLP_C_ZU_EN	73	SF_GetParam Function
SLP_C_ZU_POS	74	SF_GetParam Function
SLP_C_ZL_EN	75	SF_GetParam Function
SLP_C_ZL_POS	76	SF_GetParam Function
SLP_J2_MON_RAD	77	SF_GetParam Function
SLP_J3_MON_RAD	78	SF_GetParam Function
SLP_J5_MON_RAD	79	SF_GetParam Function
SLP_J6_MON_RAD	80	SF_GetParam Function
SLP_J1_RANGE_MAX	81	SF_GetParam Function
SLP_J1_RANGE_MIN	82	SF_GetParam Function
SLP_J2_RANGE_MAX	83	SF_GetParam Function
SLP_J2_RANGE_MIN	84	SF_GetParam Function
SLP_J3_RANGE_MAX	85	SF_GetParam Function
SLP_J3_RANGE_MIN	86	SF_GetParam Function
SLP_J4_RANGE_MAX	87	SF_GetParam Function
SLP_J4_RANGE_MIN	88	SF_GetParam Function
SLP_J5_RANGE_MAX	89	SF_GetParam Function
SLP_J5_RANGE_MIN	90	SF_GetParam Function
SLP_J6_RANGE_MAX	91	SF_GetParam Function
SLP_J6_RANGE_MIN	92	SF_GetParam Function
SIN_1_SLS_1_EN	93	SF_GetParam Function
SIN_1_SLS_2_EN	94	SF_GetParam Function
SIN_1_SLS_3_EN	95	SF_GetParam Function
SIN_1_SLP_A_EN	96	SF_GetParam Function
SIN_1_SLP_B_EN	97	SF_GetParam Function
SIN_1_SLP_C_EN	98	SF_GetParam Function
SIN_1_SG_EN	99	SF_GetParam Function
SIN_1_ESTOP_EN	100	SF_GetParam Function
SIN_2_SLS_1_EN	101	SF_GetParam Function
SIN_2_SLS_2_EN	102	SF_GetParam Function
SIN_2_SLS_3_EN	103	SF_GetParam Function
SIN_2_SLP_A_EN	104	SF_GetParam Function
SIN_2_SLP_B_EN	105	SF_GetParam Function
SIN_2_SLP_C_EN	106	SF_GetParam Function
SIN_2_SG_EN	107	SF_GetParam Function
SIN_2_ESTOP_EN	108	SF_GetParam Function
SIN_3_SLS_1_EN	109	SF_GetParam Function
SIN_3_SLS_2_EN	110	SF_GetParam Function
SIN_3_SLS_3_EN	111	SF_GetParam Function
SIN_3_SLP_A_EN	112	SF_GetParam Function
SIN_3_SLP_B_EN	113	SF_GetParam Function
SIN_3_SLP_C_EN	114	SF_GetParam Function
SIN_3_SG_EN	115	SF_GetParam Function
SIN_3_ESTOP_EN	116	SF_GetParam Function
SIN_4_SLS_1_EN	117	SF_GetParam Function
SIN_4_SLS_2_EN	118	SF_GetParam Function
SIN_4_SLS_3_EN	119	SF_GetParam Function
SIN_4_SLP_A_EN	120	SF_GetParam Function
SIN_4_SLP_B_EN	121	SF_GetParam Function

Constant name	Value	Use
SIN_4_SLP_C_EN	122	SF_GetParam Function
SIN_4_SG_EN	123	SF_GetParam Function
SIN_4_ESTOP_EN	124	SF_GetParam Function
SIN_5_SLS_1_EN	125	SF_GetParam Function
SIN_5_SLS_2_EN	126	SF_GetParam Function
SIN_5_SLS_3_EN	127	SF_GetParam Function
SIN_5_SLP_A_EN	128	SF_GetParam Function
SIN_5_SLP_B_EN	129	SF_GetParam Function
SIN_5_SLP_C_EN	130	SF_GetParam Function
SIN_5_SG_EN	131	SF_GetParam Function
SIN_5_ESTOP_EN	132	SF_GetParam Function
SOUT_1_STO	133	SF_GetParam Function
SOUT_1_SLS_1	134	SF_GetParam Function
SOUT_1_SLS_2	135	SF_GetParam Function
SOUT_1_SLS_3	136	SF_GetParam Function
SOUT_1_SLS_T2	137	SF_GetParam Function
SOUT_1_SLS_T	138	SF_GetParam Function
SOUT_1_SLP_A	139	SF_GetParam Function
SOUT_1_SLP_B	140	SF_GetParam Function
SOUT_1_SLP_C	141	SF_GetParam Function
SOUT_1_EP_RC	142	SF_GetParam Function
SOUT_1_EP_TP	143	SF_GetParam Function
SOUT_1_EN_SW	144	SF_GetParam Function
SOUT_2_STO	145	SF_GetParam Function
SOUT_2_SLS_1	146	SF_GetParam Function
SOUT_2_SLS_2	147	SF_GetParam Function
SOUT_2_SLS_3	148	SF_GetParam Function
SOUT_2_SLS_T2	149	SF_GetParam Function
SOUT_2_SLS_T	150	SF_GetParam Function
SOUT_2_SLP_A	151	SF_GetParam Function
SOUT_2_SLP_B	152	SF_GetParam Function
SOUT_2_SLP_C	153	SF_GetParam Function
SOUT_2_EP_RC	154	SF_GetParam Function
SOUT_2_EP_TP	155	SF_GetParam Function
SOUT_2_EN_SW	156	SF_GetParam Function
SOUT_3_STO	157	SF_GetParam Function
SOUT_3_SLS_1	158	SF_GetParam Function
SOUT_3_SLS_2	159	SF_GetParam Function
SOUT_3_SLS_3	160	SF_GetParam Function
SOUT_3_SLS_T2	161	SF_GetParam Function
SOUT_3_SLS_T	162	SF_GetParam Function
SOUT_3_SLP_A	163	SF_GetParam Function
SOUT_3_SLP_B	164	SF_GetParam Function
SOUT_3_SLP_C	165	SF_GetParam Function
SOUT_3_EP_RC	166	SF_GetParam Function
SOUT_3_EP_TP	167	SF_GetParam Function
SOUT_3_EN_SW	168	SF_GetParam Function
POS_ROT_U	169	SF_GetParam Function
POS_ROT_V	170	SF_GetParam Function
POS_ROT_W	171	SF_GetParam Function
POS_OFS_X	172	SF_GetParam Function
POS_OFS_Y	173	SF_GetParam Function
POS_OFS_Z	174	SF_GetParam Function
SF_TOOLVERSION	1	SF_GetParam$ Function
SF_CHECKSUM	2	SF_GetParam$ Function
SF_LAST_MODIFIED	3	SF_GetParam$ Function
SF_ROBOT_MODEL_NAME	4	SF_GetParam$ Function

Constant name	Value	Use
SF_ROBOT_CHECKSUM	5	SF_GetParam$ Function
SF_HOFS	6	SF_GetParam$ Function
SF_HOFS_LAST_MODIFIED	7	SF_GetParam$ Function
SLS_1	1	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_2	2	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_3	3	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_T	9	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_T2	10	SF_LimitSpeedS, SF_LimitSpeedSEnable