EPSON

EPSON RC+ 5.0 Ver.5.4

SPEL⁺ Language Reference

Rev.4   EM12XS2363F

EPSON RC+ 5.0 (Ver.5.4)　　SPEL$^+$ Language Reference　Rev.4

EPSON RC+ 5.0 (Ver.5.4)

SPEL+ Language Reference

Rev.4

FOREWORD

Thank you for purchasing our robot products.

This manual contains the information necessary for the correct use of the Manipulator.

Please carefully read this manual and other related manuals before installing the robot system.

Keep this manual handy for easy access at all times.

WARRANTY

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

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

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

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

2. Malfunctions caused by customers' unauthorized disassembly.

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

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

Warnings, Cautions, Usage:

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

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

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

TRADEMARKS

Microsoft, Windows, and Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other brand and product names are trademarks or registered trademarks of the respective holders.

TRADEMARK NOTATION IN THIS MANUAL

Microsoft® Windows® XP Operating system
Microsoft® Windows® Vista Operating system
Microsoft® Windows® 7 Operating system
Throughout this manual, Windows XP, Windows Vista, and Windows 7 refer to above respective operating systems. In some cases, Windows refers generically to Windows XP, and Windows Vista, and Windows 7.

NOTICE

No part of this manual may be copied or reproduced without authorization.
The contents of this manual are subject to change without notice.
Please notify us if you should find any errors in this manual or if you have any comments regarding its contents.

INQUIRIES

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

Please prepare the following items before you contact us.

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

SERVICE CENTER

MANUFACTURER

SEIKO EPSON CORPORATION

Toyoshina Plant

Factory Automation Systems Dept.

6925 Toyoshina Tazawa,

Azumino-shi, Nagano, 399-8285

JAPAN

TEL   : +81-(0)263-72-1530

FAX   : +81-(0)263-72-1495

SUPPLIERS

North & South America   **EPSON AMERICA, INC.**

Factory Automation/Robotics

18300 Central Avenue

Carson, CA 90746

USA

TEL   : +1-562-290-5900

FAX   : +1-562-290-5999

E-MAIL : info@robots.epson.com

Europe   **EPSON DEUTSCHLAND GmbH**

Factory Automation Division

Otto-Hahn-Str.4

D-40670 Meerbusch

Germany

TEL   : +49-(0)-2159-538-1391

FAX   : +49-(0)-2159-538-3170

E-MAIL : robot.infos@epson.de

China   **EPSON China Co., Ltd**

Factory Automation Division

7F, Jinbao Building No. 89 Jinbao Street

Dongcheng District, Beijing,

China, 100005

TEL   : +86-(0)-10-8522-1199

FAX   : +86-(0)-10-8522-1120

Taiwan   **EPSON Taiwan Technology & Trading Ltd.**

Factory Automation Division

14F, No.7, Song Ren Road, Taipei 110

Taiwan, ROC

TEL   : +886-(0)-2-8786-6688

FAX   : +886-(0)-2-8786-6677

Southeast Asia	**EPSON Singapore Pte Ltd.**
India	Factory Automation System
	1 HarbourFrontPlace, #03-02
	HarbourFront Tower one, Singapore
	098633
	TEL : +65-(0)-6586-5696
	FAX : +65-(0)-6271-3182

Korea	**EPSON Korea Co, Ltd.**
	Marketing Team (Robot Business)
	11F Milim Tower, 825-22
	Yeoksam-dong, Gangnam-gu, Seoul, 135-934
	Korea
	TEL : +82-(0)-2-3420-6692
	FAX : +82-(0)-2-558-4271

Japan	**EPSON SALES JAPAN CORPORATION**
	Factory Automation Systems Department
	Nishi-Shinjuku Mitsui Bldg.6-24-1
	Nishishinjuku. Shinjuku-ku. Tokyo. 160-8324
	JAPAN
	TEL : +81-(0)3-5321-4161

SAFETY PRECAUTIONS

Installation of robots and robotic equipment should only be performed by qualified personnel in accordance with national and local codes. Please carefully read this manual and other related manuals when using this software.

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

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

TABLE OF CONTENTS

Summary of SPEL+ Commands

The following is a summary of SPEL$^+$ commands.

System Management Commands

Reset	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+ 5.0 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 a execution line on the LCD.
SafetyOn	Return the Safety Door open status.

Robot Control Commands

Power	Sets / returns servo power mode.
Motor	Sets / returns motor status.
SFree	Removes servo power from the specified servo axis.
SLock	Restores servo power to the specified servo axis.
Jump	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.
Arch	Sets / returns arch parameters for Jump motion.
LimZ	Sets the upper Z limit for the Jump command.
Sense	
JS	Returns status of Sense operation.

JT	Returns the status of the most recent Jump command for the current robot.
Go	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.
Inertia	Specifies or displays the inertia settings of the robot arm.
Weight	Specifies or displays the weight settings of 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.
CV Move	Performs the continuous spline path motion defined by the Curve instruction.
SpeedS	Sets / returns speed for linear motion commands.
AccelS	Sets / returns acceleration and deceleration for linear motion.
SpeedR	Sets / returns speed for tool rotation.
AccelR	Sets / returns acceleration and deceleration for tool rotation.
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.
AtHome	Returns if the current robot is in its Home position or not.

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

InPos	Checks if robot is in position (not moving).
CurPos	Returns current position while moving.
TCPSpeed	Returns calculated current tool center point velocity.
Pallet	Defines a pallet or returns a pallet point.
Fine	Sets positioning error limits.
QP	Sets / returns Quick Pause status.
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	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.
InsidePlane	Returns the check status of the approach check plane.
Find	Specifies or displays the condition to store coordinates during motion.
FindPos	Returns a robot point stored by Fine 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.
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.
JRange	Sets / returns joint limits for one joint.
Range	Sets limits for all joints.
XYLim	Sets / returns Cartesian limits of robot envelope.
XYLimClr	Clears the XYLim definition.
XYLimDef	Returns whether XYLim has been defined or not.
XY	Returns a point from individual coordinates that can be used in a point expression.
Dist	Returns the distance between two robot points.
PTPBoost	Sets / returns boost values for small distance PTP 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.
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
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
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.
CX	Sets / returns the X axis coordinate of a point.
CY	Sets / returns the Y axis coordinate of a point.
CZ	Sets / returns the Z axis coordinate of a point.
CU	Sets / returns the U axis coordinate of a point.
CV	Sets / returns the V axis coordinate of a point.
CW	Sets / returns the W axis coordinate of a point.
Pls	Returns the pulse value of one joint.
Agl	Returns joint angle at current position.
PAgl	Return a joint value from a specified point.
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.
RealPose	Returns the current position of the specified robot.
PPls	Return the pulse position of a specified joint value from a specified point.
Here	Teach a robot point at the current position.
Where	Displays current robot position data.

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

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 intializes the average torque for one or more joints.
ATRQ	Displays the average torque for the specified joint.
PTCLR	Clears and intializes 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.

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. Used for I/O.
InW	Returns the status of the specified input word port. Used for I/O.
InBCD	Reads 8 bits of inputs in BCD format.
InReal	Reads an input data of 2 words (32 bits) as a floating-point data (IEEE754 compliant) of 32 bits.
Out	Sets / returns 8 bits of outputs. Used for I/O.
OutW	Simultaneously sets 16 output bits. Used for I/O.
OpBCD	Simultaneously sets 8 output bits using BCD format.
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).
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	Input one or more variables from current display window.
Print	Display characters on current display window.
Line Input	Input a string from the current display window.
Input #	Input one or more variables from a communications port.
Print #	Output characters to a communications port.
Line Input #	Input a string from a communications port.
Lof	Returns the number of lines in a communications buffer.
SetIn	For Virtual IO, sets specified input port (8 bits) to the specified value.
SetInReal	Set a hypothetical I/O input port (2 words (32 bits)) as a floating-point data (IEEE789 compliant) of 32 bits.
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.
OpenCom	Open an RS-232 communication port.
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.
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.

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

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.
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.

Coordinate Change Commands

Arm	Sets / returns current arm.
ArmSet	Defines an arm.
ArmDef	Returns status of arm definition.
ArmClr	Clears an arm definition.
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 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.
VxCalib	Creates the calibration data.
VxCalDelete	Deletes the calibration data.
VxCalInfo	Returns the calibration completion status / calibration data.
VxCalLoad	Loads the calibration data from the file.
VxCalSave	Saves the calibration data to the file.
VxTrans	Converts the pixel coordinates to the robot coordinates and returns the converted the point data.

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.
Trap	Specify a trap handler.
OnErr	Defines an error handler.
Era	Returns robot joint number for last error.
Erf$	Returns the function name for last error.
Erl	Returns line number of error.
Err	Returns error number.
Ert	Returns task number of error.
ErrMsg$	Returns error message.
Signal	Sends a signal to tasks executing WaitSig.
SyncLock	Synchronizes tasks using a mutual exclusion lock.
SynUnlock	Unlocks a sync ID that was previously locked with SyncLock.
WaitSig	Waits for a signal from another task.
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 contoller 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.

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

Pseudo Statements

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

Numeric Value Commands

Ctr	Return the value of a counter.
CTReset	Resets a counter.
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	Clear 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.
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.
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.
Mod	Modulus operator.
Not	Not operator.
RShift	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.
Integer	Declares integer variables.
Long	Declares long integer variables.
Real	Declares real variables.
String	Declares string variables.

Commands used with VB Guide

SPELCom_Event	Fire an event in SpelNetLib client.

SPEL$^+$ Language Reference

This section describes each SPEL$^+$ command as follows:

Syntax   Syntax describes the format used for each command. For some commands, there is more than one syntax shown, along with a number that is referenced in the command description. Parameters are shown in italics.

Parameters   Describes each of the parameters for this command.

Return Values   Describes any values that the command returns.

Description   Gives details about how the command works.

Notes   Gives additional information that may be important about this command.

See Also   Shows other commands that are related to this command. Refer to the Table of Contents for the page number of the related commands.

Example   Gives one or more examples of using this command.

SYMBOLS

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

> **>**   May be used from the command window.

S   May be used as a statement in a SPEL$^+$ program.

F   May be used as a Function in a SPEL$^+$ program.

!...!   Parallel Processing

Processes input/output statements in parallel with motion.

> **S**

Syntax

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

Parameters

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

Description

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

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

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

Notes

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

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

What happens when the Till statement is used to stop the arm prior to completing the intended motion

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

Whem the 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.

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, Pulse

!...! Parallel Processing Example

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

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

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

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

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

#define

Defines identifier to be replaced by specified replacement string.

S

Syntax

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

Parameters

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

- The first character must be alphabetic while the characters which follow may be alphanumeric or an underscore (_).

- Spaces or tab characters are not allowed as part of the *identifier* .

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

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

- Spaces or tabs are allowed in replacement strings.

- Identifiers used with other #define statements cannot be used as replacement strings.

- If the comment symbol (') is included, the characters following the comment symbol will be treated as a comment and will not be included in the replacement string.

- The replacement string may be omitted. In this case the specified identifier is replaced by "nothing" or the null string. This actually deletes the identifier from the program

Description

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

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

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

Notes

Using #define for variable declaration or label substitutions will cause an error:
It should be noted that usage of the #define instruction for variable declaration will cause an error.

See Also

#ifdef
#ifndef

#define Example

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

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

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

Integer a

a = 25

SHOWVAL(a)
```

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

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.

S

Syntax

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

Parameters

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

Description

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

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

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

Notes

Difference between #ifdef and #ifndef

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

See Also

#define, #ifdef

#ifndef Example

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

$\boxed{\textbf{S}}$

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

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

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

  Include files are used to contain #define statements.

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

  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.

$\boxed{\textbf{S}}$

Syntax
    #undef *identifier*

Parameters
    identifier   Keyword used in a previous #define statement.

See Also
    #define, #ifdef, #ifndef

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.

> | S

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 Power On
Motor On
SFree, SLock
Reset
Stop button or Ctrl + C Key
```

Notes

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

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

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

Accel vs. AccelS

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

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.

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

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

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

<Example 2>

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

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

<Example 3>

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

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

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

Accel Function

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

AccelMax Function

Returns maximum acceleration value limit available for Accel.

> F

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.

> S

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).

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 Power On
Motor On
SFree, SLock
Reset
Stop button or Ctrl + C Key
```

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

AccelR Statement Example

```
AccelR 360, 200
```

AccelR Function

Returns specified tool rotation acceleration value.

> **>** **F**

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, etc.

> S

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 differs by robot type as follows.

	accel / decel	*departAccel / departDecel* *approAccel / approDecel*
SCARA robot	0.1 to 25000	0.1 to 25000
6-axis robot	0.1 to 25000	0.1 to 50000

(mm/sec^2)

Return Values
Displays Accel and Decel values when used without parameters

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

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

```
Controller Power On
Motor On
SFree, SLock
Reset
Stop button or Ctrl + C Key
```

Notes

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

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

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

See Also

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

AccelS Example

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

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

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

<Example 2>

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

```
>AccelS 1000

>AccelS
Low Power Mode
    1000.000    1000.000
>
```

AccelS Function

Returns acceleration or deceleration for CP motion commands.

> **>** **F**

Syntax
>   **AccelS**(*paramNumber*)

Parameters
>   *paramNumber*   Integer expression which can have the following values:
>
>   1: acceleration value
>
>   2: deceleration value
>
>   3: depart acceleration value for Jump3, Jump3CP
>
>   4: depart deceleration value for Jump3, Jump3CP
>
>   5: approach acceleration value for Jump3, Jump3CP
>
>   6: approach deceleration value for Jump3, Jump3CP

Return Values
>   Real value from 0 - 5000 mm/sec/sec

See Also
>   AccelS Statement, Arc3, SpeedS, Jump3, Jump3CP

AccelS Function Example

```
Real savAccelS

savAccelS = AccelS(1)
```

Acos Function

Returns the arccosine of a numeric expression.

F

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.

F

Syntax
> **Agl**(*jointNumber*)

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

Return Values
> The joint angle for selected rotational joint or position for selected linear joints.

Description
> The **Agl** function is used to get the joint angle for the selected rotational joint or position for the selected linear joint.
>
> If the selected joint is rotational, **Agl** returns the current angle, as measured from the selected joint's 0 position, in degrees. The returned value is a real number.
>
> If the selected joint is a linear joint, **Agl** returns the current position, as measured from the selected joint's 0 position, in mm. The returned value is a real number.
>
> If an auxiliary arm is selected with the Arm statement, **Agl** returns the angle (or position) from the standard arm's 0 pulse position to the selected arm.

See Also
> PAgl, Pls, PPls

Agl Function Example
> The following examples are done from the command window using the Print instruction.

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

AglToPls Function

Converts robot angles to pulses.

$\boxed{\textbf{F}}$

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

Parameters
j1 - j6   Real expressions representing joint angles.

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

Description
Use AglToPls to create a point from joint angles.

Note

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

For example:

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

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

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

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

See Also
Agl, JA, Pls

AglToPls Function Example

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

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 axis of the specified local coordinate system.

$\boxed{F}$

Syntax

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

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.

Description

While operating the 6-axis robot, 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 axis of the specified local coordinate system.

For robots except the 6-axis robot, 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
```

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.

F

Syntax
(2) 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, 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 the 6-axis robot, 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
>
```

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.

<div style="text-align:right">

`>` `S`

</div>

Syntax

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

Parameters

midPoint   Point expression. The middle point (taught previously by the user) which the arm travels through on its way from the current point to *endPoint*.

endPoint   Point expression. The end point (taught previously by the user) which the arm travels to during the arc type motion. This is the final position at the end of the circular move.

ROT   Optional. :Decides the speed/acceleration/deceleration in favor of tool rotation.

ECP   Optional. External control point motion. This parameter is valid when the ECP option is enabled.

CP   Optional. Specifies continuous path motion.

searchExpr Optional. A Till or Find expression.
           Till | Find
           Till Sw(*expr*) = {**On | Off**}
           Find Sw(*expr*) = {**On | Off**}

!...!   Parallel processing statements may be used with the Arc statement. These are optional. (Please see the Parallel Processing description for more information.)

Description

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

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

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

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

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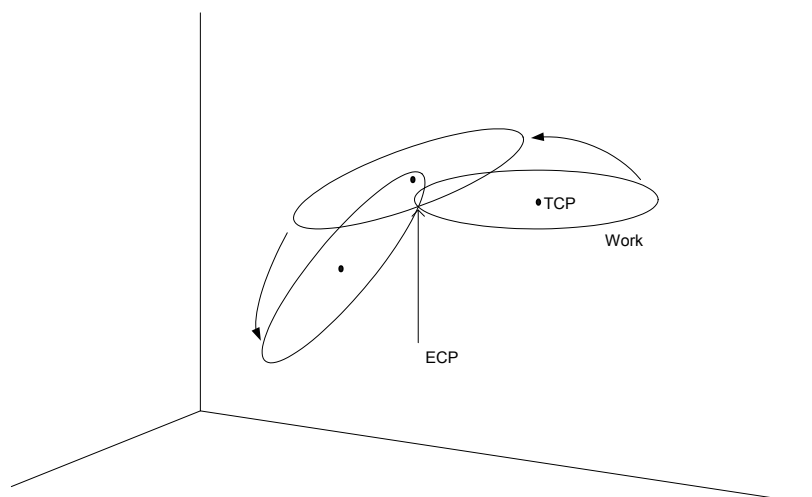
Setting Speed and Acceleration for Arc Motion

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

Notes

Arc Instruction works in Horizontal Plane Only

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

Range Verification for Arc Instruction

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

Suggested Motion to Setup for the Arc Move

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

Using Arc, Arc3 with CP

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

Potential Errors

Changing Hand Attributes

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

Attempt to Move Arm Outside Work Envelope

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

See Also

!Parallel Processing!, AccelS, Move, SpeedS

Arc Example

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

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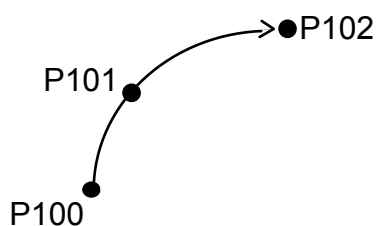

Tip

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

Arch Statement

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

> S

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 (0-6) making a total of 7 entries into the **Arch** table. (see default **Arch** Table below)

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

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

Return Values

Displays **Arch** Table when used without parameters.
The **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 diagram below)
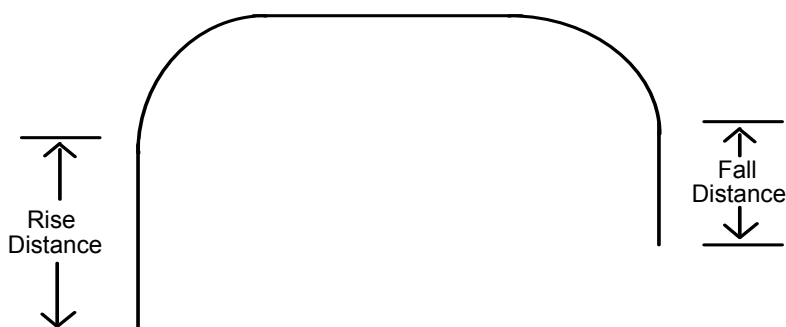
There are a total of 8 entries in the **Arch** Definition Table with 7 of them (0-6) being user definable. The 8th entry (**Arch** 7)is the default Arch which actually specifies no arch at all which is referred to as Gate Motion. (See Gate Motion diagram below) The Jump instruction used with the default **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)

P0   P1
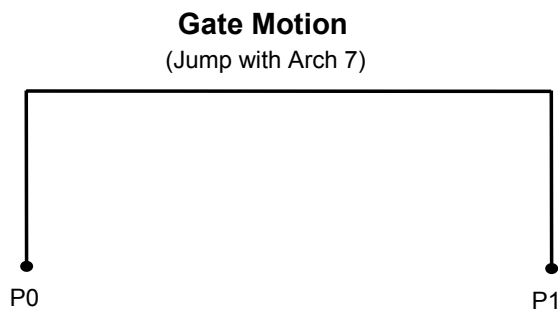
Arch Table Default Values:

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

Notes

Jump Motion trajectory changes depending on motion and speed

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

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

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

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

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

Another Cause of Gate Motion

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

Arch values are Maintained

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

See Also

Jump, Jump3, JumpCP

Arch 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.

> | F

Syntax

Arch(*archNumber, paramNumber*)

Parameters

archNumber   Integer expression representing arch setting to retrieve parameter from (0 to 6).

paramNumber   1: depart distance

                         2: approach distance

Return Value

Real number containing distance.

See Also

Arch statement

Arch Function Example

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

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

Arm Statement

Selects or displays the arm number to use.

> S

Syntax

(1) Arm *armNumber*
(2) Arm

Parameters

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

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

Notes

Arm 0

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

Arm Number Not Defined

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

See Also
    ArmClr, ArmSet, ECPSet, TLSet

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

    From the 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.

> **>** **F**

Syntax
> **Arm**

Return Values
> Integer containing the current arm number.

See Also
> Arm Statement

Arm Function Example

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

ArmClr Statement

Clears (undefines) an arm definition.

> **[>]** **[S]**

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

> Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also

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

ArmClr Example

```
ArmClr 1
```

ArmDef Function

Returns arm definition status.

> F

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 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 displays auxiliary arms.

> S

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

Parameters

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

SCARA Robots (including RS series)

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

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.

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

Notes

Arm 0

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

| SCARA Robot | Cartesian Robot | SCARA Robots (RS Series) |
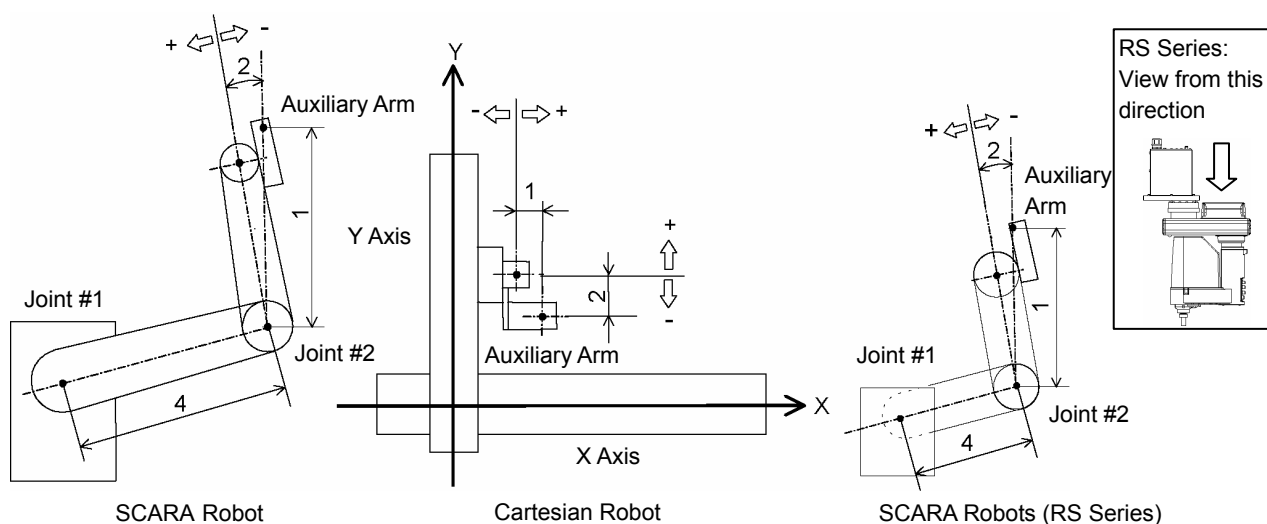
See Also

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.

> **F**

Syntax
  ArmSet(*armNumber, paramNumber*)

Parameters
  armNumber   Integer expression representing the arm number to retrieve values for.
  paramNumber   Integer expression representing the parameter to retrieve (0 to 5), as described below.

SCARA Robots (including RS series)

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

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

SCARA Robot   Cartesian Robot   SCARA Robots (RS Series)
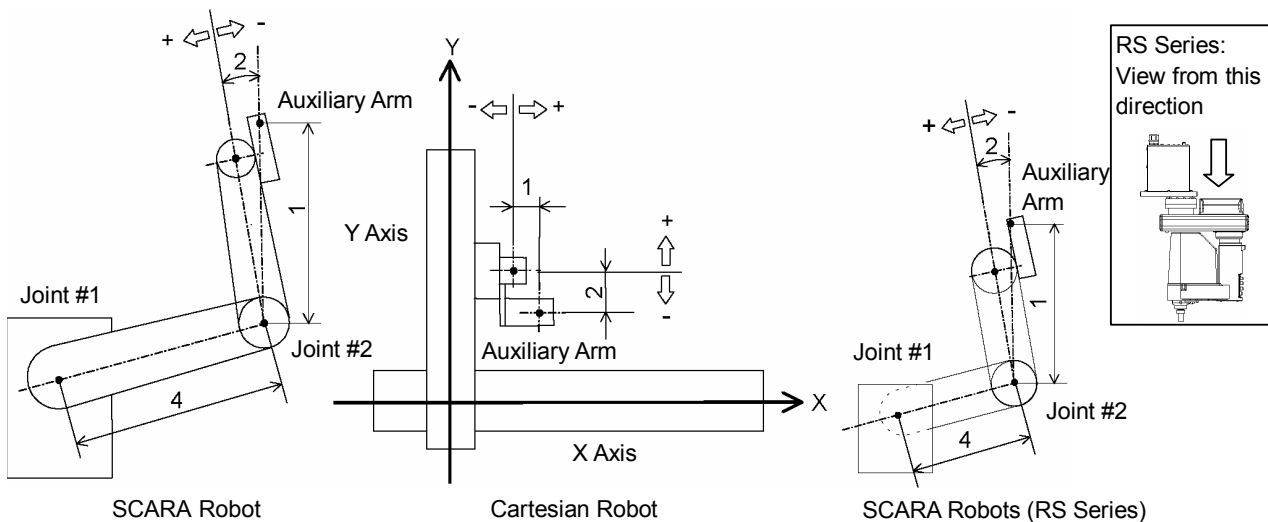
See Also
  ArmClr, ArmSet Statement

ArmSet Function Example

```
Real x

x = ArmSet(1, 1)
```

Asc Function

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

F

Syntax
 Asc(*string*)

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

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

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

Notes

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

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.

$\boxed{\textbf{F}}$

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
```

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 radans.

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

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

Atan Function Example

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

Atan2 Function

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

<div style="border:1px solid;display:inline-block;padding:4px">**F**</div>

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 intializes the average torque for one or more joints.

> | S

Syntax

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

Parameters

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

Description

    ATCLR clears the average torque values for the specified joints.

    You must execute ATCLR before executing ATRQ.

See Also

    ATRQ, PTRQ

ATCLR Statement Example

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

AtHome Function

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

$\boxed{\text{F}}$

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

ATRQ Statement

> S

Displays the average torque for the specified joint.

Syntax
>   **ATRQ** [*jointNumber*]

Parameters
>   *jointNumber*   Optional. Integer expression representing the joint number.

Return Values
>   Displays current average torque values for all joints.

Description
>   **ATRQ** displays the average RMS (root-mean-square) torque of the specified joint. The loading state of the motor can be obtained by this instruction. The result is a real value from 0 to 1 with 1 being maximum average torque.
>
>   You must execute ATCLR before this command is executed.
>
>   This instruction is time restricted. You must execute ATRQ within 60 seconds after ATCLR is executed. When this time is exceeded, error 4030 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.

$\boxed{\text{F}}$

Syntax
> **ATRQ** (*jointNumber*)

Parameters
> *jointNumber* Integer expression representing the joint number.

Return Values
> Real value from 0 to 1.

Description
> The **ATRQ** function returns the average RMS (root-mean-square) torque of the specified joint. The loading state of the motor can be obtained by this instruction. The result is a real value from 0 to 1 with 1 being maximum average torque.
>
> You must execute ATCLR before this function is executed.
>
> This instruction is time restricted. You must execute ATRQ within 60 seconds after ATCLR is executed. When this time is exceeded, error 4030 occurs.

See Also
> **ATRQ** Statement, PTCLR, PTRQ Statement

ATRQ **Function Example**
> This example uses the **ATRQ** function in a program:

```
Function CheckAvgTorque
  Integer i

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

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

AutoLJM Statement

Sets the Auto LJM function.

S

Syntax

AutoLJM { On | Off }

Parameter

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

Description

AutoLJM is avairable 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 example

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

AutoLJM Function

Returns the state of the AutoLJM.

$\boxed{\textbf{F}}$

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
```

AvoidSingularity Statement

Sets the singularity avoiding function.

S

Syntax

AvoidSingularity { 1 | 0 }

Parameter

1 | 0   1: Enables the singularity avoiding function.
              0: Disables the singularity avoiding function.

Description

AvoidSingularity is avairable for following commands.
Move, Arc, Arc3

A singurality avoiding function is to prevent accceleration errors when the vertical 6-axis robot approaches to the singularity in CP motion by passing a different trajectory and returning to the original trajectory after passing the singularity. This function is only applicable for the wrist 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 singurarity avoidance to ensure compatibility with software which does not support the singularity avoiding function, or to avoid a trajectory gap, disable the function.

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).

Notes

Condition setting of singularity neighborhood

To determine whether the manipulator approaches to the singularity neighborhood, angle of Joint #5 and angular velocity of Joint #4 are used. By default, Joint #5 angle is set to ±5 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.

See Also

AvoidSingularity Function, SingualrityAngle, SingularitySpeed

AvoidSingularity Example

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

AvoidSingularity Function

Returns the state of AvoidSingularity.

<div style="float: right; border: 1px solid black; padding: 4px;">F</div>

Syntax

AvoidSingularity

Return values

0   = Singularity avoiding function disabled
1   = 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.

> | S

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)

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

Notes

Changing the base coordinate system affects all local definitions
When base coordinates are changed, all local coordinate systems must be re-defined.

See Also
Local

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

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

BClr Function

Clear one bit in a number and return the new value

F

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
   BSet, BTst

BClr Example

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

BGo Statement

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

> S

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

Parameters

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

Description

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

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

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

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

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

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

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

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

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 Example

```
> BGo XY(100, 0, 0, 0)  'Move 100mm 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 $\boxed{\textbf{>}}$ $\boxed{\textbf{S}}$

Syntax

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

Parameters

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

Description

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

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

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

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

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

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

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

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

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

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

         EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

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

Notes

Using BMove with CP

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

See Also

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

BMove Example

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

Function BMoveTest

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

  P1 = XY(300, 300, -20, 0)
  P2 = XY(300, 300, -20, 0) /L
  Local 1, 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. (1 byte whole number).

$\boxed{\textbf{S}}$

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 to the upper bound value.
>
>   The total available number of array elements for local and global preserve variables is 1000.
>
>   The total available number of array elements for global and module variables is 10000.
>
>   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

>   **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, Integer, Long, Real, String

Boolean Statement Example

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

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

Box Statement

Specifies and displays the approach check area.

> S

Syntax

(1) **Box** *AreaNum*, *minX*, *maxX*, *mixY*, *maxY*, *minZ*, *maxZ [,Remote output logic]*
(2) **Box** *AreaNum*
(3) **Box**

Parameters

AreaNum   Integer expression representing the area number from 1 to 15.

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 output logic   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.

Return Values

When only *AreaNum* is specified, the area setting of the specified area is displayed. When all the parameters are omitted, the area settings for all area numbers 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 and is between the specified maximum and minimum X, Y, and Z.

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 InsideBox function to get the result of the approach check. InsideBox 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.
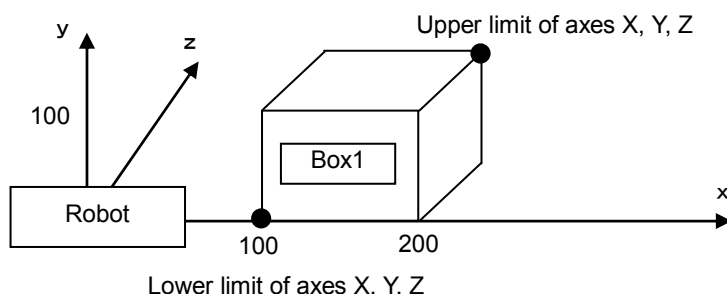
Configure the Box 1 from Robot 1 position
```
Box 1, 100, 200, 0, 100, 0, 100
```

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

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

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, definthe *minZ* and *maxZ* to be 0. For example Box 1, 200, 300, 0, 500, 0, 0.

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.

Tip

Set Box statement from Robot Manager

EPSON RC+ 5.0 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, InsideBox, Plane

Box Statement Example

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

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

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

Box Function

Returns the specified approach check area.

F

Syntax
Box(*AreaNum*, *limit*)

Parameters

AreaNum	Integer expression representing the area number from 1 to 15.
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, InsideBox

Box Function Example

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

BoxClr Statement

Clears the definition of approach check area.

> **S**

Syntax
BoxClr *AreaNum*

Parameters
AreaNum   Integer expression representing the area number from 1 to 15.

Description
Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also
Box, BoxDef, InsideBox

BoxClr Function 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.

F

Syntax
BoxDef(*AreaNum*)

Parameters
AreaNum   Integer expression representing the area number from 1 to 15.

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

See Also
Box, BoxClr, 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. It can only be executed as a command command. This command is intended for use by maintenance personel only.

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

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

See Also

Motor, Power, Reset, SFree, SLock

Brake 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 Statement

Brake 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.

$\boxed{\textbf{F}}$

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, BTst

BSet Example

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

BTst Function

Returns the status of 1 bit in a number.

F

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, Bset

BTst Example

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

Byte Statement

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

$\boxed{\textbf{S}}$

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 to the upper bound value.

The total available number of array elements for local and global preserve variables is 1000.

The total available number of array elements for global and module variables is 10000.

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

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

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

```
Function Test
    Byte A(10)          'Single dimension array of byte
    Byte B(10, 10)      'Two dimension array of byte
    Byte C(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
```

Call Statement

Calls a user function.

$\boxed{\textbf{S}}$

Syntax
> **Call** *funcName* [(*argList*)]

Parameters

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

Description

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

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

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

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

See Also
Function, GoSub

Call Statement Example

<File1: MAIN.PRG>

```
Function main
    Call InitRobot
Fend
```

<File2: INIT.PRG>

```
Function InitRobot

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

ChkCom Function

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

$\boxed{\textbf{F}}$

Syntax
    ChkCom (*portNumber)*

Parameters
    portNumber Integer expression for port number to check.

Return Values
    Number of characters received (integer).

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

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

See Also
    CloseCom, OpenCom, Read, Write

ChkCom Example

```
    Integer numChars

    numChars = ChkCom(1)
```

ChkNet Function

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

$\boxed{\textbf{F}}$

Syntax
> **ChkNet** (*portNumber)*

Parameters
> *portNumber* Integer expression for port number to check.

Return Values
> Number of characters received (integer).
>
> If the port cannot receive characters, the following negative values are returned to report the current port status:
>
> -1   Port is open but communication has not been established
> -2   Port is used by another task
> -3   Port is not open

See Also
> CloseNet, OpenNet, Read, Write

ChkNet Example

```
Integer numChars

numChars = ChkNet(201)
```

Chr$ Function

Returns the character specified by a numeric ASCII value.

$\boxed{\textbf{F}}$

Syntax

    Chr$*(number)*

Parameters

    number   An integer expression between 1 and 255.

Return Values

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

Description

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

See Also

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

Chr$ Function Example

The following example declares a variable of type String and then assigns the string "ABC" to it. The Chr$ instruction is used to convert the numeric <u>ASCII</u> 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.

S

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
>
```

CloseCom Statement

Close the RS-232C port previously opened with OpenCom.

$\boxed{\textbf{S}}$

Syntax
> **CloseCom** #*portNum* | **All**

Parameters
> *portNum*   Integer expression for port number to close.
> The task will close all the open RS-232C port when All is specified.

See Also
> ChkCom, OpenCom

CloseCom Statement Example

```
CloseCom #1
```

CloseNet Statement

Close the TCP/IP port previously opened with OpenNet.

$\boxed{\textsf{S}}$

Syntax
    CloseNet *#portNumber*

Parameters
    portNumber   Integer expression for port number to close. Range is 201 - 208.

                         The task will close all the open TCP/IP port when All is specified.

See Also
    ChkNet, OpenNet

CloseNet Statement Example

```
CloseNet #201
```

Cls Statement

Clears the EPSON RC+ 5.0 Run, Operator, or Command window text area.

$\boxed{\textbf{S}}$

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

Parameters
    deviceID   21 RC+
                         23 OP
                         24 TP
                         When *deviceID* is omitted, the display device is cleared.

Description
    Cls clears either 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 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
```

Cont Statement

Resumes the contoller 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

This command can be executed only when it is called from the event handler function. It cannot be executed from the normal command or command line.

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.

⚠ 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

Cont Example

```
Function frmmain_btnCont_Click(Sender$ As String)
    Cont
Fend
```

Cos Function

Returns the cosine of a numeric expression.

$\boxed{\textbf{F}}$

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.

$\boxed{\textbf{S}}$

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, 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.
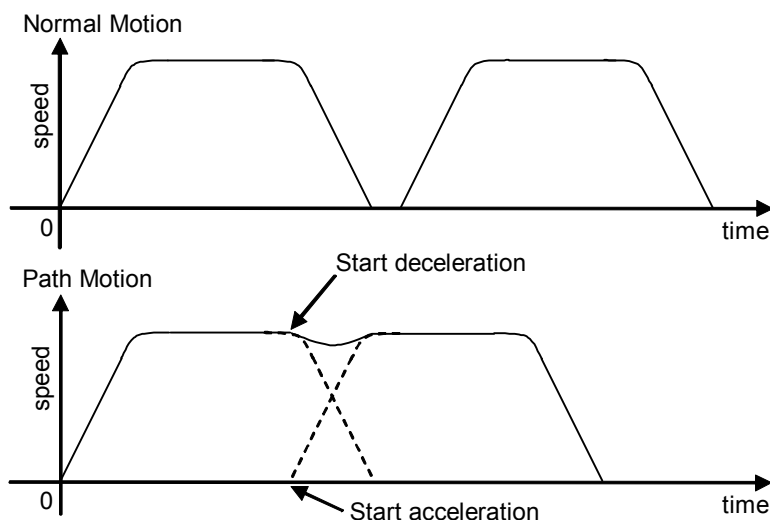
   When CP is On, path motion will continue without full deceleration between two CP motions (Arc, Arc3,
   Jump3, Jump3CP, Move), or two PTP motions (Go, Jump).
   In contrast, full deceleration will occur between a CP motion and a PTP motion.

   CP will be set to Off in the following cases
       Controller startup
       Reset
       All task stop
       Switching the Auto / Programming operation mode
       Motor On
       SFree, SLock

See Also

   CP Function, Arc, Move, Go

CP Statement Example

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

CP Function

Returns status of path motion.

Syntax
    CP

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

See Also
CP Statement

CP Function Example

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

Ctr Function

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

  High (ON)

  | 4 msec or longer |

  Low (OFF)
                    4 msec or longer
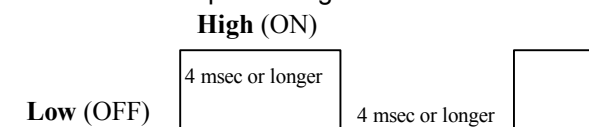
See Also
  CTReset

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

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

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

CTReset Statement

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

> | S

Syntax
    CTReset(*bitNumber*)

Parameters

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

Description

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

Notes

Turning Off Power and Its Effect on Counters

    Turning off main power releases all counters.

Using the Ctr Function

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

See Also

    Ctr

CTReset Example

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

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

CtrlDev Function

Returns the current control device number.

F

Syntax

   CtrlDev

Return Values

21	**PC**
22	**Remote I/O**
23	**OP**
26	**Remote Ethernet**
29	**Remote RS232**

See Also

   CtrlInfo Function

CtrlDev Function Example

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

CtrlInfo Function

Returns controller information.

F

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	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
	13-31		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 23 - OP
6	N/A		Undefined
7	N/A		Operation mode: 0 - Program mode 1 - Auto mode
8	N/A		1 - Motors off 0 - Hold (motor setting is on, but currently held off)
9	N/A		Firmware verision of the Controller Major No.*1000000 + Minor No.*10000 + Rev No.*100 + Build No. (Example) Version 1.6.2.4 is 1060204

Return Values
    Long value of the desired data

See Also
    CtrlInfo$, RobotInfo, TaskInfo

CtrlInfo Function Example
```
Print "The controller version: ", CtrlInfo(0)
```

CurPos Function

Returns the current target position of the specified robot.

$\boxed{\textbf{F}}$

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.

S

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.

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

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

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

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

> 2 - Generate a curve in the XY plane with no Z Axis movement or U Axis rotation.
>
> 3 - Generate a curve in the XYZ space with no U axis rotation.
>
> 4 - Generate a curve in the XYZ space with U-Axis rotation.
>
> 6 - Generate a curve in the XYZ space with U, V, and W axes rotation (6-Axis robots only).

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

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

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

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

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

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

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

Description

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

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

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

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

Note

Use tangential correction when possible

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

Open Curve Min and Max Number of Points Allowed

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

Closed Curve Min and Max Number of Points Allowed

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

Potential Errors

Attempt to Move Arm Outside Work Envelope

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

See Also

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

Curve Statement Example

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

CVMove Statement

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

$\boxed{\textbf{S}}$

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

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

Description
CVMove performs the continuous spline path motion defined by the data in the file *fileName*, which is located 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 there is no file name extension, then CVT is assumed.

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

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

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

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 Statements

Sets the coordinate of a point.

| F |

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

Parameters

>   *point*   **P***number* or **P**(*expr*) or point label.

>   *value*   Real expression representing the new coordinate value in millimeters.

See Also

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

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

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

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

Retrieves a coordinate value from a point

F

Syntax

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

Parameters

    point   Point expression.

Return Values

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

Description

Used to retrieve an individual coordinate value from a point.

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

See Also

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

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

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

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

Date Statement

Specifies and displays the current date in the controller.

> **>**

Syntax

> **Date** *yyyy, mm, dd*
> **Date**

Parameters

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

Return Values

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

Description

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

See Also

> Time, Date$

Date Example

> The following examples are done from the command window.

```
> Date
2006/09/27

> Date 2006,10,1

> Date
2006/10/01
```

Date$ Function

Returns the system date.

F

Syntax
  Date$

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

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

See Also
  Date, Time, Time$

Date$ Function Example

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

DegToRad Function

Converts degrees to radians.

$$\boxed{\textbf{>}} \quad \boxed{\textbf{F}}$$

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))
```

DispDev Statement

Sets the current display device.

S

Syntax
   DispDev (*deviceID***)**

Parameters
   deviceID   The device ID for the desired display device.

                          21 RC+
                          23 OP
                          24 TP

See Also
   DispDev Function

DispDev Statement Example

```
DispDev 23
```

DispDev Function

Returns the current display device.

F

Syntax
  DispDev

Return Values
  Integer value containing the deviceID.
  21 RC+
  23 OP
  24 TP

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).

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.

$\boxed{\textbf{S}}$

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

Or, you can use this syntax:

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

The Do Loop statement syntax has these parts:

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

Description

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

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

See Also

For...Next, Select...Send

Do 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).

$\boxed{\textbf{S}}$

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 to the upper bound value.

                The total available number of array elements for local and global preserve variables is 1000.

                The total available number of array elements for global and module variables is 10000.

                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

    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, Integer, Long, Real, String

Double Example

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

```
Function doubletest
  Double var1
  Double A(10)         'Single dimension array of double
  Double B(10, 10)     'Two dimension array of double
  Double C(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).

$\boxed{>}$ $\boxed{S}$

Syntax
   (1) ECP   *ECPNumber*
   (2) ECP

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

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

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

Note

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

Power Off and Its Effect on the ECP Selection
   Turning main power off clears the ECP selection.

See Also
   ECPSet

ECP Statement Example

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

ECP Function

Returns the current ECP (external control point) number.

Syntax

    ECP

Return Values

Integer containing the current ECP number.

Note

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

See Also

ECP Statement

ECP Function Example

```
Integer savECP

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

ECPClr Statement

Clears (undefines) an external control point.

> **>** **S**

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
    Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

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

See Also
    Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLSet

ECPClr Example

```
ECPClr 1
```

ECPDef Function

Returns ECP definition status.

$>$ F

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 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.

> S

Syntax

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

Parameters

    ECPNum   Integer number from 1-15 representing which of 15 external control points to define.

    ECPPoint   **P***number* or **P**(*expr*) or point label or point expression.

Return Values

    When parameters are omitted, displays the current **ECPSet** definitions.
    When only the **ECP** number is specified, displays the specified **ECPSet** definitions.

Description

    Defines an external control point.

    Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

Note

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

ECPSet Example

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

ECPSet Function

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

[F]

Syntax
   ECPSet(*ECPNumber*)

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

Return Values
   A point containing the ECP definition.

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

See Also
   ECPSet Statement

ECPSet Function Example

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

Elbow Statement

Sets the elbow orientation of a point.

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

Parameters

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

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

See Also
Elbow Function, Hand, J4Flag, J6Flag, Wrist

Elbow Statement Example

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

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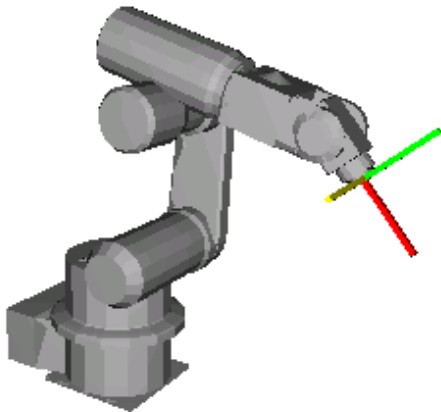


```
Elbow P1, Above
Go P1
```
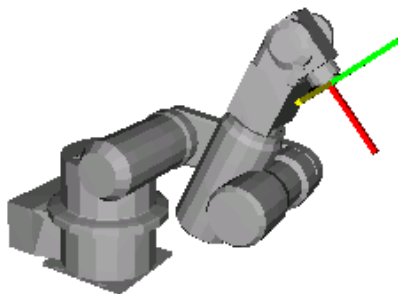


```
Elbow P1, Below
Go P1
```

Elbow Function

Returns the elbow orientation of a point.

F

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)
```

Era Function

Returns the joint number for which an error occurred.

$\boxed{\text{F}}$

Syntax

  Era[(*taskNum*)]

Parameters

  taskNum   Integer expression representing a task number from 0-16.
                         Task number omission or 0 specifies the current task.

Return Values

  The joint number that caused the error in the range 0-6 as described below:

         0 - The current error was not caused by a servo axis.
         1 - The error was caused by joint number 1
         2 - The error was caused by joint number 2
         3 - The error was caused by joint number 3
         4 - The error was caused by joint number 4
         5 - The error was caused by joint number 5
         6 - The error was caused by joint number 6

Description

  Era is used when an error occurs to determine if the error was caused by one of the robot joints and to return the number of the joint which caused the error. If the current error was not caused by any joint, **Era** returns zero.

See Also

  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.

S

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 0-16.
                           Task number omission or 0 specifies the current task.

Return Values
    The name of thefunction 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.

See Also
    Era, Erl, Err, ErrMsg$, Ert, OnErr

Erf$ Function Example
The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erf$: the name of the function the error occurred in; Erl: the line number where the error occurred; Era: if a joint caused the error....

```
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 0-16.
                                 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.

See Also
   Era, Erf$, Err, ErrMsg$, Ert, OnErr

Erl Function Example
   The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erl: where the error occurred; Era: if a joint caused the error....

```
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.

$\boxed{\textbf{F}}$

Syntax

    Err [(*taskNumber*)]

Parameters

    taskNumber   Optional. Integer expression representing a task number from 0-16. 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.

See Also

    Era, Erf$, Erl, ErrMsg$, EResume, Ert, OnErr, Return, SysErr

Err Example

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

```
Function errtest
    Integer i, errnum
    Real x

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

ErrMsg$ Function

Returns the error message which corresponds to the specified error number.

$\boxed{\textbf{F}}$

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

   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$ Example
   The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erl: where the error occurred; Era: if a joint caused the error....

```
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
```

ErrorOn Funcion

Returns the error status of the controller.

F

Syntax
ErrorOn

Return Values
True if the controller is in error status, otherwise False.

DeThis scription
ErrorOn function is used only for NoEmgAbort task (special task using NoEmgAbort at Xqt).

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 Occurence
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
```

Error Statement

Generates a user error.

Syntax
   (1) Error *task Number, errorNumber*
   (2) Error *errorNumber*

Parameters

taskNumber	Optional. Integer expression representing a task number. Range from 0 to 16. 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+ 5.0 development environment.

See Also
   Era, Erl, Err, OnErr

Error Statement Example

```
#define ER_VAC 8000

If Sw(vacuum) = Off Then
    Error ER_VAC
EndIf
```

Ert Function

Returns the task number in which an error occurred.

$\boxed{>}$ $\boxed{\textbf{F}}$

Syntax

    Ert

Return Values

    The task number in which the error occurred.

Description

    Ert is used when an error occurs to determine in which task the error occurs. The number returned will be between 1and 16.

See Also

    Era, Erl, Err, ErrMsg$, OnErr, Trap

Ert Function Example

    The following example shows a simple program using the Ert function to determine which task the error occurred in along with; Erl: where the error occurred; Err: what error occurred; Era: if a joint caused the error....

```
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
```

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

EStopOn Function

Return 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 Safety Door Open when designing the system.

Error Handling

As this program, finish the task promply 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 [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
```

Exit Statement

Exits a loop construct or function.

S

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
```

Find Statement

Specifies or displays the condition to store coordinates during motion.

> **>**   **S**

Syntax

   Find [*inputCondition*]

Parameters

   inputCondition   The following functions and operators are available.

                     Functions : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemW, Ctr

                     Operators : 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.

See Also

Sw, In, InW, Oport, Out, FindPos, Go, Jump, PosFound

Find Statement Example

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
EndIf
```

FindPos Function

Returns a robot point stored by Fine during a motion command.

$$\boxed{\text{F}}$$

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.

> **>** **S**

Syntax

(1) **Fine** *axis1, axis2, axis3, axis4,* [*axis5*], [*axis6*]
(2) **Fine**

Parameters

axis1	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 1st joint.
axis2	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 2nd joint.
axis3	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 3rd joint.
axis4	Integer expression ranging from (0-65535) which represents the allowable positioning error for the 4th joint.
axis5	Optional. Integer expression ranging from (0-65535) which represents the allowable positioning error for the 5th joint.
axis6	Optional. Integer expression ranging from (0-65535) which represents the allowable positioning error for the 6th joint.

Return Values

When used without parameters, **Fine** displays the current fine values for each of the 4 or 6 axes, depending on the robot type.

Description

Fine specifies, for each joint, the allowable positioning error for detecting completion of any given move.

This positioning completion check begins after the CPU has completed sending the target position pulse to the servo system. Due to servo delay, the robot will not yet have reached the target position. This check continues to be executed every few milliseconds until each joint has arrived within the specified range setting. Positioning is considered complete when all axes have arrived within the specified ranges. Once positioning is complete program control is passed to the next 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 robot manual for details.

Notes

Cycle Times and the Fine Instruction

The **Fine** value does not affect the acceleration or deceleration control of the manipulator arm. However, smaller **Fine** values can cause the system to run slower because it may take the servo system extra time (a few milliseconds) to get within the acceptable position range. Once the arm is located within the acceptable position range (defined by the **Fine** instruction), the CPU executes the next user instruction. (Keep in mind that all activated axes must be in position before the CPU can execute the next user instruction.)

Initialization of Fine (by Motor On, SLock, SFree)

Any time the following commands are used the Fine value is initialized to default values: SLock, SFree, Motor instructions.

Make sure that you reset Fine values after one of the above commands execute.

Potential Errors

If **Fine** positioning is not completed within about 2 seconds, Error 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

Fine Statement Example

The examples below show the Fine statement used in a program function, and used from the monitor window.

```
Function finetest
    Fine 5, 5, 5, 5            'reduce precision to +/- 1 Pulse
    Go P1
    Go P2
Fend

> Fine 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.

Return Values
> Real value.

See Also
> Accel, AccelS, Arc, Go, Jump, Move, Speed, SpeedS, Pulse

Fine Function Example
> This example uses the **Fine** function in a program:

```
Function finetst
    Integer a
    a = Fine(1)
Fend
```

Fix Function

Returns the integer portion of a real number.

$\boxed{\textbf{F}}$

Syntax
   Fix(*number*)

Parameters
   number   Real expression containing number to fix.

Return Values
   An integer value containing the integer portion of the real number.

See Also
   Int

Fix Function Example

```
>print Fix(1.123)
 1
>
```

FmtStr$ Function

Format a numeric expression.

F

Syntax
    FmtStr$ (*numeric expression, strFormat*)

Parameters
numeric expression	Numaric expression to be formatted.
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

Fromat a numeric expression.

Character	Description
None	Display the number with no formatting.
(0)	Digit placeholder. Display a digit or a zero. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification.
(#)	Digit placeholder. Display a digit or nothing. If the expression has a digit in the position where the # appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression.
(.)	Decimal placeholder. In some locales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the Number Format recognized by your system.
(,)	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.

See Also
  Left$, Right$, Str$

FmtStr$ Example

```
Function SendDateCode

    String d$, f$

    f$ = FmtStr$(10, "000.00")
    OpenCom #1
    Print #1, f$
    CloseCom #1
Fend
```

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.

<div style="float:right;border:1px solid black;padding:4px;">**S**</div>

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.

See Also

Do...Loop

For...Next 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
```

Function...Fend Statement

A function is a group of program statements which includes a **Function** statement as the first statement and a Fend statement as the last statement.

Syntax

    Function *funcName* [(*argList*)] [**As** *type*]
       statements
    Fend

Parameters

funcName	The name which is given to the specific group of statements bound between the **Function** and **Fend** instructions. The function name must contain alphanumeric characters and may be up to 32 characters in length. Underscores are also allowed.
argList	Optional. List of variables representing arguments that are passed to the Function procedure when it is called. Multiple variables are separated by commas.

The arglist argument has the following syntax:

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

ByRef	Optional. Specify ByRef when you want any changes in the value of the variable to be seen by the calling function.
ByVal	Optional. Specify ByVal when you do not want any changes in the value of the variable to be seen by the calling function. This is the default.
varName	Required. Name of the variable representing the argument; follows standard variable naming conventions.
As *type*	Required. You must declare the type of argument.

Return Values

    Value whose data type is specified with the **As** clause at the end of the function declaration.

Description

The **Function** statement indicates the beginning of a group of SPEL[+] statements. To indicate where a function ends we use the **Fend** statement. All statements located between the **Function** and **Fend** statements are considered part of the function.

The **Function...Fend** combination of statements could be thought of as a container where all the statements located between the **Function** and **Fend** statements belong to that function. Multiple functions may exist in one program file.

See Also

    Call, Fend, Halt, Quit, Return, Xqt

Function...Fend Example

The following example shows 3 functions which are within a single file. The functions called task2 and task3 are executed as background tasks while the main task called main executes in the foreground.

```
Function main
  Xqt 2, task2 'Execute task2 in background
  Xqt 3, task3 'Execute task3 in background
  '....more statements here
Fend

Function task2
  Do
    On 1
    On 2
    Off 1
    Off 2
  Loop
Fend

Function task3
  Do
    On 10
    Wait 1
    Off 10
  Loop
Fend
```

Global Statement

Declares variables with the global scope. Global variables can be accessed from anywhere. | **S** |

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, Integer, Long, Real, Double, Byte, or String.
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 preserve variables is 100 for strings and 1000 for all other types.

The total available number of array elements for global variables is 1000 for strings and 10000 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 VB Guide option.

It is recommended that global variable names begin with a "g_" prefix to make it easy to recognize globals in a program. For example:

    Global Long g_PartsCount

See Also
Boolean, Byte, Double, Integer, Long, Real, String

Global Statement Example

The following example shows 2 separate program files. The first program file defines some global variables and initializes them. The second file then uses these global variables.

FILE1 (MAIN.PRG)
```
Global Integer status1
Global Real numsts

Function Main
    Integer I

    status1 = 10
```
The following example shows 2 separate program files. The first program file defines some global variables and initializes them. The second file then also uses these global variables.

FILE1 (MAIN.PRG)

```
Global Integer g_Status
Global Real g_MaxValue

Function Main

    g_Status = 10
    g_MaxValue = 1.1
    .
    .
Fend
```

FILE2 (TEST.PRG)

```
Function Test

    Print "status1 = , g_Status
    Print "MaxValue = , g_MaxValue
    .
    .
Fend
```

Go Statement

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.

> **>** **S**

Syntax
Go destination [**CP**] [LJM [*orientationFlag*]] [*searchExpr*] [!...!]

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.

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 the 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 and RS series robots.
When using *orientationFlag* with the default value, it can be ommited.

```
        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 where as the Move instruction causes the arm to move in a straight line. The **Go** instruction is used when the user is primarily concerned with the orientation of the arm when it arrives on point. The Move instruction is used when it is important to control the path of the robot arm while it is moving.

Difference between Go and Jump

The Jump instruction and the **Go** instruction each cause the robot arm to move in a point to point type fashion. However, the JUMP instruction has 1 additional feature. Jump causes the robot end effector to first move up to the LimZ value, then in a horizontal direction until it is above the target point, and then finally down to the target point. This allows Jump to be used to guarantee object avoidance and more importantly to improve cycle times for pick and place motions.

Proper Speed and Acceleration Instructions with Go

The Speed and Accel instructions are used to specify the speed and acceleration of the manipulator during motion caused by the **Go** instruction. Pay close attention to the fact that the Speed and Accel instructions apply to point to point type motion (like that for the **Go** instruction) while linear and circular interpolation motion uses the SpeedS and AccelS instructions.

Using Go with the Optional Till Modifier

The optional Till modifier allows the user to specify a condition to cause the robot to decelerate to a stop at an intermediate position prior to completing the motion caused by the **Go** instruction. If the Till condition is not satisfied, the robot travels to the target position. The Go with Till modifier can be used in 2 ways as described below:

(1) Go with Till Modifier

Checks if the current Till condition becomes satisfied. If satisfied, this command completes by decelerating and stopping the robot at an intermediate position prior to completing the motion caused by the **Go** instruction.

(2) Go with Till Modifier, Sw(Input bit number) Modifier, and Input Condition

This version of the Go with Till modifier allows the user to specify the Till condition on the same line with the Go instruction rather than using the current definition previously defined for Till. The condition specified is simply a check against one of the inputs. This is accomplished through using the Sw instruction. The user can check if the input is On or Off and cause the arm to stop based on the condition specified. This feature works almost like an interrupt where the motion is interrupted (stopped) once the Input condition is met. If the input condition is never met during the robot motion then the arm successfully arrives on the point specified by *destination.*

Using Go with the Optional Find Modifier

The optional Find modifier allows the user to specify a condition to cause the robot to record a position during the motion caused by the **Go** instruction. The Go with Find modifier can be used in 2 ways as described below:

(1) Go with Find Modifier:

Checks if the current Find condition becomes satisfied. If satisfied, the current position is stored in the special point FindPos.

(2) Go with Find Modifier, Sw(Input bit number) Modifier, and Input Condition:

This version of the Go with Find modifier allows the user to specify the Find condition on the same line with the Go instruction rather than using the current definition previously defined for Find. The condition specified is simply a check against one of the inputs. This is accomplished through using the Sw instruction. The user can check if the input is On or Off and cause the current position to be stored in the special point **FindPos**.

Go Instruction Always Decelerates to a Stop

The **Go** instruction always causes the arm to decelerate to a stop prior to reaching the final destination of the move.

Potential Errors

Attempt to Move Outside of Robots Work Envelope

When using explicit coordinates with the **Go** instruction, you must make sure that the coordinates defined are within the robots valid work envelope. Any attempt to move the robot outside of the valid work envelope will result in an error.

See Also

!...! Parallel Processing, Accel, Find, Jump, Move, Pass, Pn=   (Point Assignment), Pulse, Speed, Sw, Till

Go Example

The example shown below shows a simple point to point move between points P0 and P1 and then moves back to P0 in a straight line. Later in the program the arm moves in a straight line toward point P2 until input #2 turns on. If input #2 turns On during the Move, then the arm decelerates to a stop prior to arriving on point P2 and the next program instruction is executed.

Function sample

```
Integer i

Home
Go P0
Go P1
For i = 1 to 10
  Go P(i)
Next i
Go P2 Till Sw(2) = On
If Sw(2) = On Then
  Print "Input #2 came on during the move and"
  Print "the robot stopped prior to arriving on"
  Print "point P2."
Else
  Print "The move to P2 completed successfully."
  Print "Input #2 never came on during the move."
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

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.

$\boxed{\textbf{S}}$

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 stand alone Return instruction has no meaning because the system doesn't know where to Return to.

See Also

    GoTo, OnErr, Return

GoSub Statement Example

The following example shows a simple function which uses a GoSub instruction to branch to a label and execute some I/O instructions then return.

```
Function main
    Integer var1, var2

    GoSub checkio  'GoSub using Label
    On 1
    On 2
    Exit Function

checkio:    'Subroutine starts here
    var1 = In(0)
    var2 = In(1)
    If var1 = 1 And var2 = 1 Then
        On 1
    Else
        Off 1
    EndIf
    Return  'Subroutine ends here
Fend
```

GoTo Statement

The **GoTo** instruction causes program control to branch unconditionally to a designated statement label.

$\boxed{\textsf{S}}$

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.

Notes

Using Too Many GoTo's
Please be careful with the **GoTo** instruction since using too many **GoTo**'s in a program can make the program difficult to understand. The general rule is to try to use as few **GoTo** instructions as possible. Some **GoTo**'s are almost always necessary. However, jumping all over the source code through using too many **GoTo** statements is an easy way to cause problems.

See Also
GoSub, OnErr

GoTo Statement Example
The following example shows a simple function which uses a GoTo instruction to branch to a line label.

```
Function main

    If Sw(1) = Off Then
        GoTo mainAbort
    EndIf
    Print "Input 1 was On, continuing cycle"
    .
    .
    Exit Function

mainAbort:
    Print "Input 1 was OFF, cycle aborted!"
Fend
```

Halt Statement

Temporarily suspends execution of a specified task.

$\boxed{\textbf{S}}$

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 pauses the task when the specified task is NoPause task or NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt).

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 orientation of a point.

> **S**

Syntax
> **(1) Hand** *point*, [*Lefty* | *Righty*]
> **(2) Hand**

Parameters
> *point*   Pnumber or P(expr) or point label.
> *Lefty | Righty*   Hand orientation.

Return Values
> When both parameters are omitted, the hand orientation is displayed for the current robot position.
> If *Lefty | Righty* is ommited, the hand orientation for the specified point is displayed.

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
```
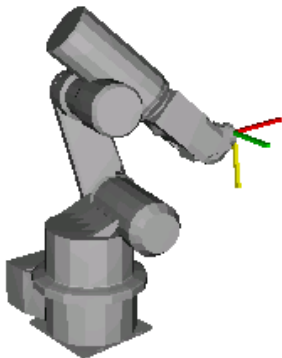


```
Hand P1, Righty
Go P1
```
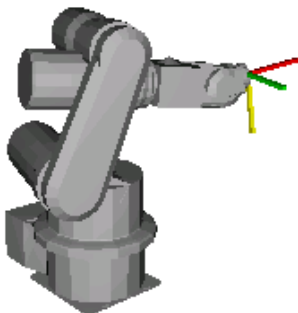


```
Hand P1, Lefty
Go P1
```

Hand Function

Returns the hand orientation of a point.

Syntax

> **Hand** [(*point*)]

Parameters

> *point*   Optional. Point expression. If *point* is omitted, then the hand orientation of the current robot position is returned.

Return Values

> 1   Righty (/R)
> 2   Lefty (/L)

See Also

> Elbow, Wrist, J4Flag, J6Flag, J1Flag, J2Flag

Hand Function Example

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

Here Statement

Teach a robot point at the current position.

S

Syntax
 Here *point*

Parameters
 point   **P***number* or **P**(*expr*) or point label.

See Also
 Here Function

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.

> **F**

Syntax
    Hex$*(number)*

Parameters
    number   Integer expression.

Return Values
    Returns a string containing the ASCII representation of the number in hexadecimal format.

Description
    Hex$** returns a string representing the specified number in hexadecimal format. Each character is from 0-9 or A-F. **Hex$ is especially useful for examining the results of the Stat function.

See Also
    Str$, Stat, Val

Hex$ Function Example

```
> print hex$(stat(0))
A00000
> print hex$(255)
FF
```

Home Statement

Moves the robot arm to the user defined home position.

> **S**

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 Errors

Attempting to Home without HomeSet Values Defined
> Attempting to **Home** the robot without setting the HomeSet values will result in an Error 143 being issued.

See Also
> HomeClr, HomeDef, HomeSet, Hordr

Home Example
> The Home instruction can be used in a program such as this:

```
Function InitRobot
   Reset
   If Motor = Off Then
      Motor On
   EndIf
   Home
Fend
```

Or it can be issued from the Command window like this:

```
> home
>
```

HomeClr Function

Clears the home position definition.

[F]

Syntax
    HomeClr

Description
Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

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.

<div style="float:right; border:1px solid black; padding:4px;">**F**</div>

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.

$\boxed{>}$ $\boxed{S}$

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

Parameters

j1Pulses	The home position encoder pulse value for joint 1.
j2Pulses	The home position encoder pulse value for joint 2.
j3Pulses	The home position encoder pulse value for joint 3.
j4Pulses	The home position encoder pulse value for joint 4.
j5Pulses	Optional for 6-axis robots. The home position encoder pulse value for joint 5.
j6Pulses	Optional for 6-axis robots. The home position encoder pulse value for joint 6.

Return Values
Displays the pulse values defined for the current Home position when parameters are omitted.

Description
Allows the user to define a new home (standby) position by specifying the encoder pulse values for each of the robot joints.

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

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 Example
The following examples are done from the command 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.

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.

> S

Syntax

(1) **Hordr** *step1, step2, step3, step4*, [*step5*] ,[*step6*]

(2) **Hordr**

Parameters

step1   Bit pattern that defines which joints should home during the 1st step of the homing process.

step2   Bit pattern that defines which joints should home during the 2nd step of the homing process.

step3   Bit pattern that defines which joints should home during the 3rd step of the homing process.

step4   Bit pattern that defines which joints should home during the 4th step of the homing process.

step5   For 6 axis robots. Bit pattern that defines which joints should home during the 5th step of the homing process.

step6   For 6 axis robots. Bit pattern that defines which joints should home during the 6th step of the homing process.

Return Values

Displays current Home Order settings when parameters are omitted.

Description

Hordr specifies joint motion order for the Home command. (i.e. Defines which joint will home 1st, which joint will home 2nd, 3rd, etc.)

The purpose of the **Hordr** instruction is to allow the user to change the homing order. The homing order is broken into 4 or 6 separate steps, depending on robot type. The user then uses **Hordr** to define the specific joints which will move to the Home position during each step. It is important to realize that more than one joint can be defined to move to the Home position during a single step. This means that all joints can potentially be homed at the same time. For 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
Bit Number:	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5
Binary Code:	&B0001	&B0010	&B0100	&B1000	&B10000	&B100000

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

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.

$\boxed{\textbf{>}}$ $\boxed{\textbf{F}}$

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 Example

    The following example is done from the Command window:

```
> hour
2560
>
```

Hour Function

Returns the accumulated controller operating time.

$\boxed{\textbf{F}}$

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.

<div style="float:right;border:1px solid black;padding:4px">S</div>

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.

The logical output of the conditional statement is any number excluding 1 when it is True, and 0 when it is false.

Notes

Sample Conditions:

a = b	:a is equal to b
a < b	:b is larger than a
a >= b	:a is greater than or equal to b
a <> b	:a is not equal to b
a > b	:b is smaller than a
a <= b	:a is less than or equal to b

Logical operations And, Or and Xor may also be used.

See Also

Else, Select...Case, 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"
```

In Function

Returns the status of the specified Byte port. Each port contains 8 input channels.

F

Syntax

   In(byte*portNumber*)

Parameters

   byteportNumber   Integer number representing one eight bit port (one byte).

Return Values

Returns an integer value between 0-255. The return value is 8 bits, with each bit corresponding to 1 input channel.

Description

In provides the ability to look at the value of 8 input channels at the same time. The In instruction can be used to store the 8 I/O channels status into a variable or it can be used with the Wait instruction to Wait until a specific condition which involves more than 1 I/O channel is met.

Since 8 channels are checked at a time, the return values range from 0-255. Please review the chart below to see how the integer return values correspond to individual input channels.

Input Channel Result (Using Byte port #0)

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

Input Channel Result (Using Byte port #2)

Return Value	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 lets 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)

F

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 (0-99).

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 0-7, _portNumber_ = 1 means inputs 8-15, etc.

    The resulting value of the 8 inputs is returned in BCD format. The return value may have 1 or 2 digits between 0 and 99. The 1st digit (or 10's digit) corresponds to the upper 4 outputs of the group of 8 outputs selected by _portNumber_. The 2nd digit (or 1's digit) corresponds to the lower 4 outputs of the group of 8 outputs selected by _portNumber_.

    Since valid entries in BCD format range from 0-9 for each digit, every I/O combination cannot be met. The table below shows some of the possible I/O combinations and their associated return values assuming that _portNumber_ is 0.

Input Settings (Input number)

Return Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

Notice that the Binary Coded Decimal format only allows decimal values to be specified. This means that through using Binary Coded Decimal format it is impossible to retrieve a valid value if all inputs for a specific port are turned on at the same time when using the **InBCD** instruction. The largest value possible to be returned by **InBCD** is 99. In the table above it is easy to see that when 99 is the return value for InBCD, all inputs are not on. In the case of a return value of 99, inputs 0, 3, 4, and 7 are On and all the others are Off.

InBCD function cannot be used for the Wait command or wait condition of Till, Find, Sense.

Notes

Difference between InBCD and In

The **InBCD** and In instructions are very similar in the SPEL$^+$ language. However, there is one major difference between the two. This difference is shown below:

- The **InBCD** instruction uses the Binary Coded Decimal format for specifying the return value format for the 8 inputs. Since Binary Coded Decimal format precludes the values of &HA, &HB, &HC, &HD, &HE or &HF from being used, all combinations for the 8 inputs cannot be satisfied.
- The In instruction works very similarly to the InBCD instruction except that In allows the return value for all 8 inputs to be used. (i.e. 0-255 vs. 0-99 for InBCD) This allows all possible combinations for the 8 bit input groups to be read.

See Also

In, MemOff, MemOn, MemOut, MemSw, Off, On, OpBCD, Oport, Out, Sw, Wait

InBCD Example

Some simple examples from the 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. Real expression that specifies total moment of inertia in kgm^2 around the center of the end effector joint, including end effector and part.

eccentricity Optional. Real expression that specifies eccentricity in mm around the center of the end effector joint, including end effector and part.

Return Values

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

Description

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.

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also

Inertia Function

Inertia Statement Example

```
Inertia 0.02, 1
```

Inertia Function

Returns inertia parameter value.

$\boxed{\textbf{F}}$

Syntax
    Inertia(*paramNumber***)**

Parameters
   paramNumber   Integer expression which can have the following values:

                       0: Causes function to return 1 if robot supports inertia parameters or 0 if not.

                       1: Causes function to return load inertia in kgm2.

                       2: Causes function to return eccentricity in mm.

Return Values
    Real value of the specified setting.

See Also
    Inertia Statement

Inertia Function Example

```
Real loadInertia, eccentricity

loadInertia = Inertia(1)
eccentricity = Inertia(2)
```

InPos Function

Returns the position status of the specified robot.

$\boxed{\textbf{F}}$

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

Allows numeric data to be received from the keyboard and stored in a variable(s).

> S

Syntax
   Input *varName* [*, varName, varName,...*]

Parameters
   varName   Variable name. Multiple variables can be used with the **Input** command as long as they are separated by commas.

Description
   Input receives numeric data from the 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 Errors

Number of variables and input data differ
   For multiple variables, the number of input data must match the number of Input variable names. When the number of the variables specified in the instruction is different from the number of numeric data received from the keyboard, an Error 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, -99
-10000
25.1 -99
B = 25.1 C = -99
>
```

Input # Statement

Allows string or numeric data to be received from a communications port or a device and stored in a variable(s).

> **>**   **S**

Syntax

Input *PortNum, varName* [*, varName, varName,...*]

Parameters

PortNum   The communications handle or the device ID. Communication handles can be specified in OpenCom (RS232) and OpenNet (TCP/IP) statements.

       Device ID integers are as follows.
       21 RC+
       23 OP
       24 TP

varName   Variable name to receive the data.

Description

The **Input #** instruction receives numeric or string data from the device specified by *PortNum*, and assigns the data to the variable(s).

Notes

Rules for Numeric Input

When inputting numeric values and non-numeric data is found in the input other than the delimiter (comma), the **Input** instruction discards the non-numeric data and all data following that non-numeric data.

Rules for String Input

When inputting strings, numeric and alpha characters are permitted as data.

Other Rules for the Input Instruction

- When more than one variable is specified in the instruction, the numeric data input intended for each variable has to be separated by a comma (",") character.
- Numeric variable names and string variable names are allowed. However, the input data type must match the variable type.

Potential Errors

Number of variables and input data differ

When the number of the variables specified in the instruction is different from the number of numeric data received from the device, an Error 2505 will occur.

See Also

Input, Line Input, Line Input #

Input # Statement Example

This is a simple program example using Input # statement.

```
Function GetData
    Integer A
    String B$

    OpenCom #1
    Print #1, "Send"
    Input #1, A    'Accept numeric data from port #1
    Input #1, B$   'Get string data from port #1
    CloseCom #1
Fend
```

InReal Function

Returns the input data of 2 words (32 bits) as the floating-point data (IEEE754 compliant) of 32 bits.

F

Syntax

InReal(*WordPortNumber*)

Parameter

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(0)
```

InsideBox Function

Returns the check status of the approach check area.

F

Syntax
   InsideBox(*AreaNum*)

Parameters
   AreaNum   Integer expression from 1 to 15 representing which approach check area to
                     return status for.

Return Values
   True if the robot end effector approaches the specified approach check area, otherwise False.

See Also
   Box, BoxClr, BoxDef, InsidePlane

InsideBox Function Example

   This is an example to start up different task from Main function and display the approach check status.

```
Function Main
      :
      :
   Xqt PrintInsideBox
      :
      :
Fend

Function PrintInsideBox
   Do
      Wait InsideBox(1) = True
      Print "Inside Box1"
      Wait InsideBox(1) = False
      Print "Outside Box1"
   Loop
Fend
```

   This is a program example to parallel process the motion command. I/O turns ON when the robot approaches the specific approach check area at running.

```
Function Main
   Motor On
   Power High
   Speed 30; Accel 30, 30

   Go P1 !D0; Wait InsideBox(1) = True; On 1!

Fend
```

Notes

Do not exclude D0 in this program.

InsidePlane Function

Returns the check status of the approach check plane.

$\boxed{\text{F}}$

Syntax

   InsidePlane(*PlaneNum*)

Parameters

   PlaneNum   Integer expression from 1 to 15 representing which approach check plane to return status for.

Return Values

   True if the robot end effector approaches the specified approach check plane, otherwise False.

See Also

   InsideBox, Plane, PlaneClr, PlaneDef

InsidePlane Function Example

   This is an example to start up different task from Main function and display the approach check status.

```
Function Main
      :
      :
   Xqt PrintInsidePlane
      :
      :
Fend

Function PrintInsidePlane
   Do
      Wait InsidePlane(1) = True
      Print "Inside Plane1"
      Wait InsidePlane(1) = False
      Print "Outside Plane1"
   Loop
Fend
```

   This is a program example to parallel process the motion command. I/O turns ON when the robot comes through the specific approach check plane at running.

```
Function Main
   Motor On
   Power High
   Speed 30; Accel 30, 30

   Go P1 !D0; Wait InsidePlane(1) = True; On 1!

Fend
```

Notes

Do not exclude D0 in this program.

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.

F

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
>
```

Integer Statement

Declares variables of type Integer. (2 byte whole number).

S

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 to the upper bound value.

> The total available number of array elements for local and global preserve variables is 1000.

> The total available number of array elements for global and module variables is 10000.

> 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

Integer is used to declare variables as type integer. Variables of type integer can contain whole numbers with values from -32768 to 32767. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

See Also

Boolean, Byte, Double, Global, Long, Real, String

Integer Statement Example

The following example shows a simple program that declares some variables using **Integer**.

```
Function inttest
    Integer A(10)          'Single dimension array of integer
    Integer B(10, 10)      'Two dimension array of integer
    Integer C(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.

$\boxed{\text{F}}$

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).

See Also
    In, Out, OutW

InW Function Example

```
Long word0

word0 = InW(0)
```

IOLabel$ Function

Returns the I/O label for a specified input or output bit, byte, or word.

F

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

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.

<div style="text-align: right">F</div>

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$

IONumber Function Example

```
Integer IObit

IObit = IONumber("myIO")

IObit = IONUmber("Station" + Str$(station) + "InCycle")
```

J1Flag Statement

Specifies the J1Flag attribute of a point.

> **S**

Syntax
> **(1) J1Flag** *point*, [*value*]
> **(2) J1Flag**

Parameters

> *point*   **P***number* or **P**(*expr*) or point label.
>
> *value*   Optional. Integer expression.
> 0 (/J1F0) J1 range is -90 to +270 degrees
>
> 1 (/J1F1) J1 range is from -270 to -90 or +270 to +450 degrees

Return Values

> The J1Flag attribute specifies the range of values for joint 1 for one point. If *value* is ommited, 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.

<div style="border:1px solid; display:inline-block; padding:4px;">F</div>

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.

`>` **`S`**

Syntax
    (1) J2Flag *point*, [*value*]
    (2) J2Flag

Parameters

    point   **P***number* or **P**(*expr*) or point label.

    value   Optional. Integer expression.
                    0 (/J2F0) J2 range is -180 to +180 degrees

                    1 (/J2F1) J2 range is from -360 to -180 or +180 to +360 degrees

Return Values

The J2Flag attribute specifies the range of values for joint 2 for one point. If *value* is ommited, 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.

<div style="float:right">F</div>

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)
```

J4Flag Statement

Sets the J4Flag attribute of a point.

> **>** **S**

Syntax
 (1) J4Flag *point*, [*value*]
 (2) J4Flag

Parameters
 point   **P***number* or **P**(*expr*) or point label.

 value   Optional. Integer expression.
      0 (/J4F0) J4 range is -180 to +180 degrees

      1 (/J4F1) J4 range is from -360 to -180 or +180 to +360 degrees

Return Values
 The J4Flag attribute specifies the range of values for joint 4 for one point. If *value* is ommited, the J4Flag value for the specified point is displayed. When both parameters are omitted, the J4Flag value is displayed for the current robot position.

See Also
 Elbow, Hand, J4Flag Function, J6Flag, Wrist

J4Flag Statement Example

```
J4Flag P0, 1
J4Flag P(mypoint), 0
```

J4Flag Function

Returns the J4Flag attribute of a point.

$\boxed{\textsf{F}}$

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.

> **>** **S**

Syntax
> **(1) J6Flag** *point*, [*value*]
> **(2) J6Flag**

Parameters
> *point*   **P***number* or **P**(*expr*) or point label.
>
> *value*   Integer expression. Range is 0 - 127 (/J6F0 - /J6F127). J6 range for the specified point is as follows:
>
> (-180 * (*value*+1) < J6 <= 180 * *value*) and (180 * *value* < J6 <= 180 * (*value*+1))

Return Values
> The J6Flag attribute specifies the range of values for joint 6 for one point. If *value* is ommited, the J6Flag value for the specified point is displayed. When both parameters are omitted, the J6Flag value is displayed for the current robot position.

See Also
> Elbow, Hand, J4Flag, J6Flag Function, Wrist

J6Flag Statement Example

```
J6Flag P0, 1
J6Flag P(mypoint), 0
```

J6Flag Function

Returns the J6Flag attribute of a point.

<div style="text-align: right; border: 1px solid; display: inline-block;">F</div>

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 - 127 /J6F0 - /J6F127

See Also
> Elbow, Hand, Wrist, J4Flag, J6Flag Statement

J6Flag Function Example

```
Print J6Flag(pick)
Print J6Flag(P1)
Print J6Flag
Print J6Flag(P1 + P2)
```

JA Function

Returns a robot point specified in joint angles.

F

Syntax
> **JA(***j1*, *j2*, *j3*, *j4*, [*j5*], [*j6*]**)**

Parameters
> *j1 - j6*   Real expressions representing joint angles.

Return Values
> A robot point whose location is determined by the specified joint angles.

Description
Use JA to specify a robot point using joint angles.

When the points returned from JA function specify a singularity of the robot, the joint angles of the robot do not always agree with the joint angles supplied to the JA function as arguments during the execution of a motion command for the points. To operate the robot using the joint angles specified for the JA function, avoid a singularity of the robot.

For example:

```
> go ja(0,0,0,90,0,-90)
> where
WORLD: X:   0.000 mm Y: 655.000 mm Z: 675.000 mm U:   0.000 deg V: -90.000 deg W: -90.000 deg
JOINT: 1:   0.000 deg 2:   0.000 deg 3:   0.000 deg 4:   0.000 deg 5:   0.000 deg 6:   0.000 deg
PULSE: 1:        0 pls 2:        0 pls 3:        0 pls 4:        0 pls 5:        0 pls 6:        0 pls

> go ja(0,0,0,90,0.001,-90)
> where
WORLD: X:  -0.004 mm Y: 655.000 mm Z: 675.000 mm U:   0.000 deg V: -90.000 deg W: -89.999 deg
JOINT: 1:   0.000 deg 2:   0.000 deg 3:   0.000 deg 4:  90.000 deg 5:   0.001 deg 6: -90.000 deg
PULSE: 1:        0 pls 2:        0 pls 3:        0 pls 4: 2621440 pls 5:       29 pls 6: -1638400 pls
```

See Also
> AglToPls, XY

JA Function Example

```
P10 = JA(60, 30, -50, 45)
Go JA(135, 90, -50, 90)
P3 = JA(0, 0, 0, 0, 0, 0)
```

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
>   >

JRange Statement

Defines the permissible working range of the specified joint in pulses.

> S

Syntax

    JRange *jointNumber, lowerLimit, upperLmit*

Parameters

    jointNumber   Integer expression between 1-6 representing the joint for which JRange will be specified.

    lowerLmit   Long integer expression representing the encoder pulse count position for the lower limit range of the specified joint.

    upperLmit   Long Integer expression representing the encoder pulse count position for the upper limit range of the specified joint.

Description

Defines the permissible working range for the specified joint with upper and lower limits in encoder pulse counts. **JRange** is similar to the Range command. However, the Range command requires that all joint range limits be set while the **JRange** command can be used to set each joint working limits individually thus reducing the number of parameters required. To confirm the defined working range, use the Range command.

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

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 Robot manual for maximum working ranges for each robot model since these vary from model to model.

See Also

Range, JRange Function

JRange Statement Example

The following examples are done from the 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.

F

Syntax
  JRange(*jointNumber, paramNumber*)

Parameters
  jointNumber   Specifies reference joint number (integer from 1 - 6) by an expression or numeric value.
  paramNumber   Integer expression containing one of two values:
                      1: Specifies lower limit value.
                      2: Specifies upper limit value.

Return Values
  Range setting (integer value, pulses) of the specified joint.

See Also
  Range, JRange Statement

JRange Function Example

```
Long i, oldRanges(3, 1)

For i = 0 To 3
    oldRanges(i, 0) = JRange(i + 1, 1)
    oldRanges(i, 1) = JRange(i + 1, 2)
Next i
```

JS Function

Jump Sense detects whether the arm stopped prior to completing a Jump, Jump3, or Jump3CP instruction which used a Sense input or if the arm completed the move.

F

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, Jump3CP Instruction:

JS can only be used to check the most recent Jump instruction's input check (which is initiated by the Sense instruction.) Once a 2nd Jump instruction is initiated, the **JS** instruction can only return the status for the 2nd Jump instruction. The **JS** status for the first Jump is gone forever. So be sure to always do any **JS** status check for Jump instructions immediately following the Jump instruction to be checked.

See Also

JT, Jump, Jump3, Jump3CP, Sense

JS Function Example

```
Function SearchSensor As Boolean
    Sense Sw(5) = On

    Jump P0
    Jump P1 Sense
    If JS = TRUE Then
        Print "Sensor was found"
        SearchSensor = TRUE
    EndIf
Fend
```

JT Function

Returns the status of the most recent Jump, Jump3, or Jump3CP instruction for the current robot.

$\boxed{\text{F}}$

Syntax

    JT

Return Values

JT returns a long with the following bits set or clear:

Bit 0   Set to 1 when rising motion has started or rising distance is 0.
Bit 1   Set to 1 when horizontal motion has started or horizontal distance is 0.
Bit 2   Set to 1 when descent motion has started or descent distance is 0.
Bit 16   Set to 1 when rising motion has completed or rising distance is 0.
Bit 17   Set to 1 when horizontal motion has completed or horizontal distance is 0.
Bit 18   Set to 1 when descent motion has completed or descent distance is 0.

Description

Use JT to determine the status of the most recent Jump command that was stopped before completion by Sense, Till, abort, etc.

See Also

JS, Jump, Jump3, Jump3CP, Sense, Till

JT Function Example

```
Function SearchTill As Boolean

    Till Sw(5) = On

    Jump P0
    Jump P1 Till
    If JT And 4 Then
        Print "Motion stopped during descent"
        SearchTill = TRUE
    EndIf
Fend
```

JTran Statement

Perform a relative move of one joint.

$\boxed{\text{S}}$

Syntax
> **JTran** *jointNumber, distance*

Parameters
> *jointNumber*   Integer expression representing which joint to move.
>
> *distance*   Real expression representing the distance to move in degrees for rotational joints or millimeters for linear joints.

Description
> Use **JTran** to move one joint a specified distance from the current position.

See Also
> Go, Jump, Move, Ptran

JTran Statement Example

```
JTran 1, 20
```

Jump Statement

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.

$\boxed{>}$ $\boxed{S}$

Syntax

> **Jump** *destination* [**C***archNumber*] [**LimZ** *zLimit*] [CP] [*searchExpr*] [*!...!*]

Parameters

destination	The target destination of the motion using a point expression.
archNumber	Optional. The arch number (*archNumber*) specifies which Arch Table entry to use for the Arch type motion caused by the Jump instruction. *archNumber* must always be proceeded by the letter C. (Valid entries are C0-C7.)
zLimit	Optional. This is a Z limit value which represents the maximum position the Z joint will travel to during the **Jump** motion. This can be thought of as the Z Height Ceiling for the **Jump** instruction. Any valid Z joint Coordinate value is acceptable.
CP	Optional. Specifies continuous path motion.
searchExpr	Optional. A Sense, Till or Find expression. **Sense** | **Till** | **Find** **Sense Sw**(*expr*) = {**On** | **Off**} **Till Sw**(*expr*) = {**On** | **Off**} **Find Sw**(*expr*) = {**On** | **Off**}
!...!	Optional. Parallel Processing statements can be added to the Jump instruction to cause I/O and other commands to execute during motion.

Description

Jump moves the arm from the current position to *destination* using what is called Arch Motion. Jump can be thought of as 3 motions in 1. For example, when the Arch table entry defined by *archNumber* is 7, the following 3 motions will occur.

1) The move begins with only Z-joint motion until it reaches the Z joint height calculated by the Arch number used for the Jump command.
2) Next the arm moves horizontally (while still moving upward in Z) towards the target point position until the upper Z Limit (defined by LimZ) is reached. Then the arm begins to move downward in the Z direction (while continuing X, Y and U joint motion) until the final X, and Y and U joint positions are reached.
3) The **Jump** instruction is then completed by moving the arm down with only Z-joint motion until the target Z-joint position is reached.

The coordinates of *destination* (the target position for the move) must be taught previously before executing the **Jump** instruction. The coordinates cannot be specified in the **Jump** instruction itself. Acceleration and deceleration for the **Jump** is controlled by the Accel instruction. Speed for the move is controlled by the Speed instruction.

archNumber Details

The Arch for the **Jump** instruction can be modified based on the *archNumber* value optionally specified with the **Jump** instruction. This allows the user to define how much Z to move before beginning the X, Y, and U joint motion. (This allows the user to move the arm up and out of the way of parts, feeders and other objects before beginning horizontal motion.) Valid *archNumber* entries for the **Jump** instruction are between C0-C7. The Arch table entries for C0-C6 are user definable with the Arch instruction. However, C7 is a special Arch entry which always defines what is called Gate Motion. Gate Motion means that the robot first moves Z all the way to the coordinate defined by LimZ before beginning any X, Y, or U joint motion. Once the LimZ Z limit is reached, X, Y and U joint motion begins. After the X, Y, and U joints each reaches its final destination position, then the Z joint can begin moving downward towards the final Z joint coordinate position as defined by *destination* (the target point). Gate Motion looks as follows:
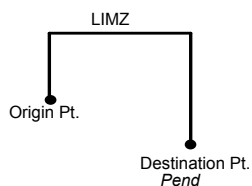
LimZ Details

LimZ *zLimit* specifies the upper Z coordinate value for the horizontal movement plane in the current local coordinate system. The specified arch settings can cause the X, Y, and U joints to begin movement before reaching LimZ, but LimZ is always the maximum Z height for the move. When the LimZ optional parameter is omitted, the previous value specified by the LimZ instruction is used for the horizontal movement plane definition.

It is important to note that the LimZ *zLimit* height limit specification is the Z value for the local robot coordinate system. It is not the Z value for Arm or Tool. Therefore take the necessary precautions when using tools or hands with different operating heights.

Sense Details

The Sense optional parameter allows the user to check for an input condition or memory I/O condition before beginning the final Z motion downward. If satisfied, this command completes with the robot stopped above the target position where only Z motion is required to reach the target position. It is important to note that the robot arm does not stop immediately upon sensing the Sense input modifier.
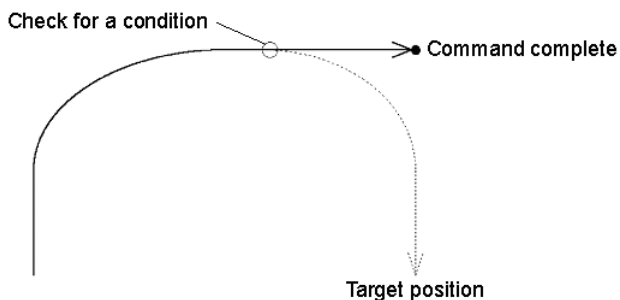
The JS or Stat commands can then be used to verify whether the Sense condition was satisfied and the robot stopped prior to its target position or that the Sense condition was not satisfied and the robot continued until stopping at its target position.

Till Details

The optional Till qualifier allows the user to specify a condition to cause the robot to decelerate to a stop prior to completing the **Jump**. The condition specified is simply a check against one of the 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

Use Jump3 or Jump3CP for 6-axis robots.

Jump Motion trajectory changes depending on motion and speed

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

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

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

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

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

Omitting *archNumber* Parameter

If the archnum optional parameter is omitted, the default Arch entry for use with the Jump instruction is C7. This will cause Gate Motion, as described above.

Difference between Jump and Jump3, Jump3CP

The Jump3 and Jump3CP instructions can be used for 6-axis robots. On the other hand the Jump instruction cannot be used for 6-axis robots. For SCARA 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
```
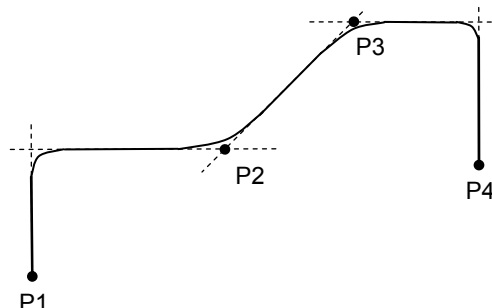

Potential Errors

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, Point Expression, Pulse, Sense, Speed, Stat, Till

Jump Statement Example

The example shown below shows a simple point to point move between points P0 and P1 and then moves back to P0 using the Jump instruction. Later in the program the arm moves using the Jump instruction. 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!
```

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.

> **S**

Syntax

(1) Jump3 *depart, approach, destination* [**C***archNumber*] [**CP**] [LJM [*orientationFlag*]] [*searchExpr*] [*!...!*]

(2) Jump3CP *depart, approach, destination* [**ROT**] [**C***archNumber*] [**CP**] [LJM [*orientationFlag*]] [*searchExpr*] [*!...!*]

Parameters

depart	The departure point above the current position using a point expression.
approach	The approach point above the destination position a point expression.
destination	The target destination of the motion using a point expression.
ROT	Optional. :Decides the speed/acceleration/deceleration in favor of tool rotation.
archNumber	Optional. The arch number (*archNumber*) specifies which Arch Table entry to use for the Arch type motion caused by the Jump instruction. *archNumber* must always be proceeded by the letter C. (Valid entries are C0-C7.)
CP	Optional. Specifies continuous path motion.
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.

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.

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4
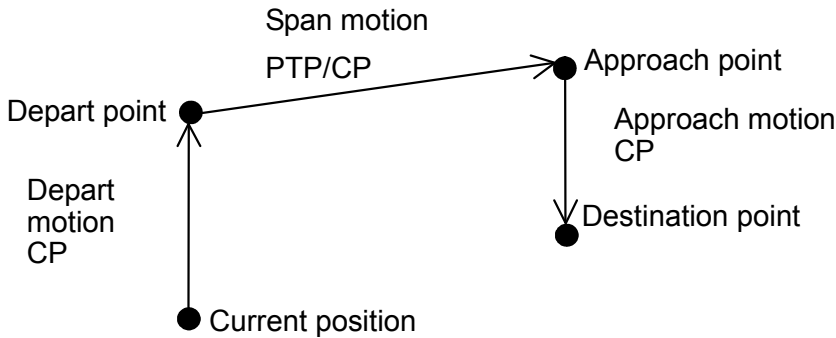
Arch motion is achieved by specifying the arch number. The arch motion for Jump3, Jump3CP is as shown in the figure below. For arch motion to occur, the Depart distance must be greater than the arch upward distance and the Approach distance must be greater than the arch downward distance.
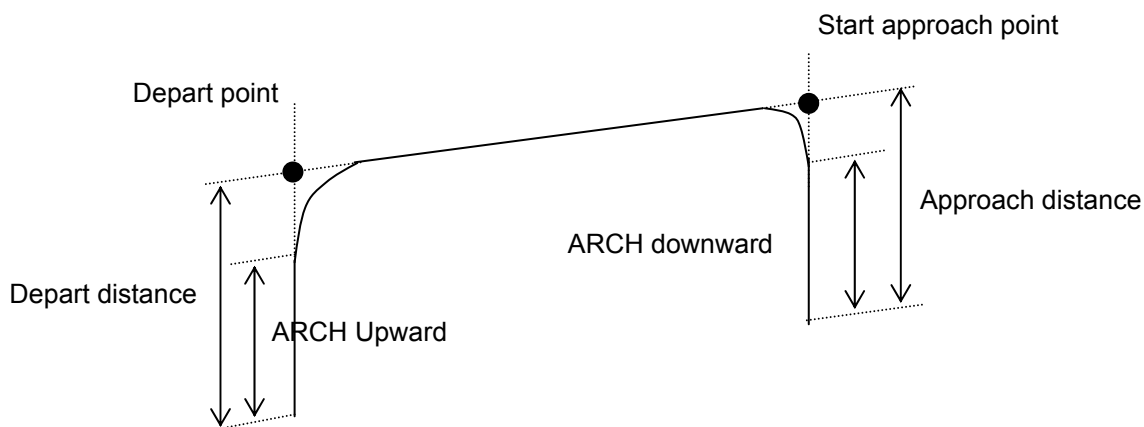
Jump3CP uses the SpeedS speed value and AccelS acceleration and deceleration values. Refer to *Using Jump3CP with CP* below on the relation between the speed/acceleration and the acceleration/deceleration. If, however, the ROT modifier parameter is used, **Jump3CP** uses the SpeedR speed value and AccelR acceleration and deceleration values. In this case SpeedS speed value and AccelS acceleration and deceleration value have no effect.

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

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

Notes

Jump3 span motion is PTP (point to point)

It is difficult to predict Jump3 span motion trajectory. Therefore, be careful that the robot doesn't collide with peripheral equipment and that robot arms don't collide with the robot.

Jump3 Motion trajectory changes depending on motion and speed

Jump3 motion trajectory is comprised of depart, span, and approach motions. It is not a continuous path trajectory. The actual Jump3 trajectory of arch motion is not determined by **Arch** parameters alone. It also depends on motion and speed.

Always use care when optimizing Jump3 trajectory in your applications. Execute Jump3 with the desired motion and speed to verify the actual trajectory.

When speed is lower, the trajectory will be lower. If Jump3 is executed with high speed to verify an arch motion trajectory, the end effector may crash into an obstacle with lower speed.

In a Jump3 trajectory, the depart distance increases and the approach distance decreases when the motion speed is set high. When the approach distance of the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the approach distance to be larger.

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

LimZ does not affect Jump3 and Jump3CP

LimZ has no affect on Jump3 or Jump3CP since the span motion is not necessarily perpendicular to the Z axis of the coordinate system.

Potential acceleration errors

An acceleration error may occur during an arch motion execution by the Jump3 andJump3CP commands. This error is issued frequently when the majority of the motion during depart or approach uses the same joint as the span motion. To avoid this error, reduce the acceleration/deceleration speed of the span motion using Accel command for Jump3 or using AccelS command for Jump3CP. Depending on the motion and orientation of the robot, it may also help to reduce the acceleration and deceleration of the depart motion (approach motion) using the AccelS command.

Using Jump3, Jump3CP with CP

The CP parameter causes the arm to move to *destination* without decelerating or stopping at the point defined by *destination*. This is done to allow the user to string a series of motion instructions together to cause the arm to move along a continuous path while maintaining a specified speed throughout all the motion. The **Jump3** and **Jump3CP** instructions without CP always cause the arm to decelerate to a stop prior to reaching the point *desination*.

Pass function of Jump3

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)
```
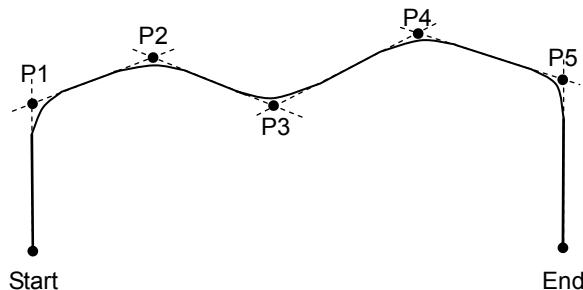
Use Jump3, Jump3CP with LJM

With LJM parameter, the program using LJM function can be more simple.

For ecample, 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 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.

See Also

Accel, Arc, Arch, Go, JS, JT, Point Expression, Pulse, Sense, Speed, Stat, Till

Jump3 Statement Example

```
' 6 axis robot motion which works like Jump of SCARA robot
Jump3  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
```

LCase$ Function

Returns a string that has been converted to lowercase.

F

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.

$\boxed{\textbf{F}}$

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.

$\boxed{\textbf{F}}$

Syntax
   Len*(string)*

Parameters
   string   String expression.

Return Values
   Returns an integer number representing the number of characters in the string *string* which was given as an argument to the **Len** instruction.

Description
   Len returns an integer number representing the number of characters in a string specified by the user. **Len** will return values between 0-255 (since a string can contain between 0-255 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
>
```

LimZ Statement

Determines the default value of the Z joint height for Jump commands.

> S

Syntax

    (1) LimZ *zLimit*
    (2) LimZ

Parameters

    zLimit   A coordinate value within the movable range of the Z joint.

Return Values

    Displays the current LimZ value when parameter is omitted.

Description

    LimZ determines the maximum Z joint height which the arm move to when using the Jump instruction, wherein the robot arm raises on the Z joint, moves in the X-Y plane, then lowers on the Z joint. **LimZ** is simply a default Z joint value used to define the Z joint ceiling position for use during motion caused by the Jump instruction. When a specific **LimZ** value is not specified in the Jump instruction, the last **LimZ** setting is used for the Jump instruction.

Note

Resetting LimZ to 0

    Restarting the 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 affect on Jump3 or Jump3CP since the span motion is not necessarily perpendicular to the Z axis of the coordinate system.

See Also

    Jump

LimZ Statement Example

    The example below shows the use of LimZ in Jump operations.

```
Function main
    LimZ -10            'Set the default LimZ value
    Jump P1             'Move up to Z=-10 position for Jump
    Jump P2 LimZ -20    'Move up to Z=-20 position for Jump
    Jump P3          'Move up to Z=-10 position for Jump
Fend
```

LimZ Function

Returns the current LimZ setting.

F

Syntax
    LimZ

Return Values
    Real number containing the current LimZ setting.

See Also
    LimZ Statement

LimZ Function Example

```
Real savLimz

savLimz = LimZ
LimZ -25
Go pick
LimZ savLimZ
```

Line Input Statement

Reads input data of one line and assigns the data to a string variable.

> S

Syntax
>   **Line Input** *stringVar$*

Parameters
>   *stringVar$*   A string variable name. (Remember that the string variable must end with the
> $ character.)

Description
>   **Line Input** reads input data of one line from the 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 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 the specified communication port or the device.

> S

Syntax
>   **Line Input** #*handle, stringVar$*

Parameters

handle	The communications handle or the device ID. Communication handles can be specified in OpenCom (RS232) and OpenNet (TCP/IP) statements.
	Device ID integers are as follows. 21 RC+ 23 OP 24 TP
stringVar$	A string variable. (Remember that string variables must end with a $ character.)

Description
>   **Line Input #** reads string data of one line from the device specified with the *handle* parameter, and assigns the data to the string variable *stringVar$.*

See Also
>   Input, Input #, Line Input

Line Input # Example
>   This example receives the string data from the communication port number 1, and assigns the data to the string variable A$.

```
Function lintest
    String a$
    Print #1, "Please input string to be sent to robot"
    Line Input #1, a$
    Print "Value entered = ", a$
Fend
```

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.

F

Syntax

    (3) **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 or J6Flag. (default)
	2:	Converts the J4Flag or J6Flag.
RS series	1:	Converts the hand orientation (Hand Flag), J1Flag or J2Flag. (default)
	2:	Converts the hand orientation (Hand Flag), J1Flag or J2Flag.
		Prevents the U axis from moving out of motion range at flag convert.

Description

When the 6-axis 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.

In the same way, 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 and RS series robot.

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 chaeck
      ' 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.

> **S**

Syntax
    LoadPoints *fileName* **[, Merge]**

Parameters

fileName	String expression containing the specific file to load into the current robot's point memory area. The extension is .PTS appended to the end so no extension is to be specified by the user. The file must exist in the current project. No path can be specified.
Merge	Optional. If supplied, then the current points are not cleared before loading the new points. Points in the file are added the to the current points. If a point exists in the file, it will overwrite the point in memory.

Description

LoadPoints loads point files into the main memory area of the controller.

Use **Merge** to combine point files. For example, you could have one main point file that includes common points for locals, parking, etc in the range 0 - 100. Then use **Merge** to load other point files for each part being run without clearing the common points. The range could be 101 - 999.

Potential Errors

File Does Not Exist
    If *fileName* does not exist, an error will be issued.

See Also
    ClearPoints, 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

Fend
```

Local Statement

Defines and displays local coordinate systems.

> **>** **S**

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.
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.

Description

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also
ArmSet, Base, ECPSet, LocalClr, TLSet, Where

Local 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
```

Local defined with only the origin point rotated 45 degrees about the X axis:

```
> local 2, 50, 200, 0, 0, 45
```

3D Local with p2 aligned with the X axis of the local:

```
> local 3, p1, p2, p3, x
```

3D Local with p3 aligned with the Y axis of the local:

```
> local 4, p1, p2, p3, y
```

Local Function

Returns the local number of a point.

F

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.

> **>** **S**

Syntax
    LocalClr *localNumber*

Parameters
    localNumber   Integer expression representing which of 15 locals (integer from 1 to 15) to clear (undefine).

Description
    Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also
    Arm, ArmSet, ECPSet, Local, Tool, TLClr, TLSet

LocalClr 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 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.

F

Syntax
> **Lof**(*portNumber*)

Parameters
> *portNumber*   The communication port number.

Return Values
> The number of lines of data in the buffer. If there is no data in the buffer, **Lof** returns 0.

Description
> **Lof** checks whether or not the specified port has received data lines. The data received is stored in the buffer irrespective of the Input# instruction.

See Also
> ChkCom, ChkNet, Input#

Lof Function Example
> This Command window example prints out the number of lines of data received through the communication port number 1.

```
>print lof(1)
 5
>
```

Long Statement

Declares variables of type long integer. (4 byte whole number).

$\boxed{\textsf{S}}$

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 to the upper bound value.

              The total available number of array elements for local and global preserve variables is 1000.

              The total available number of array elements for global and module variables is 10000.

              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

   Long is used to declare variables as type **Long**. Variables of type **Long** can contain whole numbers with values between -2,147,483,648 to 2,147,483,647. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

See Also

   Boolean, Byte, Double, Global, Integer, Real, String

Long Statement Example

   The following example shows a simple program which declares some variables as Longs using **Long**.

```
Function longtest
    Long A(10)          'Single dimension array of long
    Long B(10, 10)      'Two dimension array of long
    Long C(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..

$$\boxed{\textbf{F}}$$

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.

$\boxed{F}$

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, Integer, Long, and Real.

See Also

    And, Not, Or, RShift, Xor

LShift Function Example

```
Function lshiftst
    Integer i
    Integer num, snum
    num = 1
    For i = 1 to 10
        Print "i =", i
        snum = LShift(num, i)
        Print "The shifted num is ", snum
    Next i
Fend
```

Some other example results from the LShift instruction from the command window.

```
> Print LShift(2,2)
8
> Print LShift(5,1)
10
> Print LShift(3,2)
12
>
```

LTrim$ Function

F

Returns a string equal to specified string without leading spaces.

Syntax
    LTrim$ (*string*)

Parameters
    string   String expression.

Return Values
    Specified string with leading spaces removed.

See Also
    RTrim$, Trim$

LTrim$ Function Example

```
str$ = "  data  "
str$ = LTrim$(str$)  ' str$ = "data  "
```

Mask Operator

Bitwise mask for Wait statement condition expression.

$\boxed{\textbf{S}}$

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
```

MemIn Function

Returns the status of the specified memory I/O port. Each port contains 8 memory bits.

$\boxed{\textbf{F}}$

Syntax
    MemIn(*portNumber***)**

Parameters
    portNumber Integer expression representing memory I/O bytes.

Return Values
    Returns an integer value between 0-255. The return value is 8 bits, with each bit corresponding to 1 memory I/O bit.

Description
    MemIn provides the ability to look at the value of 8 memory I/O bits at the same time. The **MemIn** instruction can be used to store the 8 memory I/O bit status into a variable or it can be used with the Wait instruction to Wait until a specific condition which involves more than 1 memory I/O bit is met.

    Since 8 bits are retrieved at a time, the return value ranges from 0-255. Please review the chart below to see how the integer return values correspond to individual memory I/O bits.

Memory I/O Bit Result (Using Port #0)

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

Memory I/O Bit Result (Using Port #31)

Return Value	255	254	253	252	251	250	249	248
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

Notes

Difference Between MemIn and MemSw
    The MemSw instruction allows the user to read the value of 1 memory I/O bit. The return value from MemSw is either a 1 or a 0 which indicates that the memory I/O bit is either On or Off. MemSw can check each of the memory I/O bits individually. The **MemIn** instruction is very similar to the MemSw instruction in that it also is used to check the status of the memory I/O bits. However there is 1 distinct difference. The MemIn instruction checks 8 memory I/O bits at a time vs. the single bit checking functionality of the MemSw instruction. **MemIn** returns a value between 0-255 which tells the user which of the 8 I/O bits are On and which are Off.

See Also
    In, InBCD, Off, MemOff, On, MemOn, OpBCD, Oport, Out, MemOut, Sw, MemSw, Wait

MemIn Example

The program example below gets the current value of the first 8 memory I/O bits and then makes sure that all 8 I/O are currently set to 0 before proceeding. If they are not 0 an error message is given to the operator and the task is stopped.

```
Function main
    Integer var1

    var1 = MemIn(0)  'Get 1st 8 memory I/O bit values
    If var1 = 0 Then
        Go P1
        Go P2
    Else
        Print "Error in initialization!"
        Print "First 8 memory I/O bits were not all set to 0"
    EndIf
Fend
```

Other simple examples from the 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

F

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 from 0 to 15 representing the memory I/O Word.

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.

> **>** **S**

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.

> **S**

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

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.

> **>** **S**

Syntax
MemOut *portNumber, outData*

Parameters
portNumber   Integer expression representing memory I/O bit port number. The *portNumber* selection corresponds to the following:

Portnum	Outputs
0	0-7
1	8-15
.	.

outData   Integer expression between 0-255 representing the output pattern for the output group selected by *portNumber*. If represented in hexadecimal form the range is from &H0 to &HFF. The lower digit represents the least significant digits (or the 1st 4 outputs) and the upper digit represents the most significant digits (or the 2nd 4 outputs).

Description
MemOut simultaneously sets 8 memory I/O bits using the combination of the *portNumber* and *outData* values specified by the user to determine which outputs will be set. The *portNumber* parameter specifies which group of 8 outputs to use where *portNumber* = 0 means outputs 0-7, *portNumber* = 1 means outputs 8-15, etc.

Once a *portNumber* is selected, a specific output pattern must be defined. This is done using the *outData* parameter. The *outData* parameter may have a value between 0-255 and may be represented in hexadecimal or integer format. (i.e. &H0-&HFF or 0-255)

The table below shows some of the possible I/O combinations and their associated *outData* values assuming that *portNumber* is 0, and 1 accordingly.

Output Settings When *portNumber*=0 (Output number)

OutData Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Output Settings When *portNumber*=1 (Output number)

OutData Value	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

See Also

In, MemIn, InBCD, MemOff, MemOn, MemSw, Off, On, OpBCD, Oport, Out, Sw, Wait

MemOut Example

The example below shows main task starting a background task called iotask. The iotask is a simple task to toggle memory I/O bits 0 - 3 On and Off. The MemOut instruction makes this possible using only 1 command rather than turning each memory I/O bit on and off individually.

```
Function main
  Xqt 2, iotask
  Go P1
   .
   .
Fend

Function iotask

  Do

    Wait 1
    MemOut 0, &H0
    Wait 1
  Loop
Fend
```

Other simple examples from the 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.

> **>** **S**

Syntax
> **MemOutW** *wordPortNum, outputData*

Parameters

wordPortNum	Integer expression from 0 to 31 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 Example

```
MemOutW 0, 25
```

MemSw Function

Returns the status of the specified memory I/O bit.

$\boxed{\textbf{F}}$

Syntax

   MemSw(bitNumber**)**

Parameters

   bitNumber   Integer expression representing the memory I/O bit number.

Return Values

   Returns a 1 when the specified bit is On and a 0 when the specified bit is Off.

Description

   MemSw returns the status of one memory I/O bit. Valid entries for MemSw range from bit 0 to bit 511. MemOn turns the specified bit on and MemOff turns the specified bit Off.

See Also

   In, MemIn, InBCD, MemOff, MemOn, MemOut, Off, On, OpBCD, Oport, Out, Sw, Wait

MemSw Example

   The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. MemSw is used in combination with the Wait instruction to wait until the memory I/O bit 1 is the proper value before it is safe to move again.

```
Function main
    Integer I
    MemOff 1
    Xqt 2, task2
    For i = 1 to 100
        Wait MemSw(1) = Off
        Go P(i)
        MemOn 1
    Next I
Fend

Function task2
    Integer I
    For i = 101 to 200
        Wait MemSw(1) = On
        Go P(i)
        MemOff 1
    Next I
Fend
```

   Other simple examples from the Command window are as follows:

```
> memon 1
> print memsw(1)
1
> memoff 1
> print memsw(1)
0
```

Mid$ Function

Returns a substring of a string starting from a specified position.

$$\boxed{\textbf{F}}$$

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
```

Mod Operator

Returns the remainder obtained by dividing a numeric expression by another numeric expression.

Syntax
number **Mod** *divisor*

Parameters
number   The number being divided (the dividend).

divisor   The number which *number* is divided by.

Return Values
Returns the remainder after dividing *number* by *divisor*.

Description
Mod is used to get the remainder after dividing 2 numbers. The remainder is a whole number. One clever use of the **Mod** instruction is to determine if a number is odd or even. The method in which the **Mod** instruction works is as follows: *number* is divided by *divisor*. The remainder left over after this division is then the return value for the **Mod** instruction.

See Also
Abs, Atan, Atan2, Cos, Int, Not, Sgn, Sin, Sqr, Str$, Tan, Val

Mod Operator Example
The example shown below determines if a number (var1) is even or odd. When the number is even the result of the Mod instruction will return a 0. When the number is odd, the result of the Mod instruction will return a 1.

```
Function modtest
....Integer var1, result

....Print "Enter an integer number:"
....Input var1
....result = var1 Mod 2
....Print "Result = ", result
....If result = 0 Then
........Print "The number is EVEN"
....Else
........Print "The number is ODD"
....EndIf
Fend
```

Some other example results from the Mod instruction from the 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.

> S

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 automatically sets the following items:

Power	Low
Fine	Default values
Speed	Default values
SpeedR	Default values
SpeedS	Default values
Accel	Default values
AccelS	Default values
AccelR	Default values
PTPBoost	Default values
LimZ	0

See Also

    Brake, Power, Reset, SFree, SLock

Motor Example

    The following examples are done from the command window:

```
> Motor On

> Motor Off
```

Motor Function

Returns status of motor power for the current robot.

Syntax
    Motor

Return Values
    0 = Motors off, 1 = Motors on.

See Also
    Motor Statement

Motor Function Example

```
If Motor = Off Then
     Motor On
EndIf
```

Move Statement

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).

$\boxed{>}$ $\boxed{\text{S}}$

Syntax

Move *destination* [**ROT**] [**ECP**] [**CP**] [*searchExpr*] [*!...!*]

Parameters

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

Description

Move moves the arm from the current position to *destination* in a straight line. **Move** coordinates all axes to start and stop at the same time. The coordinates of *destination* must be taught previously before executing the **Move** instruction. Acceleration and deceleration for the **Move** is controlled by the AccelS instruction. Speed for the move is controlled by the SpeedS instruction. If the SpeedS speed value exceeds the allowable speed for any joint, power to all four joint motors will be turned off, and the robot will stop.

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

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

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

When ECP is used, the trajectory of the external control point coresponding to the ECP number specified by ECP instruction moves straight with respect to the tool coordinate system. In this case, the trajectory of tool center point does not follow a straight line.
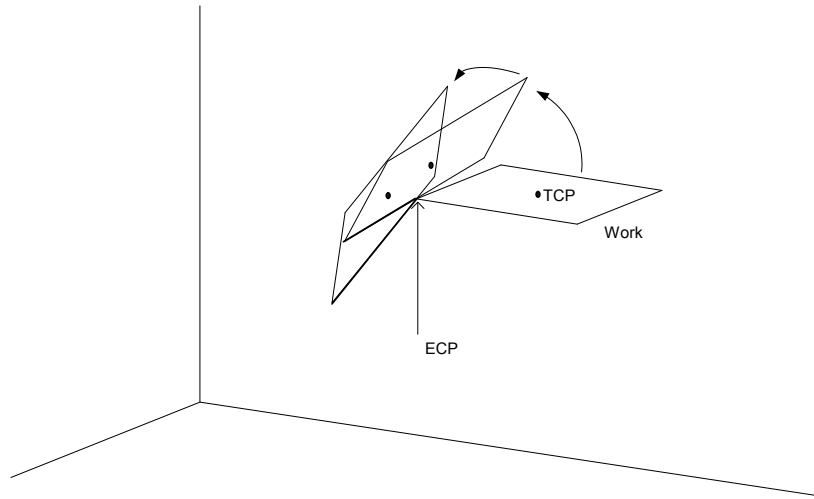
The optional Till qualifier allows the user to specify a condition to cause the robot to decelerate to a stop prior to completing the **Move**. The condition specified is simply a check against one of the inputs. This is accomplished through using the Sw instruction. The user can check if the input is On or Off and cause the arm to stop based on the condition specified. This feature works almost like an interrupt where the **Move** is interrupted (stopped) once the Input condition is met. If the input condition is never met during the **Move** then the arm successfully arrives on the point specified by *destination.* For more information about the Till qualifier see the Till command.

Notes

Move Cannot

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 *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.

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

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 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.
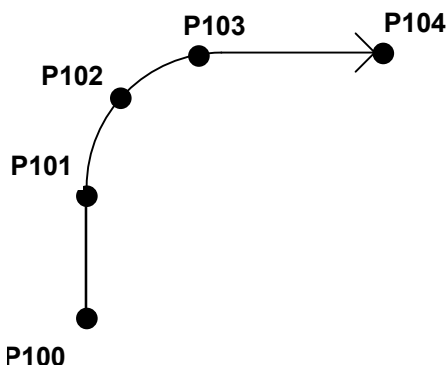

```
Function CornerArc
  Go P100
  Move P101 CP        'Do not stop at P101
  Arc P102, P103 CP   'Do not stop at P103
  Move P104           'Decelerate to stop at P104
Fend
```

MyTask Function

Returns the task number of the current program.

F

Syntax
> **MyTask**

Return Values
> The task number of the current task. Valid entries are integer numbers 1-16.

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.

>   **S**

Syntax
> **For** *var1* **=** *initval* **To** *finalval* **[Step** *Increment* **]**
> > *statements*
>
> **Next** *var1*

Parameters

var1	The counting variable used with the For/Next loop. This variable is normally defined as an integer but may also be defined as a Real variable.
initval	The initial value for the counter *var1*.
finalval	The final value of the counter var1. Once this value is met, the For/Next loop is complete and execution continues starting with the statement following the Next instruction.
Increment	An optional parameter which defines the counting increment for each time the Next statement is executed within the For/Next loop. This variable may be positive or negative. However, if the value is negative, the initial value of the variable must be larger than the final value of the variable. If the increment value is left out the system automatically increments by 1.
statements	Any valid SPEL$^+$ statements can be inserted inside the For/Next loop.

Return Values
> None

Description

> **For/Next** executes a set of statements within a loop a specified number of times. The beginning of the loop is the For statement. The end of the loop is the **Next** statement. A variable is used to count the number of times the statements inside the loop are executed.

> The first numeric expression (*initval*) is the initial value of the counter. This value may be positive or negative as long as the *finalval* variable and Step increment correspond correctly.

> The second numeric expression (*finalval*) is the final value of the counter. This is the value which once reached causes the For/**Next** loop to terminate and control of the program is passed on to the next instruction following the **Next** instruction.

> Program statements after the For statement are executed until a **Next** instruction is reached. The counter variable (*var1*) is then incremented by the Step value defined by the *increment* parameter. If the Step option is not used, the counter is incremented by one.

> The counter variable (*var1*) is then compared with the final value (*finalval*). If the counter is less than or equal to the final value (*finalval*), the statements following the For instruction are executed again. If the counter variable is greater than the final value (*finalval*), execution branches outside of the For/**Next** loop and continues with the instruction immediately following the **Next** instruction.

> Nesting of For/**Next** statements is supported up to 10 levels deep. This means that a For/**Next** Loop can be put inside of another For/**Next** loop and so on and so on until there are 10 "nests" of For/**Next** loops.

Notes

Negative Step Values

If the value of the Step increment (*increment*) is negative, the counter variable (var1) is decremented (decreased) each time through the loop and the initial value (*initval*) must be greater than the final value (*finalval*) for the loop to work.

See Also

For

For/Next Example

```
Function fornext
    Integer ctr
    For ctr = 1 to 10
        Go Pctr
    Next ctr
    '
    For ctr = 10 to 1 Step -1
        Go Pctr
    Next ctr
Fend
```

Not Operator

Performs the bitwise complement on the value of the operand.

F

Syntax
    Not o*perand*

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.

> S

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. <u>If prior to executing Off, the Output bit was already off, then it is switched On after the time interval elapses</u>.

The *parallel* parameter settings are applicable when the time interval is specified as follows:
1: Switches the output off, switches it back on after specified interval elapses, then executes the next command. (This is also the default value for the parallel parameter. If this parameter is omitted, this is the same as setting the parameter to 1.)
0: Switches the output off, and simultaneously executes the next command.

Notes

Output bits Configured as Remote Control output

If an output bit which was set up as a system output is specified, an error will occur. Remote control output bits are turned on or off automatically according to system status.

Outputs and When an Emergency Stop Occurs:

EPSON RC+ has a feature which causes all outputs to go off when an E-Stop occurs. This feature is set or disabled from Setup | 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 secs, 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.

> | S

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 Power On
Motor On
SFree, SLock
Reset
Stop button or Ctrl + C Key
```

Notes

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

```
>olrate on
>olrate
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.

F

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.

> S

Syntax
    OLRate [*jointNumber*]

Parameters
    jointNumber   Integer expression whose value is between 1 and 6.

Description
    OLRate can be used to check whether a cycle is causing stress on the servo system. Factors such as temperature and current can cause servo errors during applications with high duty cycles. **OLRate** can help to check if the robot system is close to having a servo error.

    During a cycle, run another task to 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.

$$\boxed{\text{F}}$$

Syntax
   OLRate(*jointNumber***)**

Parameters
   jointNumber   Integer expression whose value is between 1 and 6.

Return Values
   Returns the OLRate for the specified joint. Values are between 0.0 and 2.0.

Description
   OLRate can be used to check whether a cycle is causing stress on the servo system. Factors such as temperature and current can cause servo errors during applications with high duty cycles. **OLRate** can help to check if the robot system is close to having a servo error.

   During a cycle, run another task to 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 Statement

OLRate Function Example

```
Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Integer i
  Real olRates(4)
  Do
    For i = 1 to 4
      olRates(i) = OLRate(i)
      If olRate(i) > .5 Then
        Print "Warning: OLRate(", i, ") is over .5"
      EndIf
    Next i
  Loop
Fend
```

On Statement

Turns on the specified output and after a specified time can turn it back off.

<div style="float:right">[**>**] [**S**]</div>

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 the EPSON RC+ User's Guide. The individual bits for the remote connector can be set as remote or I/O from the EPSON RC+ remote configuration dialog accessible from the setup menu.

Outputs and When an Emergency Stop Occurs
The Controller has a feature which causes all outputs to go off when an E-Stop occurs. This feature is set or disabled from one of the Option Switches. To configure this go to the Setup | 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 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, 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 secs, 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.

S

Syntax
  OnErr GoTo {*label* | **0**}

Parameters
label	Statement label to jump to when an error occurs.
0	Parameter used to clear OnErr setting.

Description
  OnErr enables user error handling. When an error occurs without **OnErr** being used, the task is
  terminated and the error is displayed. However, when **OnErr** is used it allows the user to "catch" the
  error and go to an error handler to automatically recover from the error. Upon receiving an error,
  OnErr branches control to the designated 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 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)

> S

Syntax
OpBCD portNumber, outData [,*Forced*]

Parameters

portNumber   Integer expression representing I/O output bytes.   Where the *portNumber* selection corresponds to the following outputs:

PortNumber	Outputs
0	0-7
1	8-15
2	16-23
3	24-31
...	...

outData   Integer expression between 0-99 representing the output pattern for the output group selected by *portNumber*. The 2nd digit (called the 1's digit) represents the lower 4 outputs in the selected group and the 1st digit (called the 10's digit) represents the upper 4 outputs in the selected group.

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 0-7, *portNumber* = 1 means outputs 8-15, etc..

Once a port number is selected (i.e. a group of 8 outputs has be selected), a specific output pattern must be defined. This is done in Binary Coded Decimal format using the *outdata* parameter. The outdata parameter may have 1 or 2 digits. (Valid entries range from 0 to 99.) The 1st digit (or 10's digit) corresponds to the upper 4 outputs of the group of 8 outputs selected by *portNumber*. The 2nd digit (or 1's digit) corresponds to the lower 4 outputs of the group of 8 outputs selected by *portNumber*.

Since valid entries in BCD format range from 0-9 for each digit, every I/O combination cannot be met. The table below shows some of the possible I/O combinations and their associated *outnum* values assuming that *portNumber* is 0.

Output Settings (Output number)

Outnum Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

Notice that the Binary Coded Decimal format only allows decimal values to be specified. This means that through using Binary Coded Decimal format it is impossible to turn on all outputs with the **OpBCD** instruction. Please note that the maximum value for either digit for outnum is 9. This means that the largest value possible to use with OpBCD is 99. In the table above it is easy to see that 99 does not turn all Outputs on. Instead it turns outputs 0, 3, 4, and 7 On and all the others off.

Notes

Difference between OpBCD and Out

The **OpBCD** and Out instructions are very similar in the SPEL⁺ language. However, there is one major difference between the two. This difference is shown below:
- The **OpBCD** instruction uses the Binary Coded Decimal format for specifying an 8 bit value to use for turning the outputs on or off. Since Binary Coded Decimal format precludes the values of &HA, &HB, &HC, &HD, &HE or &HF from being used, all combinations for setting the 8 output group cannot be satisfied.
- The Out instruction works very similarly to the OpBCD instruction except that Out allows the range for the 8 bit value to use for turning outputs on or off to be between 0-255 (vs. 0-99 for OpBCD). This allows all possible combinations for the 8 bit output groups to be initiated according to the users specifications.

Output bits Configured as Remote:

If an output bit which was set up as remote is specified to be turned on by **OpBCD**, an error will occur. Remote output bits are turned On or Off automatically according to system status. For more information regarding remote, refer to the EPSON RC+ User's Guide. The individual bits for the remote connector can be set as remote or I/O from the EPSON RC+ remote configuration dialog accessible from the setup menu.

Outputs and When an Emergency Stop Occurs:

The Controller has a feature which causes all outputs to go off when an E-Stop occurs. This feature is set or disabled from one of the Option Switches. To configure this go to Setup | 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 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, 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
```

OpenCom Statement

Open an RS-232 communication port.

$\boxed{\textbf{S}}$

Syntax
    OpenCom #*portNumber*

Parameters
    portNumber Integer expression for port number to open.

Description
    The specified RS-232 port must be installed on the controller.

See Also
    ChkCom, CloseCom, SetCom

OpenCom Statement Example

```
OpenCom #1
```

OpenNet Statement

Open a TCP/IP network port.

$\boxed{\textbf{S}}$

Syntax
> **OpenNet** #*portNumber* **As** { **Client** | **Server** }

Parameters
> *portNumber* Integer expression for port number to open. Range is 201 - 208.

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
```

Oport Function

$\boxed{\textbf{F}}$

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 a 0 or 1.
>>   **0:** Off status
>>   **1:** On status

Description
>   **Oport** provides a status check for the outputs. It functions much in the same way as the Sw instruction does for inputs. **Oport** is most commonly used to check the status of one of the outputs which could be connected to a feeder, conveyor, gripper solenoid, or a host of other devices which works via discrete I/O. Obviously the output checked with the **Oport** instruction has 2 states (1 or 0). These indicate whether the specified output is On or Off.

Notes

Difference between Oport and Sw
>   It is very important for the user to understand the difference between the **Oport** and Sw instructions. Both instructions are used to get the status of I/O. However, the type of I/O is different between the two. The Sw instruction works inputs. The **Oport** instruction works with the standard and expansion hardware outputs. These hardware ports are discrete outputs which interact with devices external to the controller.

See Also
>   In, InBCD, MemIn, MemOn, MemOff, MemOut, MemSw, Off, On, OpBCD, Out, Sw, Wait

OPort Function Example

The example shown below turns on output 5, then checks to make sure it is on before continuing.

```
Function main
    TMOut 10
    OnErr errchk
    Integer errnum
    On 5      'Turn on output 5
    Wait Oport(5)
    Call mkpart1
    Exit Function

errchk:
    errnum = Err(0)
    If errnum = 94 Then
        Print "TIME Out Error Occurred during period"
        Print "waiting for Oport to come on. Check"
        Print "Output #5 for proper operation. Then"
        Print "restart this program."
    Else
        Print "ERROR number ", errnum, "Occurred"
        Print "Program stopped due to errors!"
    EndIf
    Exit Function
Fend
```

Other simple examples are as follows from the 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.

>   `>`   `S`

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 0-255 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 0-7, *portNumber* = 1 means outputs 8-15, etc..

    Once a portnum is selected (i.e. a group of 8 outputs has be selected), a specific output pattern must be defined. This is done using the *outData* parameter. The *outData* parameter may have a value between 0-255 and may be represented in Hexadecimal or Integer format. (i.e. &H0-&HFF or 0-255)

    The table below shows some of the possible I/O combinations and their associated *outData* values assuming that *portNumber* is 0, and 1 accordingly.

Output Settings When *portNumber*=0 (Output number)

OutData Value	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Output Settings When *portNumber*=1 (Output number)

OutData Value	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

Notes

Difference between OpBCD and Out

The **Out** and OpBCD instructions are very similar in the SPEL$^+$ language. However, there is one major difference between the two. This difference is shown below:

- The OpBCD instruction uses the Binary Coded Decimal format for specifying 8 bit value to use for turning the outputs on or off. Since Binary Coded Decimal format precludes the values of &HA, &HB, &HC, &HD, &HE or &HF from being used, all combinations for setting the 8 output group cannot be satisfied.
- The **Out** instruction works very similarly to the OpBCD instruction except that **Out** allows the range for the 8 bit value to use for turning outputs on or off to be between 0-255 (vs. 0-99 for OpBCD). This allows all possible combinations for the 8 bit output groups to be initiated according to the users specifications.

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 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, MemOff, MemOn, MemOut, MemSw, Off, On, Oport, Sw, Wait

Out Example

The example shown below shows main task start a background task called iotask. The iotask is a simple task to flip flop between turning output bits 0-3 On and then Off. The Out instruction makes this possible using only 1 command rather than turning each output On and Off individually.

```
Function main

  Xqt iotask
  Do
    Go P1
    Go P2
  Loop
Fend

Function iotask

  Do
    Out 0, &H0F
    Out 0, &H00
    Wait 10
  Loop
Fend
```

Other simple examples from the 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.

F

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 lable 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 specidying 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 Example

```
OutReal 0, 2.543
```

OutReal Function

Retrieve the output port status as the 32 bits floating-point data (IEEE754 compliant).

F

Syntax
OutReal (*WordPortNumber*)

Parameter
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 or NoEmgAbort task (special task initiated by specidying 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, OpBCD, OutReal

OutW Example

```
OutW 0, 25
```

OutW Function

Returns the status of one word (2 bytes) of outputs.

F

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
    In, InW, InBCD, InReal, Out, OutW, OpBCD, OutReal

OutW Function Example

```
Long wdata01

wdata01 = OutW(0)
```

PAgl Function

Return a joint value from a specified point.

F

Syntax
> **PAgl** (*point*, *jointNumber)*

Parameters
> *point*   Point expression.
>
> *jointNumber*   Specifies the joint number (integer from 1 to 6) using an expression or numeric value.

Return Values
> Returns the calculated joint position (real value, deg for rotary joint, mm for prismatic joint).

See Also
> Agl, CX, CY, CZ, CU, CV, CW, PPls

PAgl Function Example

```
Real joint1

joint1 = PAgl(P10, 1)
```

Pallet Statement

Defines and displays pallets.

> S

Syntax

　　Pallet [**Outside**,] [*palletNumber, Pi, Pj, Pk* [,*Pm*], *columns, rows*]

Parameters

Outside Optional. Allow row and column indexes outside of the range of the specified rows and columns. Range is from -32768 to 32767.

palletNumber Pallet number represented by an integer number from 0 to 15.

Pi, Pj, Pk Point variables which define standard 3 point pallet position.

Pm Optional. Point variable which is used with Pi, Pj and Pk to define 4 point pallet.

columns Integer expression representing the number of points on the Pi-to-Pj side of the pallet. Range is from 1-32767.

rows Integer expression representing the number of points on the Pi-to-Pk side of the pallet. Range is from 1-32767.

Return Values

　　Displays all defined pallets when parameters are omitted.

Description

　　Defines a pallet by teaching the robot, as a minimum, points Pi, Pj and Pk and by specifying the number of points from Pi to Pj and from Pi to Pk.

　　If the pallet is a well ordered rectangular shape, only 3 of the 4 corner points need to be specified. However, in most situations it is better to use 4 corner points for defining a pallet.

　　To define a pallet, first teach the robot either 3 or 4 corner points, then define the pallet as follows:
A pallet defined with 4 points: P1, P2, P3 and P4 is shown below. There are 3 positions from P1-P2 and 4 positions from P1-P3. This makes a pallet which has 12 positions total. To define this pallet the syntax is as follows:

Points that represent divisions of a pallet are automatically assigned division numbers,

Pallet 1, P1, P2, P3, P4, 3, 4
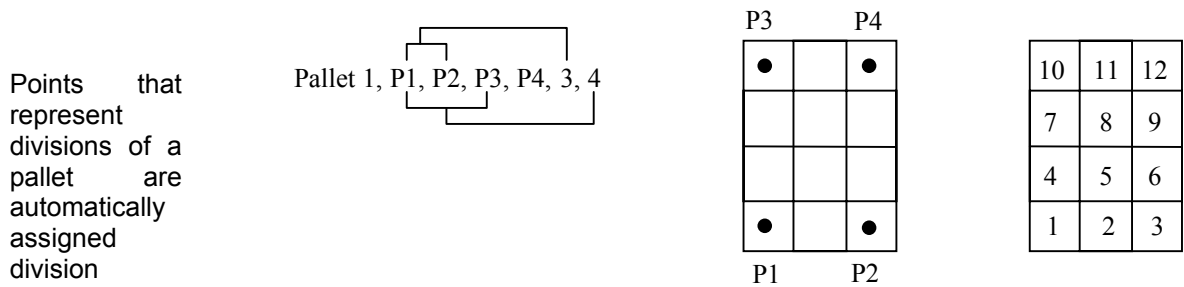
which, in this example, begin at P1. These division numbers are also required by the Pallet Function.

　　When Outside is specified, row and column indexes outside of the range of rows and columns can be specified. The Outside should be specified by two-dimensional division.

For example:

```
Pallet Outside 1, P1, P2, P3, 4, 5
Jump Pallet(1, -2, 10)
```

-2,10						
			1,5	2,5	3,5	4,5
			1,4	2,4	3,4	4,4
			1,3	2,3	3,3	4,3
			1,2	2,2	3,2	4,2
			1,1	2,1	3,1	4,1

Sample

	1,6			4,6		
-1,4		1,4	2,4	3,4	4,4		6,4
		1,3	2,3	3,3	4,3		
		1,2	2,2	3,2	4,2		
-1,1		1,1	2,1	3,1	4,1		6,1
	1,-1			4,-1		

Notes

The Maximum Pallet Size
The total number of points defined by a specific pallet must be less than 32,767.

Incorrect Pallet Shape Definitions
Be aware that incorrect order of points or incorrect number of divisions between points will result in an incorrect pallet shape definition.

Pallet Plane Definition
The pallet plane is defined by the Z axis coordinate values of the 3 corner points of the pallet. Therefore, a vertical pallet could also be defined.

Pallet Definition for a Single Row Pallet

A single row pallet can be defined with a 3 point Pallet statement or command. Simply teach a point at each end and define as follows: Specify 1 as the number of divisions between the same point.
> Pallet 2, P20, P21, P20, 5, 1   'Defines a 5×1 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.

F

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 the Go or the Jump instruction to cause the arm to move the specified pallet position.

   The pallet position number can be defined arithmetically or simply by using an integer.

Notes

Pallet Motion of 6-axis Robot

   When the 6-axis 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 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.

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 chaeck
      ' 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
```

ParseStr Statement / Function

Parse a string and return array of tokens.

$\boxed{F}$ $\boxed{S}$

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.

> [**>**] [**S**]

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 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 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.

$\boxed{\textbf{S}}$

Syntax
> **Pause**

Description
> When the **Pause** instruction 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.

Notes

QP and its Affect on Pause
> The QP instruction is used to cause the arm to stop immediately upon Pause or to complete the current move and then Pause the program. See the QP instruction help for more information.

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.

F

Syntax
    PauseOn

Return Values
    True if the status is pause, otherwise False.

Description
    PuseOn function is used only for NoPause and NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt).

See Also
    ErrorOn, EstopOn, SafetyOn, 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.

$$\boxed{\textbf{F}}$$

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
```

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 Example

```
> p1=10,300,-10,0/L
> p2=0,300,-40,0
> p10=-50,350,0,0
> pdel 1,2        'Delete points 1 and 2
> plist
P10  =  -50.000,  350.000,   0.000,    0.000 /R /0
> pdel 50         'Delete point 50
> pdel 100,200    'Delete from point 100 to point 200
>
```

PLabel Statement

Defines a label for a specified point.

> | S

Syntax
> **PLabel** *pointNumber, newLabel*

Parameters
> *pointNumber*   An integer expression representing a point number.
> *newLabel*   A string expression representing the label to use for the specified point.

See Also
> PDef Function, PLabel Function, PNumber Function

PLabel Statement Example

```
PLabel 1, "pick"
```

PLabel$ Function

Returns the point label associated with a point number.

F

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, PLabel Statement, PNumber Function

PLabel$ Function Example

```
Print PLabel$(1)
Print PLabel$(P(i))
```

Plane Statement

Specifies and displays the approach check plane.

> S

Syntax
- (1) **Plane** *PlaneNum*, *pCoordinateData*
- (2) **Plane** *PlaneNum*, *pOrigin*, *pXaxis, pYaxis*
- (3) **Plane** *PlaneNum*
- (4) **Plane**

Parameters

PlaneNum	Integer expression representing the plane number from 1 to 15.
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 only *PlaneNum* is specified, the plane setting of the specified plane is displayed. When all the parameters are omitted, the plane settings for all plane numbers 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 devided 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 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 orgin 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 InsidePlane function to get the result of the approach check plane. InsidePlane can be used for wait condition of Wait command. You can provide the detection result to the I/O by setting the remote output setting.
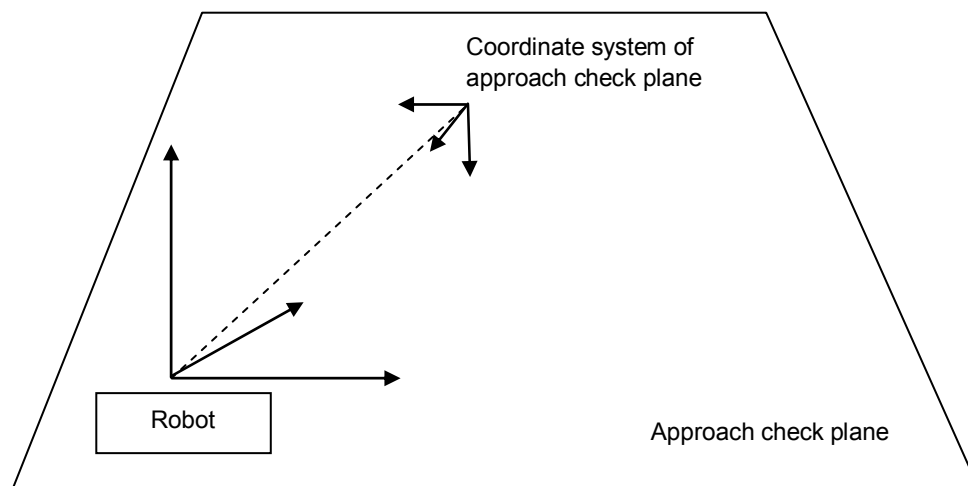
Point data is stored to the Compact Flash inside the Controller. When you execute this command, the data is written to the Compact Flash. If a data is written to the Compact Flash frequently, it may shorten the Compact Flash life. Using this command only when saving the point data is recommended.

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.

See Also

Box, InsidePlane, PlaneClr, PlaneDef

Tip

Set Plane statement from Robot Manager

EPSON RC+ 5.0 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.

$\boxed{\textbf{F}}$

Syntax
  Plane(*PlaneNum*)

Parameters
  PlaneNum   Integer expression representing the plane number from 1 to 15.

Return Values
  Returns coordinate data for specified approach check plane.

See Also
  InsidePlane, Plane, PlaneClr, PlaneDef

Plane Function Example

```
P1 = Plane(1)
```

PlaneClr Statement

Clears (undifines) the Plane definition.

> S

Syntax
    PlaneClr *PlaneNum*

Parameters
    PlaneNum   Integer expression representing the plane number from 1 to 15.

Description
    Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also
    Box, InsidePlane, Plane, PlaneDef

PlaneClr Statement Example

```
PlaneClr 1
```

PlaneDef Function

Returns the setting of the approach check plane.

F

Syntax
   PlaneDef(*PlaneNum*)

Parameters
   PlaneNum   Integer expression representing the plane number from 1 to 15.

Return Values
   True if approach detection plane is defined for the specified plane number, otherwise False.

See Also
   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 pointIndex
    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 Example
    This example displays the point data of the current arm position for the current robot.

```
> plist *
WORLD:  X:   360.000 mm  Y:    10.000 mm  Z:     0.000 mm  U:     0.000 deg
JOINT:  1:     0.000 deg 2:     0.000 deg 3:     0.000 mm  4:     0.000 deg
PULSE:  1:         0 pls 2:         0 pls 3:         0 pls 4:         0 pls
```

    This example displays the point data for one point.

```
> plist 1
P1  =   290.000,     0.000,  -20.000,     0.000 /R /0
>
```

This example displays the point data within the range of 10 and 20. In this example, only three points are found in this range.

```
> plist 10, 20
P10  =   290.000,    0.000,  -20.000,    0.000 /R /0
P12  =   300.000,    0.000,    0.000,    0.000 /R /0
P20  =   285.000,   10.000,  -30.000,   45.000 /R /0
>
```

This example displays the point data starting with point number 10.

```
> plist 10,
P10  =   290.000,    0.000,  -20.000,    0.000 /R /0
P12  =   300.000,    0.000,    0.000,    0.000 /R /0
P20  =   285.000,   10.000,  -30.000,   45.000 /R /0
P30  =   310.000,   20.000,  -50.000,   90.000 /R /0
>
```

PLocal Statement

Sets the local attribute for a point.

> S

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.

F

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 Statement

PLocal Function Example

```
Integer localNum

localNum = PLocal(pick)
```

Pls Function

Returns the current encoder pulse count for each joint at the current position.

F

Syntax

   Pls*(jointNumber)*

Parameters

   jointNumber   The specific joint for which to get the current encoder pulse count.

Return Values

   Returns a number value representing the current encoder pulse count for the joint specified by *jointNumber*.

Description

   Pls is used to read the current encoder position (or Pulse Count) of each joint. These values can be saved and then used later with the Pulse command.

See Also

   CX, CY, CZ, CU, CV, CW, Pulse

Pls Function Example

   Shown below is a simple example to get the pulse values for each joint and print them.

```
Function plstest
  Real t1, t2, z, u
  t1 = pls(1)
  t2 = pls(2)
  z = pls(3)
  u = pls(4)
  Print "T1 joint current Pulse Value: ", t1
  Print "T2 joint current Pulse Value: ", t2
  Print "Z joint current Pulse Value: ", z
  Print "U joint current Pulse Value: ", u
Fend
```

PNumber Function

Returns the point number associated with a point label.

Syntax
   PNumber(*pointLabel***)**

Parameters
   pointLabel   A point label used in the current point file or string expression containing a point label.

See Also
   PDef Function, PLabel$ Function

PNumber Function Example

```
Integer pNum
String pointName$

pNum = PNumber(pick)

pNum = PNumber("pick")

pointName$ = "place"
pNum = PNumber(pointName$)
```

Point Assignment

Defines a robot point by assigning it to a point expression.

> **S**

Syntax
   point = pointExpr

Parameters

point	A robot point specified as follows: **P***number* **P**(*expr*) *pointLabel*
pointExpr	Point expression.

Description
   Define a robot point by setting it equal to another point or point expression.

See Also
   Local, Pallet, PDef, PDel, Plist

Point Assignment Example
   The following examples are done from the 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
```

   Add 90 to U coord of Pallet(3, 5), and define resulting point as P6:

```
> P5 = Here
> P6 = pallet(3,5) +U(90)
```

Point Expression

Specifies a robot point for assignment and motion commands.

> S

Syntax
point [{ + | - } *point*] [*local*] [*hand*] [*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*) **Here** **Pallet**(*palletNumber*, *palletIndex*) *pointLabel* **XY**(*X, Y, Z, U,* [*V*], [*W*]) **JA**(*J1, J2, J3, J4,* [*J5*], [*J6*]) **Pulse**(*J1, J2, J3, J4,* [*J5*], [*J6*])
local	Optional. Local number from 1 to 15 preceded by a forward slash (**/0** to **/15**) or at sign (**@0** to **@15**). The forward slash means that the coordinates will be in the local. The at sign means that the coordinates will be translated into local coordinates.
hand	Optional for SCARA robot (including RS series) and 6-axis robots. Specify **/L** or **/R** for lefty or righty hand orientation.
elbow	Optional for 6-axis robots. Specify **/A** or **/B** for above or below orientation.
wrist	Optional for 6-axis robots. Specify **/F** or **/NF** for flip or no flip orientation.
j4flag	Optional for 6-axis robots. Specify **/J4F0** or **/J4F1**.
j6flag	Optional for 6-axis robots. Specify **/J6F0** - **/J6F127**.
j1flag	Optional for RS series. Specify **/J1F0** or **/J1F1**.
j2flag	Optional for RS series. Specify **/J2F0** - **/J2F127**.
relativeOffsets	Optional. One or more relative coordinate adjustments. {**+**	**-**} {**X**	**Y**	**Z**	**U**	**V**	**W**} (*expr*) The TL offsets are relative offsets in the current tool coordinate system. {**+**	**-**} {**TLX**	**TLY**	**TLZ**	**TLU**	**TLV**	**TLW**} (*expr*)
absoluteCoords	Optional. One or more absolute coordinates. **:** {**X**	**Y**	**Z**	**U**	**V**	**W**} (*expr*)

Description
Point expressions are used in point assignment statements and motion commands.
You can add or subtract points if direct coordinates are not used. For example:

```
Go P1 + P2
P1 = P2 + XY(100, 100, 0, 0)
```

Using relative offsets
You can offset one or more coordinates relative to the base point. For example, the following statement moves the robot 20 mm in the positive X axis from the current position:

```
Go 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.

You can also use relative tool offsets:

```
Go Here +TLX(20) -TLY(5.5)
```

When the 6-axis 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 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 teach a point in a local:

```
P1 = Here @1
```

P1 is set to the current position translated to its position in local 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)
```

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

Previously Called - Lp

$\boxed{>}$ $\boxed{\text{S}}$

Switches Power Mode to high or low and displays the current status.

Power Syntax
(1) Power { **High | Low** }
(2) Power

Parameters
High | Low   The setting can be High or Low. The default is Low.

Return Values
Displays the current Power status when parameter is omitted.

Description
Switches Power Mode to High or Low. It also displays the current mode status.

Low - When Power is set to Low, Low Power Mode is On. This means that the robot will run slow (below 250 mm/sec) and the servo stiffness is set light so as to remove servo power if the robot bumps into an object. This is the normal mode of operation for teaching points.

High - When Power is set to High, Low Power Mode is Off. This means that the robot can run at full speed with the full servo stiffness. This is the normal mode of operation for running actual applications.

The following operations will switch to low power mode. In this case, speed and acceleration settings will be limited to low values.

Conditions to Cause Power Low   **Values Limited**

Conditions to Cause Power Low		Values Limited
• Reset Command • Motor On • All tasks aborted • Teach mode	➔ **Power Low**	**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.

High Power Mode (Power High) and Its Effect on Max Speed:
In high power mode, higher speeds than the default value can be set.

See Also
Accel, AccelS, Speed, SpeedS

Power Example
The following examples are executed from the 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           'Display current speed values
Low Power Mode
   50
   50     50

> Accel           'Display current accel values
Low Power Mode
   100    100
   100    100
   100    100

> Power High      'Set high power mode

> Jump P2         'Move robot at high speed
```

Power Function

Returns status of power.

Syntax
>   **Power**

Return Values
>   0 = Power Low, 1 = Power High.

See Also
>   Power Statement

Power Function Example

```
If Power = 0 Then
    Print "Low Power Mode"
EndIf
```

PPls Function

Return the pulse position of a specified joint value from a specified point.

F

Syntax
> **PPls** (*point*, *jointNumber)*

Parameters
> *point*   Point expression.
>
> *jointNumber*   Specifies the joint number (integer from 1 to 6) using an expression or numeric value.

Return Values
> Returns the calculated joint position (long value, in pulses).

See Also
> Agl, CX, CY, CZ, CU, CV, CW, Pagl

PPls Example

```
Long pulses1

pulses1 = PPls(P10, 1)
```

Print Statement

Outputs data to the current display window, including the Run window, Operator window, Command window, and Macro window.

> S

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.

Notes

Make Sure Print is used with Wait or a motion within a loop
   Tight loops (loops with no Wait or no motion) are generally not good, especially with Print.
   The controller may freeze up in the worst case.
   Be sure to use Print with Wait command or a motion command within a loop.

   Bad example
```
Do
    Print "1234"
Loop
```
   Good example
```
Do
    Print "1234"
    Wait 0.1
Loop
```

See Also
   Print #

Print Statement Example
   The following example extracts the U Axis coordinate value from a Point P100 and puts the coordinate value in the variable *uvar*. The value is then printed to the current display window.

```
Function test
  Real uvar
  uvar = CU(P100)
  Print "The U Axis Coordinate of P100 is ", uvar
Fend
```

Print # Statement

Outputs data to the specified communications port or device.

> S

Syntax

    Print *#portNumber, expression* [*, expression...*]

Parameters

portNumber	The communications handle or the device ID. Communication handles can be specified in OpenCom (RS232) and OpenNet (TCP/IP) statements.
	Device ID integers are as follows. 21 RC+ 23 OP 24 TP
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* .

See Also

    Print

Print # Example

    The following are some simple Print # examples:

```
Function printex
  String temp$
  Print #1, "5"   'send the character "5" to serial port 1 temp$ =
                  "hello"
  Print #1, temp$
  Print #2, temp$
  Print #1 " Next message for port 1"
  Print #2 " Next message for port 2"
Fend
```

PTCLR Statement

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

> S

Syntax

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

Parameters

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

Description

PTCLR clears the peak torque values for the specified joints.

You must execute PTCLR before executing PTRQ.

See Also

ATRQ, PTRQ

PTCLR Statement Example

```
> ptclr
> go p1
> ptrq 1
      0.227
> ptrq
      0.227    0.118
      0.249    0.083
      0.000    0.000
>
```

PTPBoost Statement

Specifies or displays the acceleration, deceleration and speed algorithmic boost parameter for small distance PTP (point to point) motion.

> S

Syntax
   (1) **PTPBoost** *boost*, [*departBoost*], [*approBoost*]
   (2) **PTPBoost**

Parameters

boost	Integer expression from 0 - 100.
departBoost	Optional. Jump depart boost value. Integer expression from 0 - 100.
approBoost	Optional. Jump approach boost value. Integer expression from 0 - 100.

Return Values
   When parameters are omitted, the current PTPBoost settings are displayed.

Description
   PTPBoost sets the acceleration, deceleration and speed for small distance PTP motion. It is effective only when the motion distance is small. The PTPBoostOK function can be used to confirm whether or not a specific motion distance to the destination is small enough to be affected by **PTPBoost** or not.

   PTPBoost does not need modification under normal circumstances. Use **PTPBoost** only when you need to shorten the cycle time even if vibration becomes larger, or conversely when you need to reduce vibration even if cycle time becomes longer.

   When the **PTPBoost** value is large, cycle time becomes shorter, but the positioning vibration increases. When **PTPBoost** is small, the positioning vibration becoms smaller, but cycle time becomes longer. Specifying inappropriate **PTPBoost** causes errors or can damage the manipulator. This may degrade the robot, or sometimes cause the manipulator life to shorten.

   The **PTPBoost** value initializes to its default value when any one of the following is performed:

Controller Power On Motor On SFree, SLock Reset Stop button or Ctrl + C Key

See Also
   PTPBoost Function, PTPBoostOK

PTPBoost Statement Example

```
PTPBoost 50, 30, 30
```

PTPBoost Function

Returns the specified PTPBoost value.

F

Syntax
  PTPBoost(_paramNumber_**)**

Parameters
  paramNumber   Integer expression which can have the following values:
              1: boost value
              2: jump depart boost value
              3: jump approach boost value

Return Values
  Integer value from 0 - 100.

See Also
  PTPBoost Statement, PTPBoostOK

PTPBoost Function Example

```
    Print PTPBoost(1)
```

PTPBoostOK Function

Returns whether or not the PTP (Point to Point) motion from a current position to a target position is a small travel distance.

$\boxed{F}$

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.

| F |

Syntax

   (1) PTPTime(*destination, destArm, destTool*)
   (2) PTPTime(*start, startArm, startTool, destination, destArm, destTool*)

Parameters

start	Point expression for the starting position.
destination	Point expression for the destination position.
destArm	Integer expression for the destination arm number.
destTool	Integer expression for the destination tool number.
startArm	Integer expression for the starting point arm number.
startTool	Integer expression for the starting point tool number.

Return Values

Real value in seconds.

Description

Use PTPTime to calculate the time it would take for a point to point motion command (Go). Use syntax 1 to calculate time from the current position to the destination. Use syntax 2 to calculate time from a start point to a destination point.

The actual motion operation is not performed when this function is executed. The current position, arm, and tool settings do not change.

If the position is one that cannot be arrived at or if the arm or tool settings are incorrect, 0 is returned.

See Also

ATRQ, Go, PTRQ

PTPTime Function Example

```
Real secs

secs = PTPTime(P1, 0, 0, P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs

Go P1
secs = PTPTime(P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs
```

PTran Statement

Perform a relative move of one joint in pulses.

S

Syntax
    PTran *joint, pulses*

Parameters
joint	Integer expression representing which joint to move.
pulses	Integer expression representing the number of pulses to move.

Description
    Use **PTran** to move one joint a specified number of pulses from the current position.

See Also
    Go, JTran, Jump, Move

PTran Statement Example

```
PTran 1, 2000
```

PTRQ Statement

Displays the peak torque for the specified joint.

> **S**

Syntax
    PTRQ [*jointNumber*]

Parameters
    jointNumber   Optional. Integer expression representing the joint number.

Return Values
    Displays current peak torque values for all joints.

Description
    Use **PTRQ** to display the peak torque value for one or all joints since the PTCLR statement was executed.

    Peak torque is a real number from 0 to 1.

See Also
    ATRQ, PTCLR, PTRQ Function

PTRQ Statement Example

```
> ptclr
> go p1
> ptrq 1
      0.227
> ptrq
      0.227    0.118
      0.249    0.083
      0.000    0.000
>
```

PTRQ Function

Returns the peak torque for the specified joint.

F

Syntax
> **PTRQ**(*jointNumber*)

Parameters
> *jointNumber* Integer expression representing the joint number.

Return Values
> Real value from 0 to 1.

See Also
> ATRQ, PTCLR, PTRQ Statement

PTRQ Function Example
> This example uses the **PTRQ** function in a program:

```
Function DisplayPeakTorque
    Integer i

    Print "Peak torques:"
    For i = 1 To 4
        Print "Joint ", i, " = ", PTRQ(i)
    Next i
Fend
```

Pulse Statement

Moves the robot arm using point to point motion to the point specified by the pulse values for each joint.

> **>** **S**

Syntax
    (1) Pulse *J1, J2, J3, J4 ,* [*J5*] *,* [*J6*]
    (2) Pulse

Parameters

J1, J2, J3, J4	The pulse value for each of the first four joints. The pulse value has to be within the range defined by the Range instruction and should be an integer or long expression.
J5, J6	Optional. For use with 6-axis robots.

Return Values
    When parameters are omitted, the pulse values for the current robot position are displayed.

Description
    Pulse uses the joint pulse value from the zero pulse position to represent the robot arm position, rather than the orthogonal coordinate system. The Pulse instruction moves the robot arm using Point to Point motion.

    The Range instruction sets the upper and lower limits used in the Pulse instruction.

Note

Make Sure Path is Obstacle Free Before Using Pulse
    Unlike Jump, **Pulse** moves all axes simultaneously, including Z joint raising and lowering in traveling to the target position. Therefore, when using **Pulse**, take extreme care so that the hand can move through an obstacle free path.

Potential Errors

Pulse value exceeds limit:
    If the pulse value specified in Pulse instruction exceeds the limit set by the Range instruction, an error will occur.

See Also
    Go, Accel, Range, Speed, Pls, Pulse Function

Pulse Statement Example
    Following are examples on the 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.

$$\boxed{F}$$

Syntax
> **Pulse** (*J1, J2, J3, J4* , [*J5*] , [*J6*])

Parameters
> *J1, J2, J3, J4*   The pulse value for joints 1 to 4. The pulse value must be within the range defined by the Range instruction and should be an integer or long expression.
>
> *J5, J6*   Optional. For use with 6-axis robots.

Return Values
> A robot point using the specified pulse values.

See Also
> Go, JA, Jump, Move, Pulse Statement, XY

Pulse Function Example

```
Jump Pulse(1000, 2000, 0, 0)
```

QP Statement

Switches Quick Pause Mode On or Off and displays the current mode status.

$\boxed{>}$ $\boxed{S}$

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.

> S

Syntax
(1) **QPDecelR** *QPDecelR*
(2) **QPDecelR**

Parameters

QPDecelR   <u>Real value</u> 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 decleration 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 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.

>   **F**

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.

> S

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 decleration 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 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.

> F

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.

$\boxed{\textbf{S}}$

Syntax
   Quit { *taskIdentifier* | **All** }

Parameters
   taskIdentifier   Task name or integer expression representing the task number.
   A task name is the function name used in an Xqt statement or a function started
   from the Run window or Operator window. If an integer expression is used, the
   range is from 1 to 16 for normal tasks and from 257 to 261 for trap tasks.

   All   Specifies that all tasks should be terminated.

Description
   Quit stops the tasks that are currently being executed, or that have been temporarily suspended with
   Halt.

   Quit also stops the task when the specified task is NoPause task or NoEmgAbort task (special task
   using NoPause or NoEmgAbort at Xqt).
   Quit All stops all tasks including the tasks above.

See Also
   Exit, Halt, Resume, Xqt

Quit Example
   This example shows two tasks that are terminated after 10 seconds.

```
Function main
  Xqt winc1    'Start winc1 function
  Xqt winc2    'Start winc2 function
  Wait 10
  Quit winc1  'Terminate task winc1
  Quit winc2  'Terminate task winc2
Fend

Function winc1
  Do
    On 1; Wait 0.2
    Off 1; Wait 0.2
  Loop
Fend

Function winc2
  Do
    On 2; Wait 0.5
    Off 2; Wait 0.5
  Loop
Fend
```

RadToDeg Function

Converts radians to degrees.

$$\boxed{>} \quad \boxed{\textbf{F}}$$

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.

$$\boxed{\textbf{S}}$$

Syntax
  (1) Randomize *seedValue*
  (2) Randomize

Parameter
  seedValue   Specify a real value (0 or more) to be basis to retrieve a random number.

See Also
  Rnd Function

Randomize 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.

> S

Syntax

(1) Range *j1Min, j1Max, j2Min, j2Max, j3Min, j3Max, j4Min, j4Max, [j5Min, j5Max, j6Min, j6Max]*
(2) Range

Parameters

j1Min	The lower limit for joint 1 specified in pulses.
j1Max	The upper limit for joint 1 specified in pulses.
j2Min	The lower limit for joint 2 specified in pulses.
j2Max	The upper limit for joint 2 specified in pulses.
j3Min	The lower limit for joint 3 specified in pulses.
j3Max	The upper limit for joint 3 specified in pulses.
j4Min	The lower limit for joint 4 specified in pulses.
j4Max	The upper limit for joint 4 specified in pulses.
j5Min	Optional for 6-Axis robots. The lower limit for joint 5 specified in pulses.
j5Max	Optional for 6-Axis robots. The upper limit for joint 5 specified in pulses.
j6Min	Optional for 6-Axis robots. The lower limit for joint 6 specified in pulses.
j6Max	Optional for 6-Axis robots. The upper limit for joint 6 specified in pulses.

Return Values

Displays the current **Range** values when **Range** is entered without parameters

Description

Range specifies the lower and upper limits of each motor joint in pulse counts. These joint limits are specified in pulse units. This allows the user to define a maximum and minimum joint motion range for each of the individual joints. XY coordinate limits can also be set using the XYLim instruction.

The initial **Range** values are different for each robot. The values specified by this instruction remain in effect even after the power is switched off.

When parameters are omitted, the current **Range** values are displayed.

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

Potential Errors

Attempt to Move Out of Acceptable Range

If the robot arm attempts to move through one of the joint limits error an will occur

Axis Does Not Move

If the lower limit pulse is equal to or greater than the upper limit pulse, the joint does not move.

See Also

JRange, SysConfig, XYLim

Range Example

This simple example from the command window displays the current range settings and then changes them.

```
> range
-18205, 182045, -82489, 82489, -36864, 0, -46695, 46695
>
> range 0, 32000, 0, 32224, -10000, 0, -40000, 40000
>
```

Read Statement

Reads characters from a communications port.

S

Syntax
    Read #*portNumber*, *stringVar$*, *count*

Parameters

portNumber	Communications port to read from.
stringVar$	Name of a string variable that will receive the character string.
count	Maximum number of bytes to read.

See Also
    ChkCom, ChkNet, OpenCom, OpenNet, Write

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 communications port.

S

Syntax

>   **ReadBin** #*portNumber*, *var*
>   **ReadBin** #*portNumber, array(), count*

Parameters

portNumber	Communications port to read from.
var	Name of a byte, integer, or long variable that will receive the data byte.
array()	Name of a byte, integer, or long array variable that will receive the data byte. 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.

See Also

>   Write, WriteBin

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).

S

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 to the upper bound value.
>
>   The total available number of array elements for local and global preserve variables is 1000.
>
>   The total available number of array elements for global and module variables is 10000.
>
>   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
>   **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, Integer, Long, String

Real Example
>   The following example shows a simple program which declares some variables using **Real**.

```
Function realtest
  Real var1
  Real A(10)          'Single dimension array of real
  Real B(10, 10)      'Two dimension array of real
  Real C(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
```

RealPls Function

F

Syntax
  RealPls(jointNumber)

Parameters
  jointNumber   The specific joint for which to get the current pulse count.

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.

F

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.

$\boxed{F}$

Syntax

    RealTorque(*jointNumber*)

Parameters

    jointNumber   Specifies the joint number to acquire the torque instruction value using an expression or numeric value.

Return values

  Returns the real value (0-1) representing the proportion in the maximum torque on current power mode.

See also

  TC, TCSpeed, TCLim

RealTorque Function Example

```
Print "Current Z axis torqueinstruction value:", RealTorque(3)
```

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

Redim Statement

Redimension an array at run-time.

$$\boxed{\textbf{S}}$$

Syntax
> **Redim** [**Preserve**] *arrayName* **(***subscripts***)**

Parameters

Preserve	Optional. Specifies to preserve the previous contents of the array. If omitted, the array will be cleared.
arrayName	Name of the array variable; follows standard variable naming conventions. The array must have already been declared.
subscripts	New dimensions of the array variable. You must supply the same number of dimensions as when the variable was declared. The syntax is as follows

(*dim1,* [*dim2*], [*dim3*]**)**

dim1, *dim2*, *dim3* can be an integer expression from 0-2147483646.

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 to the upper bound value.

The total available number of array elements for local and global preserve variables is 100 for strings and 1000 for all other types.

The total available number of array elements for global and module variables is 1000 for strings and 10000 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
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.

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
```

Reset Statement

Resets the controller into an initialized state.

> S

Syntax
>   **Reset**

Description
>   **Reset** resets the following items:
>
>   Emergency Stop Status
>   Error status
>   Output Bits (All Outputs, except I/O assigned to remote output, set to Off; User can set EPSON RC+ to turn this feature off)
>   Current robot Speed, SpeedR, SpeedS (Initialized to default values)
>   Current robot Accel, AccelR, AccelS (Initialized to default values)
>   Current robot LimZ parameter (Initialized to 0)
>   Current robot Fine (Initialized to default values)
>   Current robot Power Low (Low Power Mode set to On)
>   Current robot PTPBoost (Initialized to default values)
>
>   For servo related errors, Emergency Stop status, and any other conditions requiring a Reset, no command other than Reset will be accepted. In this case first execute Reset, then execute other processing as necessary.
>
>   For example, after an emergency stop, first verify safe operating conditions, execute Reset, and then execute Motor On.
>
>   Critical error state will not be canceled by **Reset**.
>   When critical error occurs, turn Off the controller and solve the cause of the error.

Notes

Reset Option Switch
>   If the "Reset turns off outputs" controller preference is on, then when the **Reset** instruction is issued, all outputs will be turned off. This is important to remember when wiring the system such that turning the outputs off should not cause tooling to drop or similar situations.   See Setup | Controller | Preferences in the User's Guide for more details.

See Also
>   Accel, AccelS, Fine, LimZ, Motor, Off, On, PTPBoost, SFree, SLock, Speed, SpeedS

Reset Statement Example
>   This shows the Reset instruction issued from the command window.
>
>    ```
>    >reset
>    >
>    ```

Resume Statement

Continues a task which was suspended by the Halt instruction.

$\boxed{\textbf{S}}$

Syntax
   Resume { *taskIdentifier* | **All** }

Parameters
   taskIdentifier   Task name or integer expression representing the task number.
                    A task name is the function name used in an Xqt statement or a function started
                    from the Run window or Operator window. If an integer expression is used, the
                    range is from 1 to 16 for normal tasks and from 257 to 261 for trap tasks.
   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.

S

Syntax
  Return

Description
  The **Return** statement is used with the GoSub statement. The primary purpose of the **Return** statement is to return program control back to the instruction following the GoSub instruction which initiated the subroutine in the first place.

  The GoSub instruction causes program control to branch to the user specified statement line number or label. The program then executes the statement on that line and continues execution through subsequent line numbers until a **Return** instruction is encountered. The **Return** instruction then causes program control to transfer back to the line which immediately follows the line which initiated the GoSub in the first place. (i.e. the GoSub instruction causes the execution of a subroutine and then execution **Return**s to the statement following the GoSub instruction.)

Potential Errors

Return Found Without GoSub
  A **Return** instruction is used to "return" from a subroutine back to the original program which issued the GoSub instruction. If a **Return** instruction is encountered without a GoSub having first been issued then an error will occur. A stand alone **Return** instruction has no meaning because the system doesn't know where to **Return** to.

See Also
  OnErr, GoSub, GoTo

Return Statement Example
  The following example shows a simple function which uses a GoSub instruction to branch to a label called checkio and check the first 16 user inputs. Then the subroutine returns back to the main program.

```
Function main
    Integer var1, var2
    GoSub checkio
    On 1
    On 2
    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.

$\boxed{\textbf{F}}$

Syntax
> **Right$**(*string, count*)

Parameters

string	String variable or character string of up to 255 characters from which the rightmost characters are copied.
count	The number of characters to copy from *string* starting with the rightmost character.

Return Values
> Returns a string of the rightmost *count* characters from the character string specified by the user.

Description
> **Right$** returns the rightmost *count* characters of a string specified by the user. Right$ can return up to as many characters as are in the character string.

See Also
> Asc, Chr$, InStr, Left$, Len, Mid$, Space$, Str$, Val

Right$ Example
> The example shown below shows a program which takes a part data string as its input and splits out the part number, part name, and part count.

```
Function SplitPartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    PartNum$ = Left$(DataIn$, 10)

    DataIn$ = Right$(datain$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Mid$(DataIn$, 11, 10)

    PartCount = Val(Right$(dataIn$, 5))

Fend
```

```
Some other example results from the Right$ instruction from the Command
window.
> Print Right$("ABCDEFG", 2)
 FG

> Print Right$("ABC", 3)
 ABC
```

Rnd Function

Return a random number.

Syntax
   Rnd(*maxValue*)

Parameters
   maxValue   Real expression that represents the maximum return value.

Return Values
   Random real number from 0 to *range*.

Description
   Use Rnd to generate random number values.

See Also
   Int, Randomize

Rnd Function Example
Here's a Rnd example that generates a random number between 1 and 10.

```
Function main
    Real r
    Integer randNum

    Randomize
    randNum = Int(Rnd(9)) + 1
    Print "Random number is:", randNum
Fend
```

RobotInfo Function

Returns status information for the robot.

F

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	Resetable error has occurred
	2	&H4	Non-resetable error has occured
	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	0-31		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
	6-31		Undefined
4	N/A	0 - 16 -1	Number of tasks executing robot commands 0 = command executing from command window or macro -1 = no task is using the manipulator
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
	6-31		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

F

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	Undefined
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.

F

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$ 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$ Example

```
Print "The robot name is ", RobotName$
```

RobotSerial$ Function

Returns the robot serial number.

F

Syntax
>   **RobotSerial$**

Return Values
>   A string containing the robot serial number.

See Also
>   RobotInfo, RobotName$, RobotModel$

RobotSerial$ Example

```
Print "The robot serial number is ", RobotSerial$
```

RobotType Function

Returns the robot type.

F

Syntax
> **RobotType**

Return Values
> 1: JOINT
>
> 2: Cartesian
>
> 3: SCARA
>
> 5: 6-AXIS
>
> 6: RS series

See Also
> RobotModel$

RobotType Example

```
If RobotType = 3 Then
  Print "Robot type is SCARA"
EndIf
```

RSet$ Function

Returns the specified string with leading spaces added up to the specified length..

$\boxed{\textbf{F}}$

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.

$\boxed{\textbf{F}}$

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.)

Notes

Numeric Data Type:

    The numeric data (*number*) may be any valid numeric data type. **RShift** works with data types: Byte, Integer, and Real.

See Also

    And, LShift, Not, Or, Xor

RShift Example

    The example shown below shows a program which shows all the possible **RShift** values for an Integer data type starting with the integer set to 0.

```
Function rshiftst
  Integer num, snum, i
  num = 32767
  For i = 1 to 16
    Print "i =", i
    snum = RShift(num, 1)
    Print "RShift(32767, ", i, ") = ", snum
  Next i
Fend
```

    Some other example results from the **RShift** instruction from the command window.

```
> Print RShift(10,1)
5
> Print RShift(8,3)
1
> Print RShift(16,2)
4
```

RTrim$ Function

Returns a string equal to specified string without trailing spaces.

F

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"

EndIf
```

SafetyOn Function

Return the Safety Door open status.

$\boxed{\textbf{F}}$

Syntax

SafetyOn

Return Values

True if the Safety Door is Open, otherwise False.

Description

SafetyOn function is used only for NoPause task or NoEmgAbort task (special task using NoPause or NoEmgAbort at Xqt).

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.

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.

```
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.

> **>** **S**

Syntax
> **SavePoints** *filename*

Parameters
> *fileName*   String expression containing the file into which points will be stored. The specified *fileName* will have the extension .PTS appended to the end so no extension is to be specified by the user.

Description
> **SavePoints** saves points to the specified file. The specified *fileName* will have the extension .PTS appended to the end so no extension is to be specified by the user.
>
> Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

Potential Errors

Bad File name
> If a file name is entered which has spaces in the name, or other bad file name characteristics an error will be issued.

See Also
> ClearPoints, LoadPoints

SavePoints Statement Example

```
ClearPoints
For i = 1 To 10
    P(i) = XY( i, 100, 0, 0 )
Next i
SavePoints "TEST.PTS"
```

Select...Send

Executes one of several groups of statements, depending on the value of an expression.

$\boxed{\textbf{S}}$

Syntax
> **Select** *selectExpr*
>   **Case** *caseExpr*
>   *statements*
>   **[Case** *caseExpr*
>   *statements* **]**
>   **[Default**
>   *statements* **]**
>
>   **Send**

Parameters

selectExpr	Any numeric or string expression.
caseExpr	Any numeric or string expression that evaluates to the same type as *selectExpr*.
statements	One or more valid SPEL$^+$ statements or multi-statements.

Description

If any one *caseExpr* is equivalent to *selectExpr*, then the statements after the Case statement are executed. After execution, program control transfers to the statement following the Send statement.

If no *caseExpr* is equivalent to *selectExpr*, the Default statements are executed and program control transfers to the statement following the Send statement.

If no *caseExpr* is equivalent to *selectExpr* and Default is omitted, nothing is executed and program control transfers to the statement immediately following the Send statement.

selectExpr and *caseExpr* may include constants, variables, and logical operators that use And, Or and Xor.

See Also

If...Then...Else

Select Example

Shown below is a simple example for Select...Send:

```
Function Main
  Integer I
  For i = 0 To 10
    Select I
      Case 0
          Off 1;On 2;Jump P1
      Case 3
          On 1;Off 2
          Jump P2;Move P3;On 3
      Case 7
          On 4
      Default
          On 7
    Send
  Next
Fend
```

Sense Statement

Specifies and displays input condition that, if satisfied, completes the Jump in progress by stopping the robot above the target position.

$\boxed{\textbf{>}}$ $\boxed{\textbf{S}}$

Syntax
>   **Sense** [*inputCondition*]

Parameters
>   *inputCondition*   The following functions and operators may be used in the *inputCondition*:

>>>>   **Functions** : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemW, Ctr

>>>>   **Operators** : 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 execting a motion command using Sense modificators.

See Also
>   In, InW, Out, OutW, JS, Jump, Jump3, Jump3CP, MemIn, MemSw, Stat, Sw

Sense Statement Example

This is a simple example on the usage of the Sense instruction.

```
Function test
   .
   .
TrySense:
    Sense Sw(1) = 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.

$\boxed{\textbf{S}}$

Syntax

    SetCom *#portNumber,* [*baud*], [*dataBits*], [*stopBits*] , [*parity*] , [*terminator*] , [*HWFlow*], [*SWFlow*] , [*timeOut*]

Parameters

portNumber	Specifies which RS232 port to set parameters for. Valid values are 1-8.
baud	Optional. Specifies the baud rate. Valid values are:

                             110   2400   19200

                             300   4800   38400

                             600   9600   56000

                             1200 14400 115200

                             (Default: 9600)

dataBits	Optional. Specifies the number of data bits per character. Valid values are **7** and **8**.
stopBits	Optional. Specifies the number of stop bits per character. Valid values are **1** and **2**.
parity	Optional. Specifies the parity. Valid values are **O** (Odd), **E** (Even), and **N** (None).
terminator	Optional. Specifies the line termination characters. Valid values are **CR**, **LF**, **CRLF**.
HWFlow	Optional. Specifies hardware control. Valid values are **RTS** and **NONE**.
SWFlow	Optional. Specifies software control. Valid values are **XON** and **NONE**.
timeOut	Optional. Specifies the maximum time for transmit or receive in seconds. If this value is 0, then there is no time out.

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.

    Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also

    OpenCom, CloseCom, SetNet

SetCom Example

```
SetCom #1, 9600, 8, 1, N, CRLF, NONE, NONE, 0

SetCom #2, 4800
```

SetIn Statement

For Virtual IO, sets specified input port (8 bits) to the specified value.

$\boxed{\textbf{>}}$ $\boxed{\textbf{S}}$

Syntax
    SetIn *portNumber*, *value*

Parameters
    portNumber Integer expression representing the input port number.
    value   Integer expression between 0 – 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 Function Example

```
> setin 0, 1  ' Sets the first bit of port 0 to On.
```

SetInReal Statement

For Virtual IO, sets specified input port (2 words (32 bits)) as 32 bits floating-point data (IEEE754 compliant).

> **S**

Syntax

SetInReal *portNumber*, *value*

Parameters

portNumber Integer expression representing the input port word.

value Specifies Real type value.

Description

When Virtual I/O is available, set the value of 2 input words from the input word port specified by *portNumber* as IEEE754 32 bits Real type value.

Input word label can be used for the *portNumber* parameter.

When Virtual I/O is not available, SetInReal will be an error.

See Also

SetSw, SetIn, SetInW

SetInReal Example

```
> SetInReal 0, 1.23  'Set a Real value (1.23) to word 0,1
```

SetInW Statement

For Virtual IO, sets specified input word (16 bits) to the specified value.

$\boxed{\textbf{>}}\ \boxed{\textbf{S}}$

Syntax
    SetInW *portNumber*, *value*

Parameters
    portNumber Integer expression representing the input port number.
    value   Number between 0 – 65535 to set the specified word to.

Description
    SetInW provides the ability to set up to 16 bits of virtual inputs at once.

See Also
    SetSw, SetIn

SetInW Function Example

```
> setinw 0, 1  ' Sets the first bit of word 0 to On.
```

SetNet Statement

Sets parameters for a TCP/IP port.

$\boxed{\textbf{S}}$

Syntax

SetNet #*portNumber, hostAddress, TCP_IP_PortNum, terminator, SWFlow, timeout*

Parameters

portNumber	Specifies which port to set parameters for. Valid values are 201 - 208.
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.

Description

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also

OpenNet, CloseNet, SetCom

SetNet Example

```
SetNet #201, "192.168.0.1", 2001, CRLF, NONE, 0
```

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 Function Example

```
> setsw 2, on  ' Sets the 2nd input bit to On.
```

SFree Statement

Removes servo power from the specified servo axis.

> S

Syntax
> **SFree** *jointNumber* [*, jointNumber,...*]

Parameters
> *jointNumber*   An integer expression representing a servo joint number.

Description
> **SFree** removes servo power from the specified servo joints. This instruction is used for the direct teaching or the part installation by partially de-energizing a specific joint. To re-engage a joint execute the SLock instruction or Motor On.

Notes

SFree Sets Some System Items back to Their Initial State:
> SFree, for safety purposes, initializes parameters concerning the robot arm speed (Speed and SpeedS), acceleration (Accel and AccelS) and the LimZ parameter.

Notes

SFree and its Use with the Z Joint 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
> All joints of the 6-axis robots have an electromagnetic brake. The brake can be released using the Brake command with the motor off. In the motor off state, SFree is not valid. If you execute SFree with the motor on, an electromagnetic brake will be on. You cannot move any joint by hand using SFree.

Executing motion commands while joints are in SFree state
> Attempting to execute a motion command while in the SFree condition will cause an error in the controller's default state. However, to allow motion while 1 or more of the axes are in the SFree state, turn on the "Allow Motion with one or more axes free" controller preference. (This preference can be set from the Setup | Controller | Preferences EPSON RC+ 5.0.)

See Also
> Brake, LimZ, Motor, SFree Function, SLock

SFree Statement Example
> This is a simple example on the usage of the SFree instruction. The Motion with SFree controller preference must be enabled for this example to work.

```
Function GoPick
   Speed pickSpeed
   SFree 1, 2      'Release the excitation of J1 and J2,
                   'and control the Z and U joints for part installation.
   Go pick
   SLock 1, 2      'Restore the excitation of J1 and J2.
Fend
```

SFree Function

Returns SFree status for a specified joint.

$\boxed{\text{F}}$

Syntax
   SFree(*jointNumber*)

Parameters
   jointNumber   Integer expression representing the joint number to check.

Return Values
   True if the joint is free, False if not.

See Also
   SFree Statement

SetFree Example

```
If SFree(1) Then
    Print "Joint 1 is free"
EndIf
```

Sgn Function

Determines the sign of the operand.

[F]

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
>
```

Signal Statement

Send a signal to tasks executing WaitSig.

Syntax
>   **Signal** *signalNumber*

Parameters
>   *signalNumber*   Signal number to transmit. Range is 0 to 15.

Description
>   Signal can be used to synchronize multi-task execution.
>
>   Previous signals issued before WaitSig is executed are ignored.

See Also
>   WaitSig

Signal Statement Example

```
Function Main
   Xqt 2, SubTask
   Call InitSys
   Signal 1

Fend

Function SubTask
   WaitSig 1

Fend
```

Sin Function

Returns the sine of a numeric expression.

$\boxed{\textbf{F}}$

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.

$\boxed{\mathsf{S}}$

Syntax
SingularityAngle {Angle}

Parameter

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.

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 5 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 Example
```
SingularityAngle 7.0   'Sets the singularity neighborhood angle at 7 degrees
```

SingularityAngle Function

Returns the SingularityAngle setting value.

F

Syntax
SingularityAngle

Return value
Returns the singularity neighborhood angle (Unit: deg).

See Also
AvoidSingularity, SingularityAngle, SingularitySpeed, SingularitySpeed Function

SingularityAngle Function Example

```
Real currSingularityAngle
currSingularityAngle = SingularityAngle
```

SingularitySpeed Statement

Sets the singularity neighborhood angular velocity necessary for the singularity avoiding function. **S**

Syntax
SingularitySpeed {Angular velocity}

Parameter

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.

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 25 %. 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.

F

Syntax

SingularitySpeed

Return Value

Returns the singularity neighborhood angular velocity (Unit: %).

See Also

SingularitySpeed, SingularityAngle, AvoidSingularity

SingularitySpeed Function Example

```
Real currSingularitySpeed
currSingularitySpeed = SingularitySpeed
```

SLock Statement

Restores servo power from servo free condition for the specified servo axis.

$\boxed{>}$ $\boxed{S}$

Syntax
>   **SLock** *jointNumber* [*, jointNumber,...*]

Parameters
>   *jointNumber*   The servo joint number.

Description
>   **SLock** restores servo power to the specified servo joint, which was de-energized by the SFree instruction for the direct teaching or part installation.
>
>   If the joint number is omitted, all joints are engaged.
>
>   Engaging the 3rd joint (Z) causes the brake to release.
>
>   To engage all axes, Motor On may be used instead of **SLock**.
>
>   Executing **SLock** while in Motor Off state will cause an error.

Notes

SLock Sets Some System Items back to Their Initial State:
>   SLock, for safety purposes, initializes parameters concerning the robot arm feed speed (Speed and SpeedS), acceleration (Accel and AccelS) and the LimZ parameter.

See Also
>   Brake, LimZ, Reset, SFree

SLock Example
>   This is a simple example on the usage of the SLock instruction. The Motion with SFree controller preference must be enabled for this example to work.

```
Function test
   .
   .
   .
   SFree 1, 2    'Release the excitation of J1 and J2,
                 'and control the Z and U joints for part installation.
   Go P1
   SLock 1, 2    'Restore the excitation of J1 and J2.
   .
   .
   .
Fend
```

SoftCP Statement

Specifies the SoftCP motion mode.

$\boxed{\textbf{S}}$

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.

    SoftCP will be set to Off in the following cases:
        Controller startup
        Reset
        All task stop
        Switching the Auto / Programming operation mode
        Motor On
        SFree, SLock

See Also
    SoftCP Function

SoftCP Statement Example

```
SoftCP On
Move P1
Move P2
SoftCP Off
```

SoftCP Function

Returns the status of SoftCP moton mode.

F

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.

<boxed>F</boxed>

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.

> **S**

Syntax
(1) Speed *percent*, [*departSpeed*], [*approSpeed*]
(2) Speed

Parameters

percent	Integer expression between 1-100 representing the arm speed as a percentage of the maximum speed.
departSpeed	Integer expression between 1-100 representing the depart motion speed for the Jump instruction. Optional. Available only with Jump command.
approSpeed	Integer expression between 1-100 representing the approach motion speed for the Jump instruction. Optional. 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 Power On
Motor On
SFree, SLock
Reset
Stop button or Ctrl+C Key
```

In Low Power Mode, the effective speed setting is lower than the default value. If a higher speed is specified directly (from the 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 Statement

Speed Function Example

```
        Integer savSpeed

        savSpeed = Speed(1)
        Speed 50
        Go pick
        Speed savSpeed
    Fend
```

SpeedR Statement

Sets or displays the tool rotation speed for CP motion when ROT is used.

> S

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 Power On
Motor On
SFree, SLock
Reset
Stop button or Ctrl + C Key

See Also

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

SpeedR Statement Example

```
SpeedR 200
```

SpeedR Function

Returns tool rotation speed value.

> F

Syntax

SpeedR

Return Values

Real value in degrees / second

See Also

AccelR Statement, SpeedR Statement

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.

`>` **`S`**

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: 1 to 2000

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 robot manual for the default SpeedS values for your robot model. This is the initial **SpeedS** value set up automatically by the controller each time main power is turned on.

   The **SpeedS** value initializes to its default value when any one of the following is performed:

   ```
   Controller Power On
   Motor On
   SFree, SLock
   Reset
   Stop button or Ctrl + C Key
   ```

   In Low Power Mode, the effective **SpeedS** setting is lower than the default value. If a higher speed is specified directly (from the 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.

   If **SpeedS** is executed when the robot is in low power mode, the following message is displayed.
   The following example shows that the robot will move at the default speed (50) because it is in Low Power Mode even though the speed setting value by **SpeedS** is 800.

   ```
   > SpeedS 800
   Low Power State: SpeedS is limited to 50
   >
   > SpeedS
   Low Power State: SpeedS is limited to 50
     800
   >
   ```

See Also

AccelS, Arc, Jump3, Move, Speed

SpeedS Example

SpeedS can be used from the 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
        Go P0
        Move P1
        SpeedS fast
        Go P0
        Move P1
    Next i
Fend
```

From the command window the user can also set **SpeedS** values.

```
> speeds 1000
> speeds 500
> speed 30      'set point to point speed
> go p0         'point to point move
> speeds 100    'set straight line speed in mm/Sec
> move P1       'move in straight line
```

SpeedS Function

Returns the current SpeedS setting.

F

Syntax
> **SpeedS** [(*paramNumber*)]

Parameters
> *paramNumber*   Optional. Integer expression specifying which SpeedS value to return.
>
>   1: CP speed
>
>   2: Jump3 depart speed
>
>   3: Jump3 approach speed

Return Values
> Real number, in mm/sec

See Also
> SpeedS Statement

SpeedS Example

```
Real savSpeeds

savSpeeds = SpeedS

Print "Jump3 depart speed = ", SpeedS(2)
```

SPELCom_Event Statement

Generates a user event for a VB Guide SPELCom control used in a host program.

`>` `S`

Syntax

> **SPELCom_Event** *eventNumber* [*, msgArg1, msgArg2, msgArg3,...*]

Parameters

> *eventNumber*   An integer expression whose value is from 1000 - 32767.
>
> *msgArg1, msgArg2, msgArg3...*   Optional. Each message argument can be either a number, string literal, or a variable name.

Description

> This instruction makes it easy to send real time information to another application using the SPELCom ActiveX control provided in the VB Guide option. For example, you can update parts count, lot number, etc. by sending an event to your host program.

Note

> This command will only work if the VB Guide option is installed.

See Also

> VB Guide Manual

SPELCom_Event Example

In this example, a SPEL$^+$ task sends cycle data to the host program.

```
Function RunParts
    Integer cycNum

    cycNum = 0
    Do
        ...
        ...
        cycNum = cycNum + 1
        Spelcom_Event 3000, cycNum, lot$, cycTime
    Loop
Fend
```

Sqr Function

Computes the non-negative square root value of the operand.

【F】

Syntax
> **Sqr**(*Operand*)

Parameters
> *Operand*   A real expression.

Return Values
> Square root value.

Description
> The Sqr function returns the non-negative square root value of the operand.

Potential Errors

Negative operand
> If the operand is or has a negative numeric value, an error will occur.

See Also
> Abs, And, Atan, Atan2, Cos, Int, Mod, Not, Or, Sgn, Sin, Str$, Tan, Val, Xor

Sqr Function Example
> This is a simple 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
```

Stat Function

Returns the execution status information of the controller.

F

Syntax
   Stat*(address)*

Parameters

   address   Defines which status bits to check.

Return Values
   Returns a 4 byte value that presents the status of the controller. Refer to table below.

Description
   The **Stat** instruction returns information as shown in the table below:

Address	Bit		Controller Status Indicated When Bit is On
0	0 to 15	&H1 to &H8000	Task 1 is being executed (Xqt) or in Halt State to Task 16 is being executed (Xqt) or in Halt State
	16	&H10000	Task(s) is being executed
	17	&H20000	Pause condition
	18	&H40000	Error Condition
	19	&H80000	Teach mode
	20	&H100000	Emergency Stop Condition
	21	&H200000	Low Power Mode (Power Low)
	22	&H400000	Safe Guard Input is Closed
	23	&H800000	Enable Switch is Open
	24	&H1000000	Undefined
	25	&H2000000	Undefined
	26-31		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
	14-31		Undefined

See Also

EStopOn Function, TillOn Function, PauseOn Function, SafetyOn Function

Stat Example

```
Function StatDemo

    rbt1_sts = RShift((Stat(0) And &H070000), 16)
    Select TRUE
        Case (rbt1_sts And &H01) = 1
            Print "Tasks are running"
        Case (rbt1_sts And &H02) = 2
            Print "Pause Output is ON"
        Case (rbt1_sts And &H04) = 4
            Print "Error Output is ON"
    Send
Fend
```

Str$ Function

Converts a numeric value to a string and returns it.

F

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)

$$\boxed{S}$$

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 to the upper bound value.

                         The total available number of array elements for local and global preserve variables is 100.

                         The total available number of array elements for global and module variables is 1000.

                         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
    The **String** statement is used to declare variables of type String. String variables can contain up to 255 characters. Local variables should be declared at the top of a function. Global and module variables must be declared outside of functions.

    String Operators
    The following operators can be used to manipulate string variables:

        + Merges character strings together. Can be used in the assignment statements for string variables or in the Print instruction.
           Example: name$ = fname$ + " " + lname$

        = Compares character strings. True is returned only when the two strings are exactly equal, including case.
           Example: If temp1$ = "A" Then GoSub test

        < > Compares character strings. True is returned when one or more characters in the two strings are different.
           Example: If temp1$ <> "A" Then GoSub test

Notes

Variable Names Must Include "$" Character:
    Variables of type String must have the character "$" as the last character in the variable name.

See Also
    Boolean, Byte, Double, Global, Integer, Long, Real

String Example

```
String password$
String A$(10)          'Single dimension array of string
String B$(10, 10)      'Two dimension array of string
String C$(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)

$\boxed{\textbf{F}}$

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.

| **S** |

Syntax

> **SyncLock** *syncID* [, *timeOut*]

Parameters

syncID	Integer expression representing signal number to receive. Range is from 0 to 15.
timeOut	Optional. Real expression representing the maximum time to wait for lock.

Description

Use **SyncLock** to lock use of a common resource so that only one task at a time can use it. When the task is finished with the resource, it must call SyncUnlock to release the lock so other tasks can use it.

A task can only unlock a syncID that it previously locked.

A task must execute SyncUnlock to release the lock. If the task is quit, then no other task can use the lock until all tasks are aborted.

When SynLock is second consecutive used to a same signal number, an error occurs.

If the *timeOut* parameter is used, then the **Tw** function must be used to check if the lock was successful.

See Also

Signal, SyncLock, Tw, Wait, WaitPos

SyncLock Example

The following example uses SyncLock and SyncUnlock to allow only one task at a time to write a message to a communcation 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.

S

Syntax
> **SyncUnlock** *syncID*

Parameters
> *syncID*   Integer expression representing signal number to receive. Range is from 0 to 15.

Description
> Use **SyncUnlock** to unlock a sync ID previously locked with SyncLock.
>
> A task can only unlock a syncID that it previously locked.

See Also
> Signal, SyncLock, Wait, WaitPos

SyncUnlock Example

```
Function Main

    Xqt task
    Xqt task
    Xqt task
    Xqt task
Fend

Function task
  Do
    SyncLock 1
    Print "resource 1 is locked by task", MyTask
    Wait .5
    SyncUnlock 1
  Loop
Fend
```

SysConfig Command

Displays system configuration parameter.

S

Syntax
   SysConfig

Return Values
   Returns system configuration parameter.

Description
   Display current configurated 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 te 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
'   VB Guide

' 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 Board:
'  Installed
'  Type: PROFIBUS

' RS232C Boards:
'  1: Installed
'  2: None installed
```

SysConfig 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).
Error codes or warning codes of controller are the error codes or warning codes dispayed on the on the seven-segment LED.
When there are no errors or warnings, the return value will be 0.

See Also

ErrMsg$, ErrOn, 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 Occurence

As this program, finish the task promply 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.

$\boxed{\textbf{F}}$

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.

F

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.

F

Syntax
> **TaskDone** (*taskIdentifier*)

Parameters
> *taskIdentifier*   Task name or integer expression representing the task number.
> A task name is the function name used in an Xqt statement or a function started from the Run window or Operator window. If an integer expression is used, the range is from 1 to 16 for normal tasks and from 257 to 261 for trap tasks.

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.

$\boxed{\textbf{F}}$

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. If an integer expression is used, the range is from 1 to 16 for normal tasks and from 257 to 261 for trap tasks.
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	Undefined

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	-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 occured 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) = 3 Then
  Print "Task 1 is runnning"
Else
  Print "Task 1 is not running"
EndIf
```

TaskInfo$ Function

Returns text information for a task.

$\boxed{\textsf{F}}$

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. If an integer expression is used, the range is from 1 to 16 for normal tasks and from 257 to 261 for trap tasks.
>
> *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.

F

Syntax

   TaskState(*taskIdentifier*)

Parameters

  taskIdentifier   Task name or integer expression representing the task number.
                   A task name is the function name used in an Xqt statement or a function started
                   from the Run window or Operator window. If an integer expression is used, the
                   range is from 1 to 16 for normal tasks and from 257 to 261 for trap tasks.

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.

$\boxed{\textbf{S}}$

Syntax
   TaskWait (*taskIdentifier*)

Parameters
   taskIdentifier   Task name or integer expression representing the task number.
                        A task name is the function name used in an Xqt statement or a function started
                        from the Run window or Operator window. If an integer expression is used, the
                        range is from 1 to 16 for normal tasks and from 257 to 261 for trap tasks.

See Also
   TaskDone, TaskState

TaskWait Statement Example

```
Xqt 2, conveyor
TaskWait conveyor
```

TC Statement

Returns the torque control mode setting and current mode.

$\boxed{\textbf{S}}$

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 are omitted, turns 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 and speed control in TCLim and TCSpeed.
>   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 power ON |
> | Motor On |
> | SFree, SLock |
> | Reset |
> | STOP button or Ctrl + C key |

See Also
>   TCLim, TCSpeed

TC Example

```
Speed 5
Go ApproachPoint

'Set the Z axis torque limit to 20 %
TCLim -1, -1, 20, -1
'Set the speed in torque control to 5 %
TcSpeed 5

TC On
Go ContactPoint
Wait 3
Go ApproachPoint
TC Off
```

TCLim Statement

Specifies the torque limit of each joint for the torque control mode.

$\boxed{\textbf{S}}$

Syntax

    TCLim [*j1Torque limit*, *j2Torque limit*, *j3Torque limit*, *j4Torque limit*, [*j5Torque limit*], [*j6Torque 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.

Return vlaues

    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 power ON
Motor On
SFree, SLock
Reset
STOP button or Ctrl + C key
```

See Also

    TC, TCLim Function, TCSpeed

TCLim Example

```
Speed 5
Go ApproachPoint

'Set the Z axis torque limit to 20 %
TCLim -1, -1, 20, -1
'Set the speed in torque control to 5 %
TcSpeed 5

TC On
Go ContactPoint
Wait 3
Go ApproachPoint
TC Off
```

TCLim Function

Returns the torque limit of specified joint.

F

Syntax
> **TCLim** (*jointNumber*)

Parameters
> *jointNumber*   Specifies the joint number to retrieve the torque limit from using an expression or numeric value.

Return values
> Returns the integer number representing the current torque limit (1 - 100). -1 means the torque limit is invalid.

See Also
> TC, TCLim, TCSpeed

TCLim Fuction Example

```
Print "Current Z axis torque limit:", TCLim(3)
```

TCPSpeed Function

Returns the calculated current tool center point (TCP) speed.

$$\boxed{\text{F}}$$

Syntax

    TCPSpeed

Return Values

    Real value containing the calculated current tool center point speed in mm/second.

Description

    Use **TCPSpeed** to get the calculated current speed of the tool center point in mm/second when executing a CP (Continuous Path) motion command. CP motion commands include Move, TMove, Arc, Arc3, CVMove, and Jump3CP. This is not the actual tool center point speed. It is the speed that the system has calculated for the tool center point at the time the function is called.

    The motor compliance lag is excluded from the calculation.
    If the robot is executing a PTP (Point to Point) motion command, this function returns 0.

See Also

    AccelS, CurPos, InPos, SpeedS

TCPSpeed Function Example

```
Function MoveTest
  AccelS 4000, 4000
  SpeedS 200
  Xqt ShowTCPSpeed
  Do
    Move P1
    Move P2
  Loop
Fend

Function ShowTCPSpeed
  Do
    Print "Current TCP speed is: ", TCPSpeed
    Wait .1
  Loop
Fend
```

TCSpeed Statement

Specifies the speed limit in the torque contol.

$\boxed{\text{F}}$

Syntax
   TCSpeed [*speed*]

Parameters
   speed   Specifies the proportion to the maximum speed (1 -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 power ON Motor On SFree, SLock Reset STOP button or Ctrl + C key

See Also
   TC, TCLim, TCSpeed Function

TCSpeed 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 %
TcSpeed 5

TC On
Go ContactPoint
Wait 3
Go ApproachPoint
TC Off
```

TCSpeed Function

Returns the speed limit in the torque control.

Syntax
    TCSpeed

Return values
    Returns the integer number (1 - 100) representing the current speed limit.

See Also
    TC, TCSpeed, TCLim

TCSpeed Example

```
Integer var
var = TCSpeed
```

TGo Statement

Executes Point to Point relative motion, in the current tool coordinate system.

> S

Syntax

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

Parameters

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

Description

Executes point to point relative motion in the current tool coordinate system.

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

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

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

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

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

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

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, Point Assignment, Speed, Till, TMove, Tool

TGo Example

```
> TGo XY(100, 0, 0, 0)   'Move 100mm in X direction
                          '(in the tool coordinate system)
Function TGoTest

  Speed 50
  Accel 50, 50
  Power High

  Tool 0
  P1 = XY(300, 300, -20, 0)
  P2 = XY(300, 300, -20, 0) /L

  Go P1
  Print 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 input condition that, if satisfied, completes the motion command (Jump, Go, Move, etc.) in progress by decelerating and stopping the robot at an intermediate position.

> S

Syntax

  Till [*inputCondition*]

Parameters

  inputCondition   The following functions and operators may be used in the *inputCondition*:

  Functions : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemW, Ctr

  Operators : 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.

See Also

  Find, Go, In, InW, Oport, Out, OutW, Jump, MemIn, MemSw, Move, Stat, Sw, TillOn

Till 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.

F

Syntax
    TillOn

Return Values
    True if the Till condition occurred in the previous motion command using Till.

Description
    TillOn returns True if Till condition occurred.

    TillOn is equivalent to ((Stat(1) And 2) <> 0).

See Also
    EStopOn, SafetyOn, Sense, Stat, Till

TillOn Function Example

```
Go P0 Till Sw(1) = On
If TillOn Then
    Print "Till condition occurred during move to P0"
EndIf
```

Time Command

Specifies and displays the current time.

>

Syntax
> **(1) Time** *hours, minutes, seconds*
> **(2) Time**

Parameters

hours	The hour of the day to set the controller clock to. This is an function expression between 1 and 24.
minutes	The minute of the day to set the controller clock to. This is an function expression between 0 and 59.
seconds	The second of the day to set the controller clock to. This is an function expression between 0 and 59.

Return Values
> If parameters are omitted, displays the current time in 24 hour format.

Description
> Specifies the current time.
>
> The time specification is in 24 hour format.

See Also
> Date, Time$

Time Example

```
> Time
The current time is 10:15:32

> Time 1,5,0
> Time
The current time is 1:05:15
```

Time Function

Returns the controller accumulated operating time.

F

Syntax
> **Time**(*unitSelect*)

Parameters
> *unitSelect*   An integer number ranging from 0-2. This integer specifies which unit of time the
> controller returns:
>
> > 0: hours
> >
> > 1: minutes
> >
> > 2: seconds

Description
> Returns the controller accumulated operating time as an integer.

See Also
> Hour

Time Function Example
> Shown below are a few examples from the 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 system time.

F

Syntax
> **Time$**

Return Values
> A string containing the current time in 24 hour format *hh:mm:ss*.

Description
> **Time$** is used to get the system time in a program statement. To set the system time, you must use the Time command from the command window.

See Also
> Date, Date$, Time

Time$ Example

```
Print "The current time is: ", Time$
```

TLClr Statement

Clears (undefines) a tool coordinate system.

> S

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

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also

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

TLClr Example

```
TLClr 1
```

TLDef Function

Returns tool definition status.

$\boxed{>}$ $\boxed{\text{F}}$

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 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.

> S

Syntax

(1) TLSet *toolNum***,** *toolDefPoint*
(2) TLSet *toolNum*
(3) TLSet

Parameters

toolNum   Integer number from 1-15 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
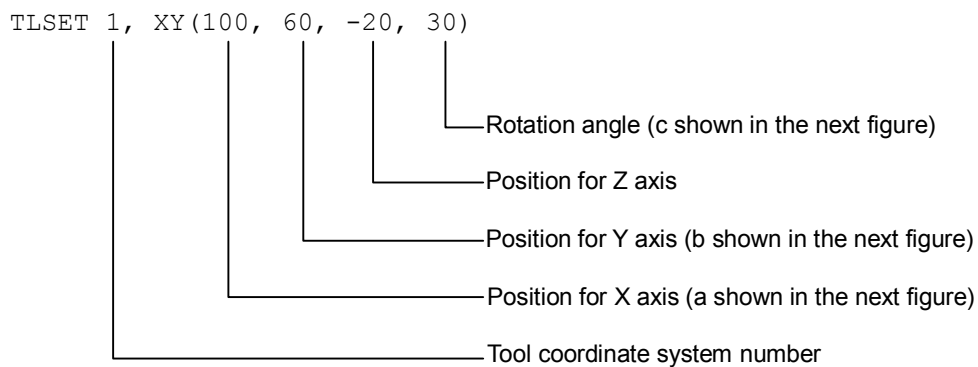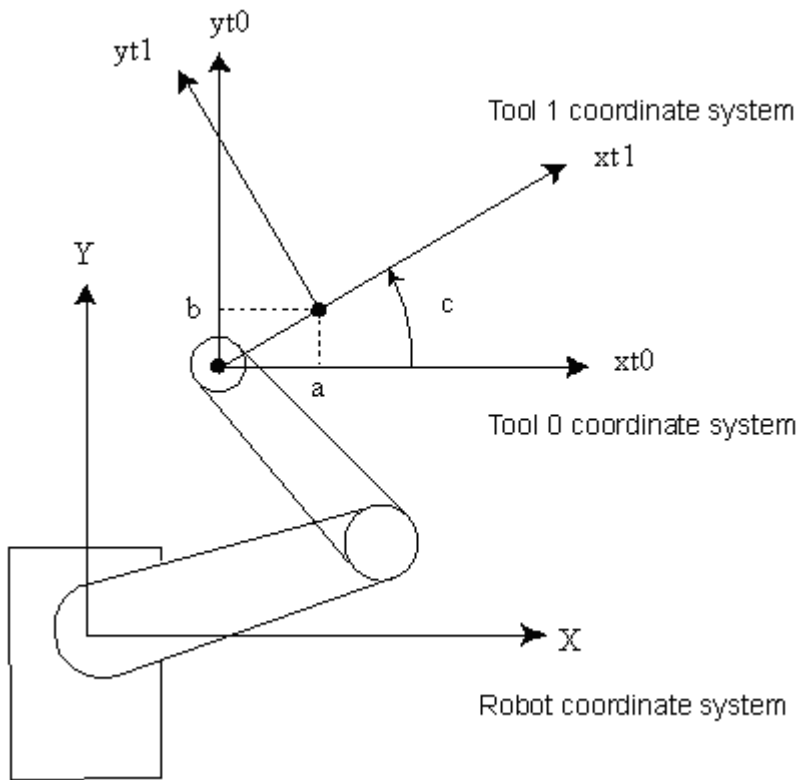
TISet for 6-Axis robots

The origin of Tool 0 is the flange side of the sixth joint. When all joints are at the 0 degree position, the Tool 0 coordinate system's X axis is aligned with the robot coordinate system's Z axis, the Y axis is aligned with the robot coordinate system's X axis, and the Z axis is perpendicular to the flange face, and is aligned with the robot coordinate system's Y axis, as shown in the figure below:
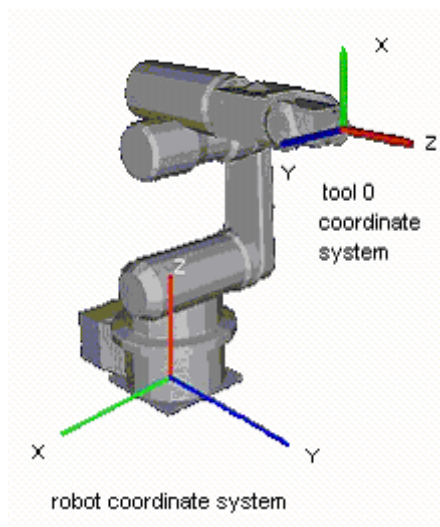
Tool 0 coordinate systems are defined for ceiling and wall mounted robots as shown in the figures below.
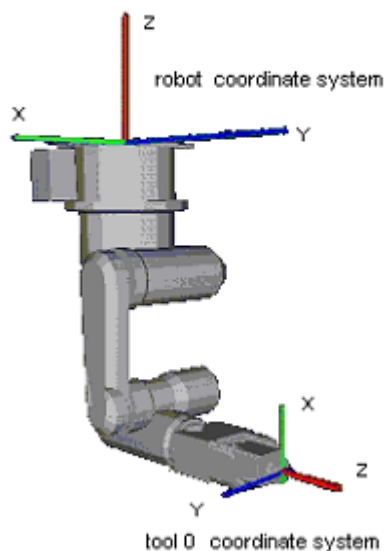
Ceiling mounting   Side (wall) mounting
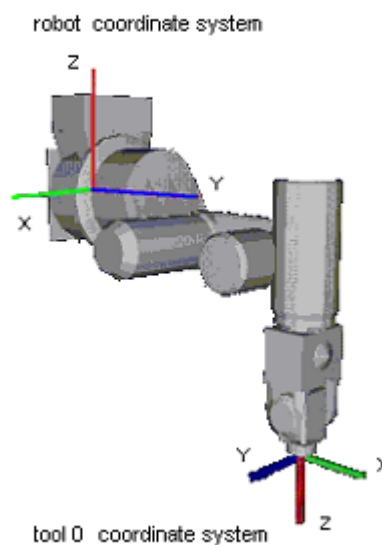
Point data is stored to the Compact Flash inside the Controller. When you execute this command, the data is written to the Compact Flash. If a data is written to the Compact Flash frequently, it may shorten the Compact Flash life. Using this command only when saving the point data is recommended.

Notes

TLSet values are maintained
The **TLSet** values are preserved. Use TLClr to clear a tool definition.

See Also
Tool, Arm, ArmSet, Point Expression, TLClr, TLSet Function

TLSet Example
The example shown below shows a good test which can be done from the 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.

<div style="float: right; border: 1px solid black; padding: 4px;">**F**</div>

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.

> **>** **S**

Syntax
TMOut *seconds*

Parameters
seconds   Real expression representing the number of seconds until a timeout occurs. Valid range is 0-2147483 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 Example

```
TMOut 5
Wait MemSw(0) = On
```

TMove Statement

Executes linear interpolation relative motion, in the current tool coordinate system $\boxed{>}$ $\boxed{S}$

Syntax
TMove *destination* [**ROT**] [**CP**] [*searchExpr*] [*!...!*]

Parameters

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

Description

Executes linear interpolated relative motion in the current tool coordinate system.

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

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

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

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

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

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

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

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

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

Notes

Using TMove with CP

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

See Also

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

TMove Example

```
> TMove XY(100, 0, 0, 0)     'Move 100mm in the X
                             'direction (in the tool coordinate system)
Function TMoveTest

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

  Tool 0
  P1 = XY(300, 300, -20, 0)
  P2 = XY(300, 300, -20, 0) /L

  Go P1
  Print 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

Timer function which returns the amount of time in seconds which has elapsed since the timer was started.

⊡ **F**

Syntax
Tmr(*timerNumber*)

Parameters
timerNumber   Integer expression representing which of the 16 timers to check the time of.

Return Values
Elapsed time for the specified timer as a real number in seconds. Timer range is from 0 - approx. 1.7E+31. Timer resolution is 0.001 seconds.

Description
Returns elapsed time in seconds since the timer specified was started. 16 timers are available numbered 0 - 15.

Timers are reset with TmReset.

```
Real overhead

TmReset 0
overHead = Tmr(0)
```

See Also
TmReset

Tmr Function Example

```
TmReset 0          'Reset Timer 0
For i = 1 To 10      'Perform operation 10 times
    GoSub Cycle
Next
Print Tmr(0) / 10  'Calculate and display cycle time
```

TmReset Statement

Resets the timers used by the Tmr function.

S

Syntax
   TmReset *timerNumber*

Parameters
   timerNumber   Integer expression from 0 - 15 specifies which of the 16 timers to reset.

Description
   Resets and starts the timer specified by *timerNumber*. 16 timers are available, numbered from 0 - 15.

   Use the Tmr function to retrieve the elapsed time for a specific timer.

See Also
   Tmr

TmReset Example

```
TmReset 0          'Reset Timer 0
For i = 1 To 10     'Perform operation 10 times
    GoSub CYL
Next
Print Tmr(0)/10    'Calculate and display cycle time
```

Toff Statement

Turns off execution line display on the seven-segment.

$\boxed{\textbf{S}}$

Syntax
  Toff

Description
  Excution line will not be displayed on the seven-segment.

See Also
  Ton

Toff Example

```
Function main
   Ton MyTask
   ...
   Toff
Fend
```

Tool Statement

Selects or displays the current tool.

> S

Syntax
>   **(1) Tool**   *toolNumber*
>   **(2) Tool**

Parameters

>   *toolNumber*   Optional. Integer expression from 0-15 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.

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.

F

Syntax

> **Tool**

Return Values

Integer containing the current tool number.

Description

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also

Tool Statement

Tool Function Example

```
Integer savTool

savTool = Tool
Tool 2
Go P1
Tool savTool
```

Ton Statement

Specifies task Turns on debug trace.

S

Syntax
> **Ton** *taskIdentifier*
> **Ton**

Parameters
> *taskIdentifier*   Specifies the task name or 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
> Execution line of task 1 is displayed in initial status.
> **Ton** statement displays the specified task execution line on the seven-segment.
> When taskIdentifier is omitted, the task execution line with **Ton** statement execution is displayed on the seven-segment.

See Also
> Toff

Ton Example

```
Function main
  Ton MyTask
  ...
  Toff
Fend
```

Trap Statement

Defines interrupts and what should happen when they occur.

$\boxed{\textbf{S}}$

Syntax

> **Trap** *trapNumber*, *condition* **GoTo** { *label*}
> **Trap** *trapNumber*, *condition* **Call** *funcname*
> **Trap** *trapNumber*, *condition* **Xqt** *funcname*
> **Trap** *trapNumber*

Parameters

trapNumber Integer number from 1-4 representing which of 4 Trap numbers to use. (SPEL$^+$ supports up to 4 active Trap interrupts at the same time.)

condition The following functions and operators may be used in the *condition* expression:

> **Functions** : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemW, Ctr
>
> **Operators** : 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.

Note

The functionality of **Trap Call** in RC+ 4.x or before has been changed to **Trap Xqt** in RC+ 5.0.
The functionality of **Trap GoSub** in RC+ 4.x or before has been removed. Use **Trap Call** instead of **Trap GoSub**.

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 cancled.
Trap Xqt will be canceld when all tasks have stopped.

If GoTo is specified
The command being executed will be processed as described below, then control branches to the specified line number or label.
- Any arm motion will pause immediately
- Waiting status by the Wait or Input commands will discontinue
- All other commands will complete execution before control branches

If 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.

See Also
Call, Era, Erl, Err, Ert, ErrMsg$, GoSub, GoTo, OnErr, Xqt

Trap Example
<Example 1> Error process defined by User
Sw(0) Input is regarded as an error input defined by user.

```
Function Main
    Trap 1 Sw(0)= On GoTo EHandle  'Defines Trap
      .
      .
      .
EHandle:
    On 31    'Signal tower lights
    OpenCom #1
    Print #1, "Error is issued"
    CloseCom #1
Fend
```

<Example 2> Usage like multi-tasking

```
Function Main
    Trap 2 MemSw(0) = On Or MemSw(1) = On Call Feeder
      .
      .
      .
Fend
.
Function Feeder
  Select TRUE
    Case MemSw(0) = On
      MemOff 0
      On 2
    Case MemSw(1) = On
      MemOff 1
      On 3
  Send

  ' Re-arm the trap for next cycle
  Trap 2 MemSw(0) = On Or MemSw(1) = On Xqt Feeder
Fend
```

Trim$ Function

Returns a string equal to specified string without leading or trailing spaces.

F

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.

F

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, Wait

TW Function Example

```
Wait Sw(0) = On, 5     'Wait up to 5 seconds for input bit 0 On
If TW = True Then
    Print "Time Up"    'Display "Time UP" after 5 seconds
EndIf
```

UBound Function

Returns the largest available subscript for the indicated dimension of an array.

$\boxed{\text{F}}$

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
```

UCase$ Function

Returns a string that has been converted to uppercase.

$$\boxed{\textbf{F}}$$

Syntax
    UCase$ (*string*)

Parameters
    string   String expression.

Return Values
    The converted uppercase string.

See Also
    LCase$, LTrim$, Trim$, RTrim$

UCase$ Example

```
str$ = "Data"
str$ = UCase$(str$)  ' str$ = "DATA"
```

Val Function

Converts a character string that consists of numbers into their numerical value and returns that value.

| F |

Syntax
   Val*(string)*

Parameters
   string   String expression which contains only numeric characters. The string may also contain a prefix: &H (hexadecimal), &O (octal), or &B (binary).

Return Values
Returns an integer or floating point result depending upon the input string. If the input string has a decimal point character than the number is converted into a floating point number. Otherwise the return value is an integer.

Description
Val converts a character string of numbers into a numeric value. The result may be an integer or floating point number. If the string passed to the Val instruction contains a decimal point then the return value will be a floating point number. Otherwise it will be an integer.

See Also
Abs, Asc, Chr$, Int, Left$, Len, Mid$, Mod, Right$, Sgn, Space$, Str$

Val Example
The example shown below shows a program which coverts several different strings to numbers and then prints them to the screen.

```
Function ValDemo
    String realstr$, intstr$
    Real realsqr, realvar
    Integer intsqr, intvar

    realstr$ = "2.5"
    realvar = Val(realstr$)
    realsqr = realvar * realvar
    Print "The value of ", realstr$, " squared is: ", realsqr

    intstr$ = "25"
    intvar = Val(intstr$)
    intsqr = intvar * intvar
    Print "The value of ", intstr$, " squared is: ", intsqr
Fend
```

Here's another example from Command window.

```
> Print Val("25.999")
25.999
>
```

VxCalib Statement

Note: 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 (SCARA robot)
                5: Mobile on Joint #4 (SCARA robot)
                6: Mobile on Joint #5 (6-axis robot)
                7: Mobile on Joint #6 (6-axis robot)

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 robot coordinates must be set as TOOL: 0, ARM: 0.

TwoRefPoints Available for syntax (2).
        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 comand 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)
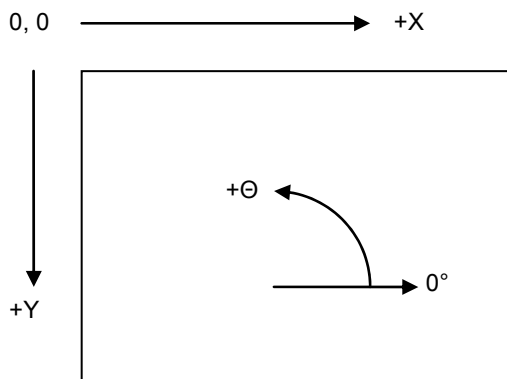
For the pixel coordinates and robot coordinates, set the top left position of the window as Point 1 and set the bottom right position as Point 9 according to the order in the table below.
It is classified into the four categories by the parameter CamOrient and TwoRefPoints.

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 coordinates10	180 degrees
11	Top center	- - -	Measuring point coordinates11	
12	Top right	- - -	Measuring point coordinates12	
13	Center right	- - -	Measuring point coordinates13	
14	Center	- - -	Measuring point coordinates14	
15	Center left	- - -	Measuring point coordinates15	
16	Bottom left	- - -	Measuring point coordinates16	
17	Bottom center	- - -	Measuring point coordinates17	
18	Bottom right	- - -	Measuring point coordinates18	

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

Notes

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

Note: 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

Note: 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

Note: 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 [deg]
6	Y Avg error [mm]
7	Y Max error [mm]
8	Y mm per pixel [mm]
9	Y tilt [deg]

Return Value

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

Note: 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

Note: 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 BASE: 0, TOOL: 0, ARM: 0.

Return Value

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 BASE: 0, TOOL: 0, 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 Statement 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.)

> | S

Syntax
  (1) **Wait** *time*
  (2) **Wait** *inputCondition*
  (3) **Wait** *inputCondition*, *time*

Parameters

time Real expression between 0 and 2,147,483 which represents the amount of time to wait when using the Wait instruction to wait based on time. Time is specified in seconds. The smallest increment is .01 seconds.

inputCondition The following functions and operators may be used in the inputCondition:

  Functions : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemW, Ctr, Motor, LOF, ErrorOn, SaftyOn, EstopOn, InsideBox, InsidePlane, AtHome

  Operators : 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 Input Conditions without Time Interval**
    When used as a conditional **Wait** interlock, the **Wait** instruction causes the program to wait until specified conditions are satisfied. If after TMOut time interval has elapsed and the **Wait** conditions have not yet been satisfied, an error occurs. The user can check multiple conditions with a single Wait instruction by using the And, Mask, Or, or Xor instructions. (Please review the example section for **Wait**.)

  (3) **Wait with Input Condition and Time Interval**
    Specifies **Wait** condition and time interval. After either **Wait** condition is satisfied, or the time interval has elapsed, program control transfers to the next command. Use Tw to verify if the Wait condition was satisfied or if the time interval elapsed.

Notes

Specifying a Timeout for Use with Wait

When the **Wait** instruction is used without a time interval, a timeout can be specified which sets a time limit to wait for the specified condition. This timeout is set through using the TMOut instruction. Please refer to this instruction for more information. (The default setting for TMOut is 0 which means no timeout.)

Wait the change of a variable using Wait

The following program does not wait for the global variable i changing to 3.
  Global Integer i
  Wait i=3
When variables are included in the Waitcondition, their values are computed when setting the Waitcondition. No use of variable is recommended. Otherwise, the condition may be an unintended condition.

Using Wait with Lof function

When Wait is used with Lof function, the port number for Lof function is 1 to 8 and 201 to 216.
Standard COM port (1001, 1002) is not available. If specified these ports, error occurs during the operation.

See Also

AtHome, Ctr, ErrorOn, EstopOn, In, InsideBox, InsidePlane, InW, LOF, Mask, MemIn, MemSw, MemW, Motor, Oport, Out, OutW, SaftyOn, Sw, TMOut, Tw

Wait Example

The example shown below shows 2 tasks each with the ability to initiate motion instructions. However, a locking mechanism is used between the 2 tasks to ensure that each task gains control of the robot motion instructions only after the other task is finished using them. This allows 2 tasks to each execute motion statements as required and in an orderly predictable fashion. MemSw is used in combination with the Wait instruction to wait until the memory I/O #1 is the proper value before it is safe to move again.

```
Function main
    Integer I
    MemOff 1
    Xqt !2, task2
    For i = 1 to 100
       Wait MemSw(1) = Off
       Go P(i)
       MemOn 1
    Next I
Fend

Function task2
    Integer i
    For i = 101 to 200
       Wait MemSw(1) = On
       Go P(i)
       MemOff 1
    Next i
Fend


' Wait until input 0 turns on
Wait Sw(0) = On

' Wait 60.5 secs and then continue execution
Wait 60.5

' Wait until input 0 is off and input 1 is on
Wait Sw(0) = Off And Sw(1) = On

' Wait until memory bit 0 is on or memory bit 1 is on
Wait MemSw(0) = On Or MemSw(1) = On

'Wait one second, then turn output 1 on
Wait 1; On 1

' Wait for the lower 3 bits of input port 0 to equal 1
Wait In(0) Mask 7 = 1
```

WaitNet Statement

Wait for TCP/IP port connection to be established.

S

Syntax
> **WaitNet** #*portNumber* [, *timeOut*]

Parameters

portNumber	Integer expression for port number to connect. Range is 201 - 208
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.

$\boxed{\textbf{S}}$

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  ' wait for robot to decelerate
On 1
CP Off
```

WaitSig Statement

Waits for a signal from another task.

$\boxed{\textbf{S}}$

Syntax
>   **WaitSig** *signalNumber* [, *timeOut*]

Parameters
>   *signalNumber*   Integer expression representing signal number to receive. Range is from 0 to 15.
>   *timeOut*   Optional. Real expression representing the maximum time to wait.

Description
>   Use **WaitSig** to wait for a signal from another task. The signal will only be received after **WaitSig** has started. Previous signals are ignored.

See Also
>   Wait, WaitPos, Signal

WaitSig Example

```
Function Main
  Xqt SubTask
  Wait 1
  Signal 1
    .
    .
Fend

Function SubTask
  WaitSig 1
  Print "signal received"
    .
Fend
```

Weight Statement

Specifies or displays the inertia of the robot arm.

> S

Syntax
Weight *payloadWeight* [*, distance*]
Weight

Parameters
payloadWeight   The weight of the end effector to be carried in Kg unit.

distance   The distance from the rotational center of the second arm to the center of the gravity of the end effector in mm unit. Valid only for SCARA robots (including RS series).

Return Values
Displays the current **Weight** settings when parameters are omitted.

Description
Specifies parameters for calculating Point to Point motion maximum acceleration. The **Weight** instruction specifies the weight of the end effector and the parts to be carried.

The Arm length (*distance*) specification is necessary only for SCARA 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 equivalent value work piece weight calculated from specified parameters exceeds the maximum allowable payload, an error occurs.

Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

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.

See Also
Accel, Inertia

Weight Statement Example
This Weight instruction on the Command window displays the current setting.

```
> weight
2.000, 200.000
>
```

Weight Function

Returns a Weight parameter.

F

Syntax
    Weight(*paramNumber*)

Parameters
    paramNumber   Integer expression containing one of the values below:

                                   1: Payload weight

                                   2: Arm length

Return Values
    Real number containing the parameter value.

See Also
    Inertia, Weight Statement

Weight Function Example

```
Print "The current Weight parameters are: ", Weight(1)
```

Where Statement

Displays current robot position data.

> |

Syntax
Where [*localNumber*]

Parameters
localNumber　　　Optional. Specifies the local coordinate system number. Local 0 is default.

See Also
Joint, PList, Pulse

Where Statement Example

```
>where
WORLD:  X:  350.000 mm  Y:    0.000 mm  Z:    0.000 mm  U:    0.000 deg
JOINT:  1:    0.000 deg 2:    0.000 deg 3:    0.000 mm  4:    0.000 deg
PULSE:  1:        0 pls 2:        0 pls 3:        0 pls 4:        0 pls

> local 1, 100,100,0,0

> where 1
WORLD1: X:  250.000 mm  Y: -100.000 mm  Z:    0.000 mm  U:    0.000 deg
JOINT:  1:    0.000 deg 2:    0.000 deg 3:    0.000 mm  4:    0.000 deg
PULSE:  1:        0 pls 2:        0 pls 3:        0 pls 4:        0 pls
```

Wrist Statement

Sets the wrist orientation of a point.

> S

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 ommited, the wrist orientation for the specified point is displayed.

See Also
    Elbow, Hand, J4Flag, J6Flag, Wrist Function

Wrist Statement Example

```
Wrist P0, Flip
Wrist P(mypoint), NoFlip

P1 = 320.000, 400.000, 350.000, 140.000, 0.000, 150.000
```
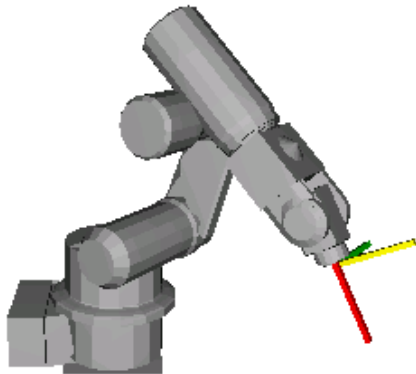


```
Wrist P1, NoFlip

Go P1
```
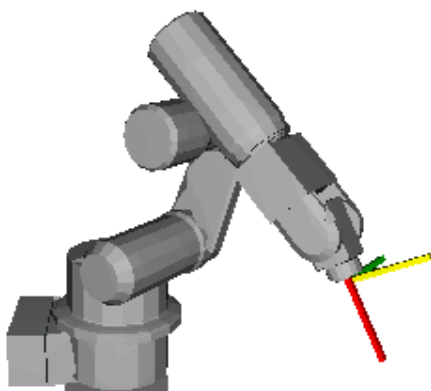


```
Wrist P1, Flip
Go P1
```

Wrist Function

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 communication port without end of line terminator.

$\boxed{\textbf{S}}$

Syntax

    Write #*portNumber*, *string*

Parameters

portNumber	Communications port number to write to.
string	String expression that will be written to the file.

Description

    Write is different from Print in that it does not add an end of line terminator.

See Also

    Print, Read

Write Example

```
OpenCom #1
For i = 1 to 10
    Write #1, data$(i)
Next i
CloseCom #1
```

WriteBin Statement

Writes binary data to a communications port.

$\boxed{\textbf{S}}$

Syntax
> **WriteBin** #*portNumber*, *data*
> **WriteBin** #*portNumber*, *array(), count*

Parameters

portNumber	Communications port to read from.
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 and must be less than or equal to the number of array elements.

Parameters

portNumber	Communications port to read from.
data	Integer expression containing the data to be written.

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.

Description

The **Xor** operator performs the bitwise **Xor** operation on the values of the operands. Each bit of the result is the **Xor**ed value of the corresponding bits of the two operands.

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

See Also

And, LShift, Not, Or, Rshift

Xor Operator Example

```
>print 2 Xor 6
 4
>
```

Xqt Statement

Initiates execution of a task from within another task.

$\boxed{\textbf{S}}$

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 16.
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.
Normal	Executes a normal task.
NoPause	Executes a task that does not pause at Pause statement or Pause input signal occurance or Safety Door Open.
NoEmgAbort	Executes a task that continue processing at Emergency Stop or error occurence.

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

Speciify *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+ 5.0 User's Guide*.

Unavailable Commands in NoEmgAbort Task

The following commands cannot be executed in NoEmgAbort task.

A	Accel	E	ECP	M	Motor	T	TGo
	AccelR		ECPClr		Move		Till
	AccelS		ECPSet	O	OLAccel		TLSet
	Arc	F	Fine	P	Pass		TLClr
	Arc3		Find		Plane		TMove
	Arch	G	Go		PlaneClr		Tool
	Arm	H	Home		Power		Trap
	ArmSet		HomeClr		PTPBoost	V	VGet
	ArmClr		HomeSet		Pulse		VRun
B	Base		Hordr	Q	QP		VSet
	BGo	I	Inertia	R	Range	W	WaitPos
	BMove	J	JTran		Reset		Weight
	Box		Jump	S	Sense	X	XYLim
	BoxClr		Jump3		SFree		Xqt
	Brake		Jump3CP		SLock		
C	CP		JRange		Speed		
	Curve	L	LimZ		SpeedR		
	CVMove		Local		SpeedS		
			LocalClr				

See Also

Function...Fend, Halt, Resume, Quit

Xqt 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.

F

Syntax
> **XY(***x, y, z, u,* [*v, w*]**)**

Parameters
> | | |
> |---|---|
> | *x* | Real expression representing the X coordinate. |
> | *y* | Real expression representing the Y coordinate. |
> | *z* | Real expression representing the Z coordinate. |
> | *u* | Real expression representing the U coordinate. |
> | *v* | Optional for 6-Axis robots. Real expression representing the V coordinate. |
> | w | Optional for 6-Axis robots. Real expression representing the W coordinate. |

Return Values
> A point constructed from the specified coordinates.

See Also
> JA, Point Expression

XY Function Example

```
P10 = XY(60, 30, -50, 45) + P20
```

XYLim Statement

Sets or displays the permissible XY motion range limits for the robot.

> S

Syntax

XYLim *minX, maxX, minY, maxY,* [*minZ*], [*maxZ*]
XYLim

Parameters

minX	The minimum X coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the X Coordinate less than *minX*.)
maxX	The maximum X coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the X Coordinate greater than *maxX*.)
minY	The minimum Y coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Y Coordinate less than *minY*.)
maxY	The maximum Y coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Y Coordinate greater than *maxY*.)
minZ	Optional. The minimum Z coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Z Coordinate less than *minZ*.)
maxZ	Optional. The maximum Z coordinate position to which the manipulator may travel. (The manipulator may not move to a position with the Z Coordinate greater than *maxZ*.)

Return Values

Displays current **XYLim** values when used without parameters

Description

XYLim is used to define XY motion range limits. Many robot systems allow users to define joint limits but the SPEL$^+$ language allows both joint limits and motion limits to be defined. In effect this allows users to create a work envelope for their application. (Keep in mind that joint range limits are also definable with SPEL.)

The motion range established with **XYLim** values applies to motion command target positions only, and not to motion paths from starting position to target position. Therefore, the arm may move outside the XYLim range during motion. (i.e. The **XYLim** range does not affect Pulse.)

Notes

Turning Off Motion Range Checking

There are many applications which don't require Motion Range limit checking and for that reason there is a simple method to turn this limit checking off.   To turn motion range limit checking off, define the Motion Range Limit values for *minX*, *maxX*, *minY*, and *maxY* to be 0. For example XYLim 0, 0, 0, 0.

Default Motion Range Limit Values

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+ 5.0 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

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.

$\boxed{\textbf{F}}$

Syntax
>   **XYLim**(*limit*)

Parameters
>   *limit*   Integer expression that specifies which limit to return.
>
>   1: Lower limit.
>
>   2: Upper limit.

Return Values
>   Point containing the specified limit coordinates.

Description
>   Point data is stored to the Compact Flash inside the Controller. When you execute this command, 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 this command only when saving the point data is recommended.

See Also
>   XYLim Statement

XYLim Function Example

```
P1 = XYLim(1)
P2 = XYLim(2)
```

XYLimClr Statement

Clears the XYLim definition.

`>` `S`

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.

F

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
```

SPEL⁺ Error Messages

To get help for any SPEL⁺ error, place the cursor on the error message in the run or command windows and press the F1 key.

There are 18 types of errors as follows.

Events	Simulator	Points
Warnings	Interpreter	Fieldbus
Controller Main	Parser	Vision
Operator Panel	Motor control	GUI Builder
Teach Pendant	Servo	Hardware
PC	Vision Calibration	EPSON RC+

Events

No.	Message	Remedy	Note 1	Note 2
1	Controller control program started.			
2	Termination due to low voltage of the power supply.			
3	Controller control program has completed.	Stores this log when the controller is rebooted from EPSON RC+ or TP1.		
4	Preserve variables save area has been cleaned.			
5	Function Main started.			
6	Function Main started. Later same logs are skiped.	Skip the log "Function Main started." to prevent system history space run out.		
7	Serial number has been saved.			
8	System backup has been executed.			
9	System restore has been executed.			
10	Robot parameters have been initialized.			
11	Offset pulse value between the encoder origin and the home sensor (HOFS) is changed. Additional data is J1 value.		J1 value after change	J1 value before change
12	Offset pulse value between the encoder origin and the home sensor (HOFS) is changed. Additional data is J2 value.		J2 value after change	J2 value before change
13	Offset pulse value between the encoder origin and the home sensor (HOFS) is changed. Additional data is J3 value.		J3 value after change	J3 value before change
14	Offset pulse value between the encoder origin and the home sensor (HOFS) is changed. Additional data is J4 value.		J4 value after change	J4 value before change
15	Offset pulse value between the encoder origin and the home sensor (HOFS) is changed. Additional data is J5 value.		J5 value after change	J5 value before change
16	Offset pulse value between the encoder origin and the home sensor (HOFS) is changed. Additional data is J6 value.		J6 value after change	J6 value before change
17	Move to the message saving mode.			
18	Conversion of Robot Parameter file has been executed.			
100	Device connected to Controller.			

No.	Message	Remedy	Note 1	Note 2
101	Console device has changed.		21:PC 22:Remote 23:OP1	
102	Display device has changed.			
103	Working mode has changed.			
110	Controller firmware has been installed.		1:Setup 2:Initialize 3:Upgrade 4:Recover	
111	IP address has been restored.	May store this log when the controller firmware is installed.		
120	PC connected to the Controller.		1:Ethernet 2:USB	
121	TP connected to the Controller.			
122	OP connected to the Controller.			
123	PC disconnected from the Controller.			
124	TP disconnected from the Controller.			
125	OP disconnected from the Controller.			
126	Working mode changed to AUTO.			
127	Working mode changed to Program.			
128	Working mode changed to Teach.			
129	Remote Ethernet connected to the Controller.			
130	Remote Ethernet disconnected from the Controller.			
131	Remote RS232 connected to the Controller.			
132	Remote RS232 disconnected from the Controller.		LogoutStatus 0:Nomal 1:Abnormal (Timeout)	

Warnings

No.	Message	Remedy	Note 1	Note 2
501	Trace history is active.	Effects system performance if trace history is active.		
502	Memory has been initialized.	When this error occurs, the value of the Global Preserve variable will be initialized. Replace the CPU board battery. Replace the CPU board.		
511	Battery voltage of the CPU board backup is lower than the allowed voltage. Replace the CPU board battery.	Replace the CPU board battery immediately. Keep the power to the controller ON as far as possible until you replace the battery.	Current value	Boundary value
512	5V input voltage for the CPU board is lower than the allowed voltage.	If normal voltage is not generated by a 5V power supply alone, replace the power supply.	Current value	Boundary value
513	24 V input voltage for the motor brake, encoder and fan is lower than the specified voltage.	If normal voltage is not generated by a 24V power supply alone, replace the power supply.	Current value	Boundary value
514	Internal temperature of the Controller is higher than the allowed temperature.	Stop the controller as soon as possible and check whether the ambient temperature of the controller is not high. Check whether the filter is not clogged up.	Current value	Boundary value
515	Rotating speed of the controller fan is below the allowed speed. (FAN1)	Check whether the filter is not clogged up. If the warning is not cleared after the controller is rebooted, replace the fan.	Current value	Boundary value
516	Rotating speed of the controller fan is below the allowed speed. (FAN2)	Check whether the filter is not clogged up. If the warning is not cleared after the controller is rebooted, replace the fan.	Current value	Boundary value
517	Internal temperature of the Controller is higher than the allowed temperature.			
700	Motor driver type does not match the current robot model. Check the robot model. Replace the motor driver.	Check the robot model.		
736	Encoder has been reset. Reboot the controller.	Reboot the controller.		
737	Low voltage from the encoder battery. Replace the battery with the controller ON.	Replace the battery for the robot with the controller ON.		
752	Servo alarm D.			

Controller Main

No.	Message	Remedy	Note 1	Note 2
1001	Operation Failure. Command parameter is invalid.			
1002	Requested data cannot be accessed. The data is not set up or the range is invalid.	Check whether the target I/O, variables, and tasks exist.		
1003	The password is invalid	Enter the correct password.		
1004	Cannot execute with unsupported version.	Use the correct version file.		
1005	Cannot execute with invalid serial number.	Use the backup data for the same controller to restore the controller configuration.		
1006	Cannot execute with invalid Robot model.	Use the backup data for the same controller to restore the controller configuration.		
1007	Cannot execute with invalid Controller.	Controller connected with PC is not supported. Connect with a regular controller.		
1008	Initialization failure. Failed to initialize TP.			
1009	OP is not supported by the connected controller.			
1020	Cannot execute in recovery mode.	Boot the controller as normal.		
1021	Cannot execute due to controller initialization failure.	Restore the controller configuration.		
1022	Cannot execute without the project being open.	Open a project.		
1023	Cannot execute while the project is open.	Rebuild the project.		
1024	Cannot activate from remote.	Enable the remote input.		
1025	Execution in Teach mode is prohibited.	Change to the AUTO mode.		
1026	Cannot execute in Teach mode except from TP.	Change to the AUTO mode.		
1027	Cannot execute in Auto mode.	Change to the Program mode.		
1028	Cannot execute in Auto mode except from the main console.	Change to the Program mode.		
1029	Cannot execute from OP.	Enable the OP input.		
1030	Does not allow Operation mode to be changed.	Change to the Auto mode with a console in the Program mode.		
1031	Cannot execute while tasks are executing.	Stop the task and then execute.		
1032	Cannot execute while the maximum number of tasks are executing.	Stop the task and then execute.		
1033	Cannot execute during asynchronous motion command.	Execute after the motion ends.		
1034	Asynchronous command stopped during operation.	The asynchronous command already stopped when the controller received a stop command.		
1035	Cannot execute in Remote enable except from the Remote.			
1036	Cannot execute in OP enable except from the OP.			

No.	Message	Remedy	Note 1	Note 2
1037	Cannot execute in Remote Ethernet enable except from the Remote Ethernet.			
1040	Cannot execute in Remote RS232C enable except from the Remote RS232C.			
1041	Cannot execute during Emergency Stop status.	Cancel the Emergency Stop status.		
1042	Cannot execute while the safeguard is open.	Close the safeguard.		
1043	Cannot execute during error condition.	Cancel the error condition.		
1044	Cannot execute when the remote pause input is ON.	Change the remote pause input to OFF.		
1045	Input waiting condition is the only available condition to input.	The controller received an input while it was not in the Input waiting condition.		
1046	Cannot execute during file transfer.	Execute after the file transmission.		
1047	Cannot cancel the command executed from other devices.	Cancel the motion command from the device the command was issued from.		
1048	Cannot execute after after low voltage was detected.			
1049	Other devices are in program mode.			
1050	Password is too long.			
1051	Export Controller Status failed.			
1052	Export Controller Status busy.			
1100	File failure. Cannot access the file.			
1102	File failure. Read and write failure of the registry			
1103	File is not found.	Check whether the file exists.		
1104	Project file was not found.	Rebuild the project.		
1105	Object file was not found.	Rebuild the project.		
1106	Point files were not found.	Rebuild the project.		
1107	The program is using a feature that is not supported by the current controller firmware version.			
1108	One or more source files are updated. Please build the project.	Rebuild the project.		
1109	Not enough storage capacity.	Increase free space of the USB memory.		
1110	File is not found.			
1120	File failure. Setting file is corrupt.	Restore the controller configuration.		
1121	File failure. Project file is corrupt.	Rebuild the project.		
1122	File failure. Point file is corrupt.	Rebuild the project.		
1123	File failure. I/O label file is corrupt.	Rebuild the project.		
1124	File failure. User error file is corrupt.	Rebuild the project.		
1125	File failure. Error message file is corrupt.			
1126	File failure. Software option infomation is corrupt.			
1127	File failure. Vision file is corrupt.	Rebuild the project.		

No.	Message	Remedy	Note 1	Note 2
1128	File failure. Backup information file is corrupt.			
1130	Error message failure. No item is found in the error history.			
1131	Cannot access the USB memory.	Insert the USB memory properly. When this error still occurs after the USB memory is inserted properly, the memory may be unrecognizable to controller. Insert another memory to check the operation.		
1132	File failure. Failed to copy the file.			
1133	File failure. Failed to delete the file.			
1135	File failure. The name of Playback is invalid.			
1140	File failure. Failed to open the object file.			
1141	File failure. Failed to open the project file.			
1142	File failure. Failed to read the project file.			
1143	File failure. Failed to open the condition save file.			
1144	File failure. Failed to write the condition save file.			
1150	File failure. Error history is invalid.			
1151	File failure. Failed to map the error history.			
1152	File failure. Failed to open the error history file.			
1153	File failure. Failed to write the error history file.			
1155	File failure. Failed to open the settings file.	Restore the controller configuration.		
1156	File failure. Failed to save the settings file.	Restore the controller configuration.		
1157	File failure. Failed to read the settings file.	Restore the controller configuration.		
1158	File failure. Failed to write the settings file.	Restore the controller configuration.		
1160	MCD failure. Failed to open the MCD file.	Restore the controller configuration.		
1161	MCD failure. Failed to read the MCD file.	Restore the controller configuration.		
1162	MCD failure. Failed to write the MCD file.	Restore the controller configuration.		
1163	MCD failure. Failed to save the MCD file.	Restore the controller configuration.		
1165	MPD failure. Failed to open the MPD file.			
1166	MPD failure. Failed to read the MPD file.			
1167	MPD failure. Failed to write the MPD file.			
1168	MPD failure. Failed to save the MPD file.			

No.	Message	Remedy	Note 1	Note 2
1170	MPL failure. Failed to open the MPL file.			
1171	MPL failure. Failed to read the MPL file.			
1172	MPL failure. Failed to write the MPL file.			
1173	MPL failure. Failed to save the MPL file.			
1175	MAL failure. Failed to open the MAL file.			
1176	MAL failure. Failed to read the MAL file.			
1177	MAL failure. Failed to write the MAL file.			
1178	MAL failure. Failed to save the MAL file.			
1180	MTR failure. Failed to create the MTR file.			
1181	PRM failure. Failed to replace the PRM file.			
1185	File failure. Failed to open the backup information file.			
1186	File failure. Failed to read the backup information file.			
1187	File failure. Failed to write the backup information file.			
1188	File failure. Failed to save the backup information file.			
1189	The backup data was created by an old version.	Cannot restore the controller configuration in the specified procedure for using old backup data. Check the backup data.		
1190	The backup data was created by a newer version.			
1191	There is no project in the backup data.			
1200	Compile failure. Check the compile message.	This error occurs during compilation from TP. Correct where the error occurred.		
1201	Link failure. Check the link message.	This error occurs during compilation from TP. Correct where the error occurred.		
1500	Communication error.			
1501	Command did not complete in time.	Execute the command again after a while. Check the connection between the PC and controller.		
1502	Communication disconnection between PC and Controller. Re-establish communication.	Check the connection between the PC and controller.		
1503	Disconnection while executing a task.			
1510	Out of IP Address range.			
1521	Vision communication. Failed to initialize Ethernet.			

No.	Message	Remedy	Note 1	Note 2
1522	Vision communication. Failed to terminate Ethernet.			
1523	Vision communication. Failed to create the socket handle.			
1524	Vision communication. Failed to connect.			
1526	Vision communication. Failed to send to the server.	Check the connection of the camera and controller.		
1527	Vision communication. Failed to read from the server.	Check the connection of the camera and controller.		
1528	Vision communication. Failed to set option.			
1529	Vision communication. Ethernet has not been initialized yet.			
1530	Vision communication. Connection is not completed.	Check the connection of the camera and controller.		
1531	Vision communication. All sockets are used.			
1532	Vision communication. Send timeout.	Check the connection of the camera and controller.		
1533	Vision communication. Read timeout.	Check the connection of the camera and controller.		
1534	Vision communication. Communication error.	Check the connection of the camera and controller.		
1550	Communication failure. Ethernet initialization error.			
1551	Communication failure. USB initialization error.			
1552	Communication failure. Controller internal communication error.			
1553	Communication failure. Invalid data is detected.			
1555	Ethernet transmission error.	Check the connection between the PC and controller.		
1556	Ethernet reception error.	Check the connection between the PC and controller.		
1557	USB transmission error.	Check the connection between the PC and controller.		
1558	USB reception error.	Check the connection between the PC and controller.		
1559	Communication failure. Failed to allocate memory.			
1580	Parser communication error.			
1581	Parser communication failure. Timeout error occurred during communication with parser.			
1582	Parser communication failure. Parser transmission error.			
1583	Parser communication failure. Parser initialization error.			
1584	Parser communication failure. Connection error.			
1585	Parser communication failure. Parameter is invalid.			

No.	Message	Remedy	Note 1	Note 2
1586	Parser communication failure. Busy.			
1587	Parser communication failure. Invalid data is detected.			
1901	Unsupported. Unsupported command was attempted.			
1902	Unsupported. Unsupported parameter was specified.			
1903	System error.			

Operator Panel

No.	Message	Remedy	Note 1	Note 2
1600	Initialization failure. Failed to initialize OP.			
1603	Timeout error occurred during communication with OP.	Check whether the cable is firmly connected. Replace the cable.		
1604	Parity error occurred during communication with OP.	Check whether the cable is firmly connected. Replace the cable.		
1605	Framing error occurred during communication with OP.	Check whether the cable is firmly connected. Replace the cable.		
1606	Overrun error occurred during communication with OP.	Check whether the cable is firmly connected. Replace the cable.		
1607	Checksum error occurred during communication with OP.	Check whether the cable is firmly connected. Replace the cable.		
1608	Retry error occurred during communication with OP.	Check whether the cable is firmly connected. Replace the cable.		
1609	OP cannot be connected.	Upgrade the controller software. Upgrade the OP firmware.		

Teach Pendant

No.	Message	Remedy	Note 1	Note 2
1700	Initialization failure. Failed to initialize TP.			
1701	Initialization failure. Failed to initialize TP.			
1702	Initialization failure. Failed to initialize TP.			
1703	File failure. Failed to read the screen data file.			
1704	Failed to read the setting file.			
1706	Failed to open the TP port.			
1708	Failed to read the key table for TP.			
1709	Failed to change the language.			
1710	Failed to make the screen.			

PC

No.	Message	Remedy	Note 1	Note 2
1800	The controller is already connected to a PC.	Only one PC can be connected to the controller.		
1802	The command was attempted without being connected to a controller.			
1803	Failed to read or write the file on the PC.			
1804	Initialization failure. Failed to allocate memory on the PC.			
1805	Connection failure. Check the controller startup and connection of the communication cable.			
1806	Timeout during connection via Ethernet.			
1807	Timeout during connection via USB.			
1808	USB driver is not installed.	Failed to install EPSON RC+ 5.0. Install EPSON RC+ 5.0 again.		

Simulator

No.	Message	Remedy	Note 1	Note 2
1861	Initialization failure. Failed to initialize SimulatorMNG.			
1862	Initialization failure. Failed to initialize WBProxy.			
1863	The parameter is invalid.			
1864	Initialization failure. Virtual controller does not exist.	Installation of EPSON RC+ 5.0 was failed. Reinstall EPSON RC+ 5.0.		
1865	Initialization failure. Failed to start virtual controller.	Try again after a while.		
1866	Termination failure. Failed to terminate virtual controller.			
1867	Cannot execute because it is not dry run mode.			
1868	Initialization failure. Directory cannot be found.			
1869	Initialization failure. File cannot be found.			
1870	Pallet failure. Number of point is beyond the maximum value.			
1871	Connection failure. Virtual controller version is old.			
1872	Connection failure. Files for simulator that used real controller cannot be found.			
1873	Connection failure. Files for simulator that used virtual controller cannot be found.	Register the virtual controller again in the connection setting.		
1874	Virtual Controller cannot be added.	Installation of EPSON RC+ 5.0 was failed. Reinstall EPSON RC+ 5.0.		
1875	Simulator Object failure. Cannot register data of the simulator object.			
1876	Simulator Object failure. Cannot register data of the simulator object.			
1877	Simulator Object failure. Cannot remove data of the simulator object.			
1878	Simulator Object failure. Cannot update data of the simulator object.			
1879	Other virtual controllers are starting.	Start another EPSON RC+5.0 and check if it connects with the virtual controller.		
1880	Cannot execute during controller reset.			

No.	Message	Remedy	Note 1	Note 2
7750	Initialization failure.	Reboot RC+.		
7751	Failed to save the objects.	Reboot RC+.		
7752	Failed to load the objects.	Reboot RC+.		
7753	Failed to mapping of memory.	Reboot RC+.		
7754	The virtual controller already exists.	Name of the virtual controller may be duplicated. Check the virtual controller name.		
7755	Failed to create the virtual controller connection information.	Reboot RC+.		
7756	The copy source of the virtual controller does not exist.	Check the virtual controller name.		
7757	The copy destination of the virtual controller already exists.	Name of the virtual controller may be duplicated. Check the virtual controller name.		
7758	Failed to copy the virtual controller connection information.	Reboot RC+.		
7759	Failed to delete the virtual controller connection information.	Reboot RC+.		
7760	Failed to delete the controller connection information.	Reboot RC+.		
7761	Failed to rename the controller connection information.	Check the virtual controller name.		
7762	The rename source of the virtual controller does not exist.	Check the virtual controller name.		
7763	The rename destination of the virtual controller already exists.	Check the virtual controller name.		
7764	Invalid Robot number.	Reboot RC+.		
7765	Failed to read the Robot definition file.	Check whether the definition file exists.		
7766	Failed to copy the layout objects.	Reboot RC+		
7767	Failed to cut the layout objects.	Reboot RC+		
7768	Failed to paste the layout objects.	Reboot RC+		
7769	Failed to remove the Robot.	Reboot RC+		
7770	Cannot execute with unsupported version.	Update RC+ to the latest version		

Interpreter

No.	Message	Remedy	Note 1	Note 2
2000	Unsupported. Unsupported command was attempted.	Rebuild the project.		
2001	Unsupported. Unsupported motion command was attempted.	Rebuild the project.		
2002	Unsupported. Unsupported conveyer command was attempted.	Rebuild the project.		
2003	Unsupported. Unsupported Function argument was specified.	Rebuild the project.		
2004	Unsupported. Unsupported Function return value was specified.	Rebuild the project.		
2005	Unsupported. Unsupported condition was specified.	Rebuild the project.		
2006	Unsupported. Unsupported I/O command was specified.	Rebuild the project.		
2007	Unsupported condition was specified.			
2008	Unsupported. Unknown error number.			
2009	Unsupported. Invalid Task number.			
2010	Object file error. Build the project. Out of internal code range.	Rebuild the project.		
2011	Object file error. Build the project. Function argument error.	Rebuild the project.		
2012	Object file error. Build the project. Command argument error.	Rebuild the project.		
2013	Object file error. Build the project. Cannot process the code.	Rebuild the project.		
2014	Object file error. Build the project. Cannot process the variable type code.	Rebuild the project.		
2015	Object file error. Build the project. Cannot process the string type code.	Rebuild the project.		
2016	Object file error. Build the project. Cannot process the variable category code.	Rebuild the project.		
2017	Object file error. Build the project. Cannot process because of improper code.	Rebuild the project.		
2018	Object file error. Build the project. Failed to calculate the variable size.	Rebuild the project.		
2019	Object file error. Cannot process the variable wait. Build the project.	Rebuild the project.		
2020	Stack table number exceeded. Function call or local variable is out of range.	Check whether no function is called infinitely. Reduce the Call function depth.		

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4

No.	Message	Remedy	Note 1	Note 2
2021	Stack area size exceeded. Stack error. Function call or local variable is out of range.	If using many local variables, especially String type, replace them to global variables.		
2022	Stack failure. Required data not found on the stack.	Rebuild the project.		
2023	Stack failure. Unexpected tag found on the stack.	Rebuild the project.		
2030	System failure. Drive unit quantity is beyond the maximum count.	Restore the controller configuration.		
2031	System failure. Robot number is beyond the maximum count.	Restore the controller configuration.		
2032	System failure. Task number compliance error.	Rebuild the project.		
2033	System failure. Too many errors.	Remedy the errors occurring frequently.		
2040	Thread failure. Failed to create the thread.			
2041	Thread failure. Thread creation timeout.			
2042	Thread failure. Thread termination timeout.			
2043	Thread failure. Thread termination timeout.			
2044	Thread failure. Daemon process timeout.			
2045	Thread failure. Task continuance wait timeout.			
2046	Thread failure. Task stop wait timeout.			
2047	Thread failure. Task startup wait timeout.			
2050	Object file operation failure. Object file size is beyond the allowable size.	Rebuild the project.		
2051	Object file operation failure. Cannot delete the object file during execution.	Reboot the controller.		
2052	Object file operation failure. Cannot allocate the memory for the object file.	Reboot the controller.		
2053	Object file update. Updating the object file.	Perform the same processing after a while. Rebuild the project.		
2054	Object file operation failure. Synchronize the project. Function ID failure.	Synchronize the files of the project. Rebuild the project.		
2055	Object file operation failure. Synchronize the project. Local variable ID failure.	Synchronize the files of the project. Rebuild the project.		
2056	Object file operation failure. Synchronize the project. Global variable ID failure.	Synchronize the files of the project. Rebuild the project.		
2057	Object file operation failure. Synchronize the project. Global Preserve variable ID failure.	Synchronize the files of the project. Rebuild the project.		

No.	Message	Remedy	Note 1	Note 2
2058	Object file operation failure. Failed to calculate the variable size.	Synchronize the files of the project. Rebuild the project.		
2059	Exceed the global variable area. Cannot assign the Global variable area.	Reduce the number of Global variables to be used.		
2070	SRAM failure. SRAM is not mapped.	Replace the CPU board.		
2071	SRAM failure. Cannot delete when Global Preserve variable is in use.	Perform the same processing after a while. Rebuild the project.		
2072	Exceed the backup variable area. Cannot assign the Global Preserve variable area.	Reduce the number of Global Preserve variables to be used.	Maximum size	The size you attempted to use
2073	SRAM failure. Failed to clear the Global Preserve variable area.	Rebuild the project.		
2074	SRAM failure. Failed to clean up the Global Preserve variable save area.	Reboot the controller.		
2100	Initialization failure. Failed to open the initialization file.	Restore the controller configuration.		
2101	Initialization failure. Duplicated initialization.			
2102	Initialization failure. Failed to initialize MNG.			
2103	Initialization failure. Failed to create an event.			
2104	Initialization failure. Failed to setup a priority.			
2105	Initialization failure. Failed to setup the stack size.			
2106	Initialization failure. Failed to setup an interrupt process.			
2107	Initialization failure. Failed to start an interrupt process.			
2108	Initialization failure. Failed to stop an interrupt process.			
2109	Initialization failure. Failed to terminate MNG.	Reboot the controller.		
2110	Initialization failure. Failed to allocate memory.	Reboot the controller.		
2111	Initialization failure. Failed to initialize motion.	Restore the controller configuration.		
2112	Initialization failure. Failed to terminate motion.	Reboot the controller.		
2113	Initialization failure. Failed to map SRAM.	Replace the CPU board.		
2114	Initialization failure. Failed to register SRAM.	Replace the CPU board.		
2115	Initialization failure. Fieldbus board is beyond the maximum count.			
2116	Initialization failure. Failed to initialize fieldbus.			

No.	Message	Remedy	Note 1	Note 2
2117	Initialization failure. Failed to terminate fieldbus.			
2118	Initialization failure. Failed to open motion.	Restore the controller configuration.		
2119	Initialization failure. Failed to initialize conveyor tracking.	Make sure the settings of conveyor and encoder are correct.		
2120	Initialization failure. Failed to allocate the system area.	Reboot the controller.		
2121	Initialization failure. Failed to allocate the object file area.	Reboot the controller.		
2122	Initialization failure. Failed to allocate the robot area.	Reboot the controller.		
2123	Initialization failure. Failed to create event.	Reboot the controller.		
2130	MCD failure. Failed to open the MCD file.	Restore the controller configuration.		
2131	MCD failure. Failed to map the MCD file.	Restore the controller configuration.		
2132	PRM failure. PRM file cannot be found.	Restore the controller configuration.		
2133	PRM failure. Failed to map the PRM file.	Restore the controller configuration.		
2134	PRM failure. PRM file contents error.	Restore the controller configuration.		
2135	PRM failure. Failed to convert the PRM file.			
2136	PRM failure. Failed to convert the PRM file.			
2137	PRM failure. Failed to convert the PRM file.			
2150	Operation failure. Task number cannot be found.			
2151	Operation failure. Executing the task.			
2152	Operation failure. Object code size failure.			
2153	Operation failure. Jog parameter failure.			
2154	Operation failure. Executing jog.			
2155	Operation failure. Cannot execute the jog function.			
2156	Operation failure. Jog data is not configured.			
2157	Operation failure. Failed to change the jog parameter.			
2158	Operation failure. Failed to allocate the area for the break point.			
2159	Operation failure. Break point number is beyond the allowable setup count.			
2160	Operation failure. Failed to allocate the function ID.			

No.	Message	Remedy	Note 1	Note 2
2161	Operation failure. Failed to allocate the local variable address.			
2162	Operation failure. Not enough buffer to store the local variable.			
2163	Operation failure. Value change is available only when the task is halted.			
2164	Operation failure. Failed to allocate the global variable address.			
2165	Operation failure. Not enough buffer to store the global variable.			
2166	Operation failure. Failed to obtain the Global Preserve variable address.			
2167	Operation failure. Not enough buffer to store the Global Preserve variable.			
2168	Operation failure. SRAM is not mapped.			
2169	Operation failure. Cannot clear the Global Preserve variable when loading the object file.			
2170	Operation failure. Not enough buffer to store the string.			
2171	Operation failure. Cannot start the task after low voltage was detected.			
2172	Operation failure. Duplicated remote I/O configuration.			
2173	Remote setup error. Cannot assign non-existing input number to remote function.			
2174	Remote setup error. Cannot assign non-existing output number to remote function.			
2175	Operation failure. Remote function is not configured.			
2176	Operation failure. Event wait error.			
2177	Operation failure. System backup failed.			
2178	Operation failure. System restore failed.			
2179	Remote setup error. Cannot assign same input number to some remote functions.			
2180	Remote setup error. Cannot assign same output number to some remote functions.			
2190	Cannot calculate because it was queue data.	Check the program.		
2192	Cannot execute AbortMotion because robot task is already finished.			

No.	Message	Remedy	Note 1	Note 2
2193	Cannot execute Recover without motion because AbortMotion was not executed.	Execute AbortMotion in advance to execute Recover WithoutMove.		
2194	Conveyor setting error.	Make sure the settings of conveyor and encoder are correct.		
2195	Conveyor setting error.	Make sure the settings of conveyor and encoder are correct.		
2196	Conveyor number is out of range.	Make sure the settings of conveyor and encoder are correct.		
2200	Robot in use. Cannot execute the motion command when other tasks are using the robot.	The motion command for the robot cannot be simultaneously executed from more than one task. Review the program.		
2201	Robot does not exist.	Check whether the robot setting is performed properly. Restore the controller configuration.		
2202	Motion control module status failure. Unknown error was returned.			
2203	Cannot clear local number ' 0 '.	The Local number 0 cannot be cleared. Review the program.		
2204	Cannot clear an arm while in use.	The Arm cannot be cleared while it is in use. Check whether the Arm is not used.	The Arm number you attempted to clear	
2205	Cannot clear arm number ' 0 '.	The Arm number 0 cannot be cleared. Review the program.		
2206	Cannot clear a tool while in use.	The Tool cannot be cleared while it is in use. Check whether the Tool is not used.	The Tool number you attempted to clear	
2207	Cannot clear tool number ' 0 '.	The Tool number 0 cannot be cleared. Review the program.		
2208	Cannot clear ECP ' 0 '.	The ECP number 0 cannot be cleared. Review the program.		
2209	Cannot clear an ECP while in use.	The ECP cannot be cleared while it is in use. Check whether the ECP is not used.	The ECP number you attempted to clear	
2210	Cannot specify ' 0 ' as the local number.	The command processing the Local cannot specify the Local number 0. Review the program.		
2216	Box number is out of range.			
2217	Box number is not defined.			
2218	Plane number is out of range.			
2219	Plane number is not defined.			
2220	PRM failure. No PRM file data is found.	Reboot the controller. Restore the controller configuration.		
2221	PRM failure. Failed to flash the PRM file.	Reboot the controller. Restore the controller configuration.		
2222	Local number is not defined.	Check the Local setting. Review the program.	The specified Local number	

EPSON RC+ 5.0 (Ver.5.4) SPEL+ Language Reference Rev.4 499

No.	Message	Remedy	Note 1	Note 2
2223	Local number is out of range.	Available Local number is from 1 to 15. Review the program.	The specified Local number	
2224	Unsupported. MCOFS is not defined			
2225	CalPls is not defined.	Check the CalPls setting.		
2226	Arm number is out of range.	Available Arm number is from 0 to 3. Depending on commands, the Arm number 0 is not available. Review the program.	The specified Arm number	
2227	Arm number is not defined.	Check the Arm setting. Review the program.	The specified Arm number	
2228	Pulse for the home position is not defined.	Check the HomeSet setting.		
2229	Tool number is out of range.	Available Tool number is from 0 to 3. Depending on commands, the Tool number 0 is not available. Review the program.	The specified Tool number	
2230	Tool number is not defined.	Check the Tool setting. Review the program.	The specified Tool number	
2231	ECP number is out of range.	Available Tool number is from 0 to 15. Depending on commands, the Tool number 0 is not available. Review the program.	The specified ECP number	
2232	ECP number is not defined.	Check the ECP setting. Review the program.	The specified ECP number	
2233	Axis to reset the encoder was not specified.	Be sure to specify the axis for encoder reset.		
2234	Cannot reset the encoder with motor in the on state.	Turn the motor power OFF before reset.		
2235	XYLIM is not defined.	Check the XYLim setting. Review the program.		
2236	PRM failure. Failed to set up the PRM file contents to the motion control status module.	Reboot the controller. Restore the controller configuration.		
2240	Array subscript is out of user defined range. Cannot access or update beyond array bounds.	Check the array subscript. Review the program.	The dimensions exceeding the definition	The specified subscript
2241	Dimensions of array do not match the declaration.	Check the array's dimensions. Review the program.		
2242	Zero '0' was used as a divisor.	Review the program.		
2243	Variable overflow. Specified variable was beyond the maximum allowed value.	Check the variable type and calculation result. Review the program.		
2244	Variable underflow. Specified variable was below the minimum allowed value.	Check the variable type and calculation result. Review the program.		
2245	Cannot execute this command with a floating point number.	This command cannot be executed for Real or Double type. Review the program.		
2246	Cannot calculate the specified value using the Tan function.	Check the specified value. Review the program.	The specified value	
2247	Specified array subscript is less than ' 0 '.	Check the specified value. Review the program.	The specified value	

No.	Message	Remedy	Note 1	Note 2
2248	Array failure. Redim can only be executed for an array variable.	You attempted to redimension the variable that is not array. Rebuild the project.		
2249	Array failure. Cannot specify Preserve for other than a single dimension array.	Other than a single dimension array was specified as Preserve for Redim. Rebuild the project.		
2250	Array failure. Failed to calculate the size of the variable area.	Rebuild the project.		
2251	Cannot allocate enough memory for Redim statement.	Reduce the number of subscripts to be specified for Redim. Perform Redim modestly.		
2252	Cannot allocate enough memory for ByRef.	Reduce the number of array's subscripts to be seen by ByRef.		
2253	Cannot compare characters with values.	Check whether the string type and the numeric data type are not compared. Review the program.		
2254	Specified data is beyond the array bounds. Cannot refer or update beyond the array bounds.	Check the number of array's subscripts and data. Review the program.	The number of array subscripts	The number of data to be referred or updated
2255	Variable overflow or underflow. Specified variable is out of value range.	The value that exceeds the range of Double type is specified. Review the program.		
2256	Specified array subscript is beyond the maximum allowed range.	Reduce the number of subscripts to be specified. For available subscripts, see the online help.		
2260	Task number is out of the available range.	For available task number, see the online help. Review the program.	The specified task number	
2261	Specified task number does not exist.	Review the program.	The specified task number	
2262	Robot number is out of the available range.	The available Robot number is 1. Review the program.	The specified robot number	
2263	Output number is out of the available range. The Port No. or the Device No. is out of the available range.	For available output number, see the online help. Review the program.	The specified output number	
2264	Command argument is out of the available range. Check the validation. Added data 1: Passed value. Added data 2: argument order.	For available range of argument, see the online help. Review the program.	The specified value	What number argument?
2265	Joint number is out of the available range.	Available Joint number is from 1 to 6. Review the program.	The specified joint number	
2266	Wait time is out of available range.	Available wait time is from 0 to 2147483. Review the program.	The specified wait time	
2267	Timer number is out of available range.	Available timer number is from 0 to 15. Review the program.	The specified timer number	
2268	Trap number is out of available range.	Available trap number is from 1 to 4. Review the program.	The specified trap number	
2269	Language ID is out of available range.	For available language ID, see the online help. Review the program.	The specified language ID	

No.	Message	Remedy	Note 1	Note 2
2270	Specified D parameter value at the parallel process is out of available range.	Available D parameter value is from 0 to 100. Review the program.	The specified D parameter value	
2271	Arch number is out of available range.	Available arch number is from 0 to 7. Review the program.	The specified arch number	
2272	Device No. is out of available range.	The specified number representing a control device or display device is out of available range. For available device number, see the online help. Review the program.	The specified device number	
2273	Output data is out of available range.	Available output data value is from 0 to 255. Review the program.	Output data	What number byte data is out of range?
2274	Asin argument is out of available range. Range is from -1 to 1.	Review the program.		
2275	Acos argument is out of available range. Range is from -1 to 1.	Review the program.		
2276	Sqr argument is out of available range.	Review the program.		
2277	Randomize argument is out of available range.	Review the program.		
2278	Sin, Cos, Tan argument is out of available range.	Review the program.		
2280	Timeout period set by the TMOut statement expired before the wait condition was completed in the WAIT statement.	Investigate the cause of timeout. Check whether the set timeout period is proper.	Timeout period	
2281	Timeout period set by TMOut statement in WaitSig statement or SyncLock statement expired.	Investigate the cause of timeout. Check whether the set timeout period is proper.	Signal number	Timeout period
2282	Timeout period set by TMOut statement in WaitNet statement expired.	Investigate the cause of timeout. Check whether the set timeout period is proper.	Port number	Timeout period
2283	Timeout. Timeout at display device setting.	Reboot the controller.		
2290	Cannot execute a motion command.	Cannot execut the motion command after using the user function in the motion command. Review the program.		
2291	Cannot execute the OnErr command.	Cannot execute OnErr in the motiion command when using user function in the motion command. Review the program.		
2292	Cannot execute an I/O command while the safeguard is open. Need Forced.			
2293	Cannot execute an I/O command during emergency stop condition. Need Forced.			
2294	Cannot execute an I/O command when an error has been detected. Need Forced.			
2295	Cannot execute this command from a NoEmgAbort Task.			

No.	Message	Remedy	Note 1	Note 2
2296	One or more source files are updated. Please build the project.	Rebuild the project.		
2297	Cannot execute an I/O command in TEACH mode without the Forced parameter.	-		
2298	Cannot continue execution in Trap SGClose process.	You cannot execute Cont and Recover statements with processing task of Trap SGClose.		
2299	Cannot execute this command. Need the setting.	Enable the [enable the advance taskcontrol commands] from RC+ to execute the command.		
2300	Robot in use. Cannot execute the motion command when other task is using the robot.	The motion command for the robot cannot be simultaneously executed from more than one task. Review the program.	Task number that is using the robot	
2301	Cannot execute the motion command when the Enable Switch is OFF.			
2302	Cannot execute a Call statement in a Trap Call process.	Another function cannot be called from the function called by Trap Call. Review the program.		
2303	Cannot execute a Call statement in a parallel process.	Review the program.		
2304	Cannot execute an Xqt statement in a parallel process.	Review the program.		
2305	Cannot execute a Call statement from the command window.			
2306	Cannot execute an Xqt statement from the task started by Trap Xqt.	Review the program.		
2307	Cannot execute this command while tasks are executing.	Check whether all tasks are completed.		
2308	Cannot turn on the motor because of a critical error.	Find the previously occurring error in the error history and resolve its cause. Then, reboot the controller.		
2309	Cannot execute a motion command while the safeguard is open.	Check the safeguard status.		
2310	Cannot execute a motion command while waiting for continue.	Execute the Continue or Stop and then execute the motion command.		
2311	Cannot execute a motion command during the continue process.	Wait until the Continue is complete and then execute the motion command.		
2312	Cannot execute a task during emergency stop condition.	Check the emergency stop status.		
2313	Cannot continue execution immediately after closing the safeguard.	Wait 1.5 seconds after the safeguard is open, and then execute the Continue.		
2314	Cannot continue execution while the safeguard is open.	Check the safeguard status.		
2315	Duplicate execution continue.	Wait until the Continue is completed.		
2316	Cannot continue execution after an error has been detected.	Check the error status.		
2317	Cannot execute the task when an error has been detected.	Reset the error by Reset and then execute the task.		

No.	Message	Remedy	Note 1	Note 2
2318	Cannot execute a motion command when an error has been detected.			
2319	Cannot execute a I/O command during emergency stop condition.			
2320	Function failure. Argument type does not match.	Rebuild the project.		
2321	Function failure. Return value does not match to the function.	Rebuild the project.		
2322	Function failure. ByRef type does not match.	Rebuild the project.		
2323	Function failure. Failed to process the ByRef parameter.	Rebuild the project.		
2324	Function failure. Dimension of the ByRef parameter does not match.	Rebuild the project.		
2325	Function failure. Cannot use ByRef in an Xqt statement.	Rebuild the project.		
2326	Cannot execute a Dll Call statement from the command window.	-		
2327	Failed to execute a Dll Call.	-		
2328	Cannot execute the task before connect with RC+.	You need to connect with RC+ before executing the task.		
2329	Cannot execute a Eval statement in a Trap Call process.	Check the program.		
2330	Trap failure. Cannot use the argument in Trap Call or Xqt statement.	Check the program.		
2331	Trap failure. Failed to process Trap Goto statement.	Rebuild the project.		
2332	Trap failure. Failed to process Trap Goto statement.	Rebuild the project.		
2333	Trap failure. Trap is already in process.	Rebuild the project.		
2334	Cannot execute a Eval statement in a Trap Finsh and Trap Abort process.	Check the program.		
2335	Cannot continue execution and Reset Error in TEACH mode.	Check the program.		
2336	Cannot use Here statement with a parallel process.	Go Here :Z(0) ! D10; MemOn(1) ! is not executable. Change the program to: P999 = Here Go P999 Here :Z(0) ! D10; MemOn(1) !		
2337	Cannot execute except from the event handler function of GUI Builder	Check the program.		
2340	Value allocated in InBCD function is an invalid BCD value.	Review the program.	Tens digit	Units digit
2341	Specified value in the OpBCD statement is an invalid BCD value.	Review the program.	The specified value	

No.	Message	Remedy	Note 1	Note 2
2342	Cannot change the status for output bit configured as remote output.	Check the remote I/O setting.	I/O number	1: bit, 2: byte, 3: word
2343	Output time for asynchronous output commanded by On or Off statement is out of the available range.	Review the program.	The specified time	
2344	I/O input/output bit number. is out of available range or the board is not installed.	Review the program. Check whether the expansion I/O board and Fieldbus I/O board are correctly detected.	Bit number	
2345	I/O input/output byte number is out of available range or the board is not installed.	Review the program. Check whether the expansion I/O board and Fieldbus I/O board are correctly detected.	Byte number	
2346	I/O input/output word No. is out of available range or the board is not installed.	Review the program. Check whether the expansion I/O board and Fieldbus I/O board are correctly detected.	Word number	
2347	Memory I/O bit number is out of available range.	Review the program.	Bit number	
2348	Memory I/O byte number is out of available range.	Review the program.	Byte number	
2349	Memory I/O word number is out of available range.	Review the program.	Word number	
2350	Command allowed only when virtual I/O mode is active.	The command can be executed only for virtual I/O mode.		
2351	Cannot change the status for CC-Link system area.			
2352	Remote setup error. Cannot assign CC-Link system area to remote function.			
2360	File failure. Failed to open the configuration file.	Restore the controller configuration.		
2361	File failure. Failed to close the configuration file.	Restore the controller configuration.		
2362	File failure. Failed to open the key of the configuration file.	Restore the controller configuration.		
2363	File failure. Failed to obtain the string from the configuration file.	Restore the controller configuration.		
2364	File failure. Failed to write in the configuration file.	Restore the controller configuration.		
2365	File failure. Failed to update the configuration file.	Restore the controller configuration.		
2370	The string combination exceeds the maximum string length.	The maximum string length is 255. Review the program.	Combined string length	
2371	String length is out of range.	The maximum string length is 255. Review the program.	The specified length	
2372	Invalid character is specified after the ampersand in the Val function.	Review the program.		
2373	Illegal string specified for the Val function.	Review the program.		
2374	String Failure. Invalid character code in the string.	Review the program.		

No.	Message	Remedy	Note 1	Note 2
2380	Cannot use ' 0 ' for Step value in For...Next.	Check the Step value.		
2381	Relation between For...Next and GoSub is invalid. Going in or out of a For...Next using a Goto statement.	Review the program.		
2382	Cannot execute Return while executing OnErr.	Review the program.		
2383	Return was used without GoSub. Review the program.	Review the program.		
2384	Case or Send was used without Select. Review the program.	Review the program.		
2385	Cannot execute EResume while executing GoSub.	Review the program.		
2386	EResume was used without OnErr. Review the program.	Review the program.		
2400	Curve failure. Failed to open the Curve file.	Reboot the controller. Create a Curve file again.		
2401	Curve failure. Failed to allocate the header data of the curve file.	Reboot the controller. Create a Curve file again.		
2402	Curve failure. Failed to write the curve file.	Reboot the controller. Create a Curve file again.		
2403	Curve failure. Failed to open the curve file.	Reboot the controller. Create a Curve file again.		
2404	Curve failure. Failed to update the curve file.	Reboot the controller. Create a Curve file again.		
2405	Curve failure. Failed to read the curve file.	Reboot the controller. Create a Curve file again.		
2406	Curve failure. Curve file is corrupt.	Reboot the controller. Create a Curve file again.		
2407	Curve failure. Specified a file other than the curve file.	Reboot the controller. Create a Curve file again.		
2408	Curve failure. Version of the curve file is invalid.	Reboot the controller. Create a Curve file again.		
2409	Curve failure. Robot number in the curve file is invalid.	Reboot the controller. Create a Curve file again.		
2410	Curve failure. Cannot allocate enough memory for the CVMove statement.	Reboot the controller.		
2411	Specified point data in the Curve statement is beyond the maximum count.	The maximum number of points specified in the Curve statement is 200. Review the program.		
2412	Specified number of output commands in the Curve statement is beyond the maximum count.	The maximum number of output commands specified in the Curve statement is 16. Review the program.		
2413	Curve failure. Specified internal code is beyond the allowable size in Curve statement.	Reboot the controller.		
2414	Specified continue point data P(:) is beyond the maximum count.	The maximum number of points specified continuously is 200. Review the program.	Start point	End point
2415	Curve failure. Cannot create the curve file.	Reboot the controller. Create a Curve file again.		

No.	Message	Remedy	Note 1	Note 2
2416	Curve file does not exist.	Check whether the specified Curve file name is correct.		
2417	Curve failure. Output command is specified before the point data.	Check whether no output command is specified before the point data.		
2418	Curve file name is too long.	Check whether the specified Curve file name is correct. The maximum string length of the file name is 32.		
2419	Curve failure. Curve file path is too long.	Check whether the specified Curve file name is correct.		
2420	Curve file name is invalid.			
2430	Error message failure. Error message file does not exist.	Reboot the controller.		
2431	Error message failure. Failed to open the error message file.	Reboot the controller.		
2432	Error message failure. Failed to obtain the header data of the error message file.	Reboot the controller.		
2433	Error message failure. Error message file is corrupted.	Reboot the controller.		
2434	Error message failure. Specified a file other than the error message file.	Reboot the controller.		
2435	Error message failure. Version of the error message file is invalid.	Reboot the controller.		
2440	File Error. File number is used.	Check the file number.		
2441	File Error. Failed to open the file.	Make sure the file exists and you specified the file correctly.		
2442	File Error. The file is not open.	Open the file in advance.		
2443	File Error. The file number is being used by another task.	Check the program.		
2444	File Error. Failed to close the file.			
2445	File Error. File seek failed.			
2446	File Error. All file numbers are being used.			
2447	File Error. No read permision.	Use ROpen or UOpen that has read access to the file.		
2448	File Error. No write permision.	Use WOpen or UOpen that has write access to the file.		
2449	File Error. No binary permision.	Use BOpen that has binary access to the file.		
2450	File Error. Failed to access the file.			
2451	File Error. Failed to write the file.			
2452	File Error. Failed to read the file.			
2453	File Error. Cannot execute the commnad for current disk.	The specified command is not available in the current disk (ChDisk).		
2454	File Error. Invalid disk.			

No.	Message	Remedy	Note 1	Note 2
2455	File Error. Invalid drive.			
2456	File Error. Invalid folder.			
2460	Database Error. The database number is already being used.			
2461	Database Error. The database is not open.			
2462	Database Error. The database number is being used by another task.			
2470	Windows Communication Error. Invalid status.			
2471	Windows Communication Error. Invalid answer.			
2472	Windows Communication Error. Already initialized.			
2473	Windows Communication Error. Busy.			
2474	Windows Communication Error. No request.			
2475	Windows Communication Error. Data buffer overflow.			
2476	Windows Communication Error. Failed to wait for event.			
2477	Windows Communication Error. Invalid folder.	Make sure the specified folder is correct.		
2478	Windows Communication Error. Invalid error code.			
2500	Specified event condition for Wait is beyond the maximum count.	The maximum number of event conditions is 8. Review the program.		
2501	Specified bit number in the Ctr function was not setup with a CTReset statement.	Review the program.	The specified bit number	
2502	Task number is beyond the maximum count to execute.	The available number of the tasks that can be executed simultaneously is 16. Review the program.		
2503	Cannot execute Xqt when the specified task number is already executing.	Review the program.	The specified task number	
2504	Task failure. Specified manipulator is already executing a parallel process.	Rebuild the project.		
2505	Not enough data for Input statement variable assignment.	Check the content of communication data. Review the program.		
2506	Specified variable for the Input statement is beyond the maximum count.	For OP, only one variable can be specified. For other devices, up to 32 variables can be specified.		
2507	All counters are in use and cannot setup a new counter with CTReset.	The available number of the counters that can be set simultaneously is 16. Review the program.		
2508	OnErr failure. Failed to process the OnErr statement.	Rebuild the project.		
2509	OnErr failure. Failed to process the OnErr statement.	Rebuild the project.		

No.	Message	Remedy	Note 1	Note 2
2510	Specified I/O label is not defined.	The specified I/O label is not registered. Check the I/O label file.		
2511	SyncUnlock statement is used without executing a previous SyncLock statement. Review the program.	Review the program.	Signal number	
2512	SyncLock statement was already executed.	The SyncLock statement cannot be executed for the second time in a row. Review the program.	Signal number	
2513	Specified point label is not defined.	The specified point label is not registered. Check the point file.		
2514	Failed to obtain the motor on time of the robot.	Reboot the controller.		
2515	Failed to configure the date or the time.	Check whether a date and time is set correctly.		
2516	Failed to obtain the debug data or to initialize.	Reboot the controller.		
2517	Failed to convert into date or time.	Check the time set on the controller. Reboot the controller.		
2518	Larger number was specified for the start point data than the end point data .	Specify a larger number for the end point data than that for the start point data.	Start point	End point
2519	Specified the format for FmtStr$ can not understand.	Check the format.		
2520	Point file name is too long.	Check whether the specified point file name is correct. The maximum string length of the file name is 32.		
2521	Point failure. Point file path is too long.	Check whether the specified point file name is correct.		
2522	Point file name is invalid.			
2523	The continue process was already executed.			
2524	Cannot execute Xqt when the specified trap number is already executing.			
2525	Password is invalid.	Check whether a password is set correctly.		
2526	No wait terms.			
2527	Too many variables used for global valiable wait.			
2528	The variables cannot use global valiable wait.			
2529	Cannot use Byref if the variables used for global variable wait.			
2530	Too many point files.			
2531	The point file is used by another robot.			
2532	Cannot calculate the point position because there is undefined data.			
2533	Error on INP or OUTP.			
2534	No main function to start on Restart statement.	Without executing main function, Restart is called.		
2539	Password is invalid.	Check whether a password is set correctly.		

No.	Message	Remedy	Note 1	Note 2
2900	Failed to open as server to the Ethernet port.	Check whether the Ethernet port is set properly. Check whether the Ethernet cable is connected properly.		
2901	Failed to open as client to the Ethernet port.	Check whether the Ethernet port is set properly. Check whether the Ethernet cable is connected properly.		
2902	Failed to read from the Ethernet port.	Check whether the port of communication recipient is not close.		
2904	Invalid IP Address was specified.			
2905	Ethernet failure. No specification of Server/Client.			
2906	Ethernet port was not configured.	Check whether the Ethernet port is set properly.	Port number	
2907	Ethernet pot was already in use by another task.	A single port cannot be used by more than one task.	Port number	
2908	Cannot change the port parameters while the Ethernet port is open.	The port parameters cannot be changed while the port is open.	Port number	
2909	Ethernet port is not open.	To use the Ethernet port, execute the OpenNet statement.	Port number	
2910	Timeout reading from an Ethernet port.	Check the communication.	Timeout value	
2911	Failed to read from an Ethernet port.	Check the communication.		
2912	Ethernet port was already open by another task.	A single port cannot be used by more than one task.	Port number	
2913	Failed to write to the Ethernet port.	Check whether the Ethernet port is set properly. Check whether the Ethernet cable is connected properly.	Port number	
2914	Ethernet port connection was not completed.	Check whether the port of communication recipient is open.	Port number	
2915	Data received from the Ethernet port is beyond the limit of one line.	The maximum length of a line is 255 bytes.	The number of bytes in a received line	
2920	RS-232C failure. RS-232C port process error.	Check whether the RS-232C board is correctly detected.		
2921	RS-232C failure. Uncommon error. RS-232C port read process error.			
2926	The RS-232C port hardware is not installed.	Check whether the RS-232C board is correctly detected.	Port number	
2927	RS-232C port is already open by another task.	A single port cannot be used by more than one task.	Port number	
2928	Cannot change the port parameters while the RS-232C port is open.	The port parameters cannot be changed while the port is open.	Port number	
2929	RS-232C port is not open.	To use the RS-232C port, execute the OpenCom statement.	Port number	
2930	Timeout reading from the RS-232C port.	Check the communication.	Timeout value	
2931	Failed to read from the RS-232C port.	Check the communication.		
2932	RS-232C port is already open by another task.	A single port cannot be used by more than one task.	Port number	
2933	Failed to write to the RS-232C port.	Check the communication.	Port number	

No.	Message	Remedy	Note 1	Note 2
2934	RS-232C port connection not completed.			
2935	Data received from the RS-232C port is beyond the limit of one line.	The maximum length of a line is 255 bytes.	The number of bytes in a received line	
2937	Cannot execute while Remote RS-232C are useing.			
2950	Daemon failure. Failed to create the daemon thread.			
2951	Daemon failure. Timeout while creating the daemon thread.			
2952	TEACH/AUTO switching key input signal failure was detected.	Set the TP key switch to TEACH or AUTO properly. Check whether the TP is connected properly.		
2953	ENABLE key input signal failure was detected.	Check whether the TP is connected properly.		
2954	Relay weld was detected.	Overcurrent probably occurred due to short-circuit failure. Investigate the cause of the problem and take necessary measures and then replace the DPB.		
2955	Temperature of regeneration resistor was higher than the specified temperature.	Check whether the filter is not clogged up and the fan does not stop. If there is no problem on the filter and fan, replace the regenerative module.		
2970	MNG failure. Area allocate error.			
2971	MNG failure. Real time check error.			
2972	MNG failure. Standard priority error.			
2973	MNG failure. Boost priority error.			
2974	MNG failure. Down priority error.			
2975	MNG failure. Event wait error.			
2976	MNG failure. Map close error.			
2977	MNG failure. Area free error.			
2978	MNG failure. AddIOMem error.			
2979	MNG failure. AddInPort error.			
2980	MNG failure. AddOutPort error.			
2981	MNG failure. AddInMemPort error.			
2982	MNG failure. AddOutMemPort error.			
2983	MNG failure. IntervalOutBit error.			
2984	MNG failure. CtrReset error.			
2998	AbortMotion attempted when robot was not moving	See Help for AbortMotion.		
2999	AbortMotion attempted when robot was moving	See Help for AbortMotion.		

Parser

No.	Message	Remedy	Note 1	Note 2
3000	OBJ file size is large. TP1 may not be able to build this project.			
3050	Main function is not defined.	Declare a Main function.		
3051	Function does not exist.	Declare an unresolved function.		
3052	Variable does not exist.	Declare an unresolved variable.		
3100	Syntax error.	Correct the syntax error.		
3101	Parameter count error.	The number of parameters is excess or deficiency. Correct the parameters.		
3102	File name length is beyond the maximum allowed.	Shorten the file name.		
3103	Duplicate function definition.	Change the function name.		
3104	Duplicate variable definition ' ** '.	Change the variable name.		
3105	Global and Global Preserve variables cannot be defined inside a function block.	Declare the Global and Global Preserve variables outside the function block.		
3106	An undefined function was specified.	Specify a valid function name.		
3107	Both While and Until for Do...Loop was specified.	The While/Until statement is specified for both Do statement and Loop statement. Delete either While/Until statement.		
3108	Specified line number or label ' ** ' does not exist.	Set the line label.		
3109	Overflow error.	The direct numerical specification overflows. Reduce the numeric value.		
3110	An undefined variable was specified ' ** '.	There is an undefined variable. Declare the variable.		
3111	Specified variable is not an array variable.	Specify the array variable.		
3112	Cannot change the dimensions of the array variable.			
3114	Specified Next variable does not match the specified For variable.	Correct the variable name.		
3115	Cannot use a point expression in the first argument.	Specify a single point for the point flag setting. Do not specify a point expression.		
3116	Array number of dimensions does not match the declaration.	Check the number of array dimensions.		
3117	File cannot be found.			
3118	Corresponding EndIf cannot be found.	The number of EndIf statements is not enough. Add the EndIf.		
3119	Corresponding Loop cannot be found.	The number of Loop statements is not enough. Add the Loop.		
3120	Corresponding Next cannot be found.	The number of Next statements is not enough. Add the Next.		
3121	Corresponding Send cannot be found.	The number of Send statements is not enough. Add the Send.		
3123	On/Off statements are beyond the maximum count.	An upper limit is set on the number of On/Off statements. Check the upper limit and correct the program.		

No.	Message	Remedy	Note 1	Note 2
3124	Point number is beyond the maximum count.	An upper limit is set on the available number of points. Check the upper limit and correct the program.		
3125	Corresponding If cannot be found.	The number of EndIf statements is too many. Delete the unnecessary EndIf.		
3126	Corresponding Do cannot be found.	The number of Loop statements is too many. Delete the unnecessary Loop.		
3127	Corresponding Select cannot be found.	The number of Send statements is too many. Delete the unnecessary Send.		
3128	Corresponding For cannot be found.	The number of Next statements is too many. Delete the unnecessary Next.		
3129	'_' cannot be used as the first character of an identifier.	Change the first character of the identifier to an alphabetic character.		
3130	Cannot specify Rot parameter.			
3131	Cannot specify Ecp parameter.			
3132	Cannot specify Arch parameter.			
3133	Cannot specify LimZ parameter.			
3134	Cannot specify Sense parameter.			
3135	Invalid parameter is specified.			
3136	Cannot use #include.			
3137	Cannot specify the array variable subscript.	The array variable subscript cannot be specified.		
3138	ByRef was not specified on Function declaration.			
3139	Cannot execute the Xqt statement for a function that needs a ByRef parameter.	The Xqt statement cannot be executed for a function needing a ByRef parameter. Delete the ByRef parameter.		
3140	Cannot execute the Redim statement for a ByRef variable.			
3141	OBJ file is corrupt.			
3142	OBJ file size is beyond the available size after compiling.	The compilation result exceeds the limit value. Divide the program.		
3143	Ident length is beyond the available size.			
3144	' ** ' already used for a function name.			
3145	' ** ' already used for a Global Preserve variable.			
3146	' ** ' already used for a Global variable.			
3147	' ** ' already used for a Module variable.			
3148	' ** ' already used for a Local variable.			
3149	' ** ' already used for a I/O label.			
3150	' ** ' already used for a User Error label.			

No.	Message	Remedy	Note 1	Note 2
3151	Cannot use a function parameter.	Argument cannot be specified for the function that is executed by the Trap statement.		
3152	Over elements value.			
3153	Parameter type mismatch.			
3154	' ** ' is not Input Bit label.			
3155	' ** ' is not Input Byte label.			
3156	' ** ' is not Input Word label.			
3157	' ** ' is not Output Bit label.			
3158	' ** ' is not Output Byte label.			
3159	' ** ' is not Output Word label.			
3160	' ** ' is not Memory Bit label.			
3161	' ** ' is not Memory Byte label.			
3162	' ** ' is not Memory Word label.			
3163	Too many function arguments.			
3164	Cannot compare Boolean value.			
3165	Cannot use Boolean value in the expression.			
3166	Cannot compare between Boolean and expression.			
3167	Cannot store Boolean value to the numeric variable.			
3168	Cannot store numeric value to the Boolean variable.			
3169	Undefined I/O label was specified.			
3170	Invalid condition expression was specified.			
3171	Cannot compare between numeric value and string.			
3172	Cannot use keyword for the variable name.			
3172	' ** ' already used for a line label.			
3173	Duplicate line number or label (**).			
3175	Undefined Point label was specified.			
3176	An undefined variable was specified.			
3177	' ** ' already used for a Point label.			
3178	Cannot use the result number.			
3179	String literal is beyond the available length.			
3180	Cannot change a calibration property value with the VSet command.			
3181	Array variable should be used with ByRef.			
3182	Subscription was not specified.			
3187	Invalid Point flag value was specified.			
3200	Value is missing.			
3201	Expected ' , '.			
3202	Expected ' ('.			
3203	Expected ') '.			

No.	Message	Remedy	Note 1	Note 2
3204	Identifier is missing.			
3205	Point is not specified.			
3206	Event condition expression is missing.			
3207	Formula is missing.			
3208	String formula is missing.			
3209	Point formula is missing.			
3210	Line label was not specified.			
3211	Variable was not specified.			
3212	Corresponding Fend cannot be found.			
3213	Expected ' : '.			
3214	True/False was not specified.			
3215	On/Off was not specified.			
3216	High/Low was not specified.			
3217	Input bit label was not specified.			
3218	Input byte label was not specified.			
3219	Input word label was not specified.			
3220	Output bit label was not specified.			
3221	Output byte label was not specified.			
3222	Output word label was not specified.			
3223	Memory bit label was not specified.			
3224	Memory byte label was not specified.			
3225	Memory word label was not specified.			
3226	User error label was not specified.			
3227	Function name was not specified.			
3228	Variable type was not specified.			
3229	Invalid Trap statement parameter. Use Goto, Call, or Xqt.			
3230	Expected For/Do/Function.			
3231	Above/Below was not specified.			
3232	Righty/lefty was not specified.			
3233	NoFlip/Flip was specified.			
3234	Port number was not specified.			
3235	String type variable was not specified.			
3236	RS-232C port number was not specified.			
3237	Network communication port number was not specified.			
3238	Communication speed was not specified.			
3239	Data bit number was not specified.			
3240	Stop bit number was not specified.			
3241	Parity was not specified.			
3242	Terminator was not specified.			

No.	Message	Remedy	Note 1	Note 2
3243	Hardware flow was not specified.			
3244	Software flow was not specified.			
3245	None was not specified.			
3246	Parameter ' O ' or ' C ' was not specified.			
3247	NumAxes parameter was not specified.			
3248	J4Flag value (0-1) was not specified.			
3249	J6Flag value (0-127) was not specified.			
3250	Array variable was not specified.			
3251	String Array variable was not specified.			
3252	Device ID was not specified.			
3253	I/O type was not specified.			
3254	I/O bit width was not specified.			
3255	ByRef was not specified.	Although the ByRef is specified in the function declaration, no ByRef is specified for calling.		
3256	Variable type was not specified.			
3257	Condition expression does not return Boolean value.			
3258	RS232C port number was not specified.			
3259	Network communication port number was not specified.			
3260	Language ID was not specified.			
3261	Expected '.'.			
3262	Vision Sequence Name was not specified.			
3263	Vision Sequence Name or Calibration Name was not specified.			
3264	Vision Property Name or Result Name was not specified.			
3265	Vision Property Name, Result Name or Object Name was not specified.			
3266	Vision Calibration Property Name was not specified.			
3267	Task type was not specified.			
3300	External definition symbol was included. (Not in use)			
3301	Version of linked OBJ file does not match.	Not all project files are complied in the same version. Perform the rebuild.		
3302	Linked OBJ file does not match the compiled I/O label.	The project configuration has been changed. Perform the rebuild.		
3303	Linked OBJ file does not match the compiled user error label.	The project configuration has been changed. Perform the rebuild.		
3304	Linked OBJ file does not match the compiled compile option.	The project configuration has been changed. Perform the rebuild.		

No.	Message	Remedy	Note 1	Note 2
3305	Linked OBJ file does not match the compiled link option.	The project configuration has been changed. Perform the rebuild.		
3306	Linked OBJ file does not match the compiled SPEL option.	The project configuration has been changed. Perform the rebuild.		
3307	Duplicate function.	The same function name is used for more than one file.		
3308	Duplicate global preserve variable.	The same global preserve variable name is used for more than one file.		
3309	Duplicate global variable.	The same global variable name is used for more than one file.		
3310	Duplicate module variable.	The same module variable name is used for more than one file.		
3311	File cannot be found.			
3312	OBJ file is corrupt.			
3313	The specified file name includes character(s) that cannot be used.			
3314	Cannot open the file.	The file is used for other application. Quit the other application.		
3315	' ** ' is already used for the function name.			
3316	' ** ' is already used for the global preserve variable.			
3317	' ** ' is already used for the global variable.			
3318	' ** ' is already used for the module variable.			
3319	Dimension of the array variable does not match the declaration.			
3320	Return value type of the function does not match the declaration.			
3321	' ** ' is already used with function name.			
3322	' ** ' is already used with Global Preserve name.			
3323	' ** ' is already used with Global name.			
3324	' ** 'is already used with Module name.			
3325	' ** ' is already used with Local name.			
3326	The number of parameters does not match the declaration.			
3327	ByRef was not specified on Function declaration on parameter **.			
3328	ByRef was not specified on parameter **.			
3329	Parameter ** type mismatch.			
3330	Linked OBJ file does not match the compiled Vision Project.			
3331	OBJ file size is beyond the available size after linking.	The OBJ file size exceeds the limit value. Reduce the program.		

No.	Message	Remedy	Note 1	Note 2
3332	Variable '%s' is redefined.	Variable '%s' is redefined with the different variable type. Review the definition.		
3500	Duplicate macro in #define statement.	Another macro with the same name has been defined. Change the macro name.		
3501	Macro name was not specified.			
3502	Include file name cannot be found.			
3503	Specified include file is not in the project.	The include file that is not registered in the project configuration is specified. Add the include file to the project configuration.		
3504	Parameter of the macro function does not match to the declared.			
3505	Macro has a circular reference.	The macro has a circular reference. Correct the circular reference.		
3506	#define, #ifdef, #ifndef, #else, #endif, #undef and variable declaration statements are only valid in an include file.			
3507	Over #ifdef or #ifndef nesting level.	Reduce the nesting level to under the limited value.		
3508	Cannot find corresponding #ifdef or #ifndef.			
3509	No #endif found for #ifdef or #ifndef.			
3510	Cannot obtain the macro buffer.			
3550	Parameter for the macro function was not specified.	The macro declared as a macro function is called without argument.		
3800	Compile process aborted.			
3801	Link process aborted.			
3802	Compile process aborted. Compile errors reached the maximum count.			
3803	Link process aborted. Link errors reached the maximum count.			
3804	Specified command cannot be executed from the Command window.			
3805	Specified command can only be executed from the Command window.			
3806	Specified function cannot be executed from the Command window.			
3810	The number of point file is beyond the limit.	There are too many point files. Reduce some point files that are registered to project.		
3811	The number of points is beyond the limit.	There are too many points defined by registered point files. Reduce some points.		
3850	File not found.			
3900	Uncommon error. Cannot obtain the internal communication buffer.			

No.	Message	Remedy	Note 1	Note 2
3910	Undefined command was specified.			
3911	Cannot enter the file name in the file name buffer.			
3912	Cannot obtain the internal buffer.			
3913	Cannot set priority.			
3914	Invalid ICode.			
3915	Invalid ICode.			
3916	Invalid ICode.			
3917	Invalid ICode.			
3918	Invalid ICode.			
3919	Invalid ICode.			
3920	Invalid ICode.			
3921	Invalid ICode.			

Motor Control

No.	Message	Remedy	Note 1	Note 2
4001	Arm reached the limit of motion range.	Check the point to move, current point, and Range setting.		
4002	Specified value is out of allowable range.	Review the setting parameters.		The parameter causing the error
4003	Motion device driver failure. Communication error within the motion control module.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4004	Motion device driver failure. Event waiting error within the motion control module.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4005	Current point position is above the specified LimZ value.	Lower the Z axis. Increase the specified LimZ value.		
4006	Target point position is above the specified LimZ value.	Lower the Z coordinate position of the target point. Increase the specified LimZ value.		
4007	Coordinates conversion error. The end/mid point is out of the motion area. Jogging to the out of the motion area.	Check whether the coordinate out of the motion range is not specified.		
4008	Current point position or specified LimZ value is out of motion range.	Change the specified LimZ value.		
4009	Motion device driver failure. Timeout error within motion control module.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4010	Specified Local coordinate was not defined.	Define the Local coordinate system.		Local number
4011	Arm reached the limit of XY motion range specified by XYLim statement.	Check the area limited by the XYLim statement.		
4013	Motion control module internal calculation error.			
4016	SFree statement was attempted for prohibited joint(s).	Due to robot mechanistic limitation, setting some joint(s) to servo free status is prohibited. Check the robot specifications.		
4018	Communication error within the motion control module. Check sum error.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4021	Point positions used to define the Local are too close.	Set the distance between points more than 1µm.		
4022	Point coordinate data used to define the Local is invalid.	Match the coordinate data for the points to be specified.		
4023	Cannot execute when the motor is in the off state.	Turn the motor power ON and then execute.		
4024	Cannot complete the arm positioning using the current Fine specification.	Check whether the robot does not generate vibration or all parts and screws are secured firmly. Increase the Fine setting value.		
4025	Cannot execute a motion command during emergency stop condition.	Clear the emergency stop condition and execute the motion command.		

No.	Message	Remedy	Note 1	Note 2
4026	Communication error within the motion control module. Servo I/F failure.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4028	Communication error within the motion control module. Device driver status failure.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4030	Buffer for the average torque calculation has overflowed. Shorten the time interval from Atclr to Atrq.	Shorten the time interval from Atclr to Atrq less than about two minutes.		
4031	Cannot execute a motion command when the motor is in the off state.	Turn the motor power ON and then execute the motion command.		
4032	Cannot execute a motion command when one or more joints are in SFree state.	Set all joints to the SLock state and execute the motion command.		
4034	Specified command is not supported for this manipulator model.	Use the Jump3 and Jump3CP statements.		
4035	Only the tool orientation was attempted to be changed by the CP statement.	Set a move distance between points. Use the ROT modifier, SpeedR statement, and AccelR statement.		
4036	Rotation speed of tool orientation by the CP statement is too fast.	Decrease the setting values for the SpeedS and AccelS statements. Use the ROT modifier, SpeedR statement, and AccelR statement.		
4037	The point attribute of the current and target point positions differ for executing a CP control command.	Match the point attribute.		
4038	Two point positions are too close to execute the Arc statement.	Set the distance between points more than 1μm.		
4039	Three point positions specified by the Arc statement are on a straight line.	Use the Move statement.		
4041	Motion command was attempted to the prohibited area at the backside of the robot.	Check the robot motion range.		
4042	Motion device driver failure. Cannot detect the circular format interruption.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4043	Specified command is not supported for this manipulator model or this joint type.			
4044	Curve failure. Specified curve form is not supported.	Create a Curve file again with the Curve statement.		
4045	Curve failure. Specified mode is not supported.	Specify the Curve mode properly. Create a Curve file again with the Curve statement.		
4046	Curve failure. Specified coordinate number is out of the allowable range.	The number of the available coordinate axes is 2, 3, 4, and 6. Create a Curve file again with the Curve statement.		
4047	Curve failure. Point data was not specified.	Create a Curve file again with the Curve statement.		

No.	Message	Remedy	Note 1	Note 2
4048	Curve failure. Parallel process was specified before the point designation.	Create a Curve file again with the Curve statement.		
4049	Curve failure. Number of parallel processes is out of the allowable range.	Create a Curve file again with the Curve statement.		
4050	Curve failure. Number of points is out of the allowable range.	The number of available point numbers differs according to the curve form. Check the number of points again.		
4051	Curve failure. Local attribute and the point attribute of all specified points do not match.	Match the local and point flag for all the specified points.		
4052	Curve failure. Not enough memory to format the curve file.			
4053	Curve failure. Failed to format the curve file.	Review the point data. Check whether adjacent two points do not overlap on the specified point line.		
4054	Curve failure. Curve file error	The Curve file is broken. Create a Curve file again with the Curve statement.		
4055	Curve failure. No distance for curve file movement.	Review the point data.		
4056	Curve failure. Point positions for the Curve statement are too close.	Set the distance between two points adjacent to the specified point more than 0.001 mm.		
4059	Executed encoder reset command while the motor is in the on state.	Turn the motor power OFF.		
4060	Executed an invalid command while the motor is in the on state.	Turn the motor power OFF.		
4061	Specified parameter is in use.	You attempted to clear the currently specified Arm and Tool. Select other Arm and Tool and execute.		
4062	Orientation variation is over 360 degrees.	You attempted to rotate the joint #J6 more than 360 degrees with a CP motion command.		
4063	Orientation variation of adjacent point is over 90 degrees.	On the specified point line by the Curve statement, set the orientation variation of U, V, and W coordinate values between two adjacent points to under 90 degrees.		
4064	Cannot execute the orientation correction automatically.	On the specified point line, a curve cannot be created by automatic orientation correction. Change the specified point line so that the joint #J6 orientation variation decreases.		
4065	Attempt to revolve J6 one rotation with the same orientation in CP statement.	You attempted to rotate the joint #J6 more than 360 degrees with a CP motion command. You attempted to revolve the joint 6 one rotation with the same as motion start orientation. Change the target point so that the joint #J6 revolves less than one rotation.		

No.	Message	Remedy	Note 1	Note 2
4066	Motion command was attempted in the prohibited area depended on joint combination.	You attempted to move the joints to the robot's interference limited area.		
4068	ROT modifier parameter was specified for the CP motion command without orientation rotation.	Delete the ROT from the CP motion command.		
4069	Specified ECP without selecting ECP in CP statement.	Specify a valid ECP.		
4070	Specified ECP number does not match the ECP number used in curve file creation.	Specify a valid ECP.		
4071	Attempted motion command during electronic brake lock condition.			
4072	Initialization failure. Hardware monitor was not initialized.			
4074	Motor type does not match the current robot setting.	Check whether the specified robot model is connected.		
4075	ECP Option is not active.	Enable the ECP option.		
4076	Point positions used to define the Plane are too close.	Set the distance between points more than 1 μm.		
4077	Point coordinate data used to define the Plane is invalid.	Match the coordinate data for the points to be specified.		
4080	Cannot execute when the Enable Switch is OFF.	Turn the Enable Switch ON and then execute.		
4099	Servo error was detected during operation.			
4100	Communication error in motion control module. Cannot calculate the current point or pulse.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4101	Communication error in the motion control module. Cannot calculate the current point or pulse.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4103	Initialization failure. Motion control module initialization error.	Reboot the controller. Initialize the controller firmware. Replace the controller.		
4105	EMERGENCY connector connection failure.			
4106	Drive unit failure.			
4150	Redundant input signal failure of the emergency stop.	The input status of the redundant emergency stop input continuously differs for more than two seconds. Check whether no disconnection, earth fault, or short-circuit of the emergency stop input signal exits. Then reboot the controller.		
4151	Redundant input signal failure of the safeguard.	The input status of the redundant emergency stop input continuously differs for more than two seconds. Check whether no disconnection, earth fault, or short-circuit of the emergency stop input signal exits. Then reboot the controller.		

No.	Message	Remedy	Note 1	Note 2
4152	Relay welding error of the main circuit.	A relay welding error was detected due to power system over current. Replace the controller. Replace the robot.		
4153	Redundant input signal failure of the enable switch.	The input status of the redundant enable signal differs continuously for more than two seconds. Check the TP connector connection. Replace the TP. Replace the controller.		
4154	Temperature of regeneration resistor was higher than the specified temperature.			
4180	Manipulator initialization failure. Specified manipulator was is not found.			
4181	Manipulator initialization failure. Specified manipulator was in use by another task.			
4182	Manipulator initialization failure. Manipulator name is too long.			
4183	Manipulator initialization failure. Manipulator data version error.			
4184	Manipulator initialization failure. Duplication of single axis joint is assigned.			
4185	Manipulator initialization failure. Specified axis is in use by the other manipulator.			
4186	Manipulator initialization failure. Necessary hardware resource is not defined.			
4187	Manipulator initialization failure. Communication error with the module : VSRCMNPK.			
4188	Manipulator initialization failure. Joint angle interference matrix is invalid.			
4189	Manipulator initialization failure. Communication error with the module : VSRCMC.			
4191	Manipulator initialization failure. Physical-logical pulse transformation matrix is invalid.			
4192	Manipulator initialization failure. Communication error with the servo module.			
4210	RAS circuit detected the servo system malfunction. Reboot the controller. Measure the noise. Replace the controller.			
4211	Servo CPU internal RAM failure. Reboot the controller. Measure the noise. Replace the DMB.			

No.	Message	Remedy	Note 1	Note 2
4212	RAM for the main and servo CPU communication failure. Reboot the controller. Measure the noise. Replace the DMB.			
4213	Servo CPU internal RAM failure. Reboot the controller. Measure the noise. Replace the DMB.			
4214	Initialization communication of main CPU and servo CPU failure. Reboot the Controller. Measure the noise. Replace DMB.			
4215	Initialization communication of the main and servo CPU failure. Reboot the controller. Noise measure. Replace the DMB.			
4216	Communication of the main and servo CPU failure. Reboot the controller. Measure the noise. Replace the DMB.			
4217	Communication of the main and servo CPU failure. Reboot the controller. Measure the noise. Replace the DMB.			
4218	Servo long time command overrun.			
4219	Servo long time command check sum error.			
4220	System watchdog timer detected the failure. Reboot the controller. Measure the noise. Replace the DMB.			
4221	Drive unit check failure.			
4222	RAM failure of the servo CPU. Reboot the controller. Measure the noise. Replace the DMB.			
4223	Failure of duplicate circuit of the emergency stop or the safeguard. Check the wiring.			
4224	Low voltage of the main circuit power supply is detected. Check the power supply voltage. Reboot the controller.			
4225	Control relay contact of the main circuit power supply is welded. Replace the DPB.			
4230	Servo real time status failure. Check sum error.	A data checksum error was detected in the controller. Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency, D-I/O, and Expansion I/O connectors) Replace the controller.		

No.	Message	Remedy	Note 1	Note 2
4232	Servo real time status failure. Free running counter error with the servo.	A free running counter error was detected in the controller. Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency, D-I/O, and Expansion I/O connectors) Replace the controller.		
4233	Servo real time status failure. Communication error with the servo CPU.	A communication error was detected in the controller. Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency, D-I/O, and Expansion I/O connectors) Replace the controller.		
4240	Irregular motion control interruption was detected. Interruption duplicate.	A interruption error was detected in the controller. Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency, D-I/O, and Expansion I/O connectors) Replace the controller.		
4241	Over speed during low power mode was detected.	The robot over speed was detected during low power mode. Check the robot mechanism. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check whether the robot does not interfere with peripheral equipment. (Collision, contact) Replace the motor driver. Replace the motor. (Motor and encoder failure) Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency, D-I/O, and Expansion I/O connectors)		
4242	Improper acceleration reference was generated.	You attempted to operate the robot with the acceleration reference exceeding the specified value. For a CP motion, decrease the AccelS value.		
4243	Improper speed reference is generated in the high power mode.	The robot over speed was detected during high power mode. Check the robot mechanism. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check whether the robot does not interfere with peripheral equipment. (Collision, contact) Replace the motor driver. Replace the motor. (Motor and encoder failure) Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency, D-I/O, and Expansion I/O connectors)		

No.	Message	Remedy	Note 1	Note 2
4250	Arm reached the limit of motion range during the operation.	Check whether a CP motion trajectory is within the motion range.		
4251	Arm reached the limit of XY motion range specified by XYLim during the operation.	Check the XYLim setting.		
4252	Coordinate conversion error occurred during the operation.	Check whether a CP motion trajectory is within the motion range.		
4267	Attempt to exceed the J4Flag attribute without indication.	You attempted to exceed the J4Flag attribute during motion without the J4Flag indication. Change the J4Flag for the target point.		
4268	Attempt to exceed the J6Flag attribute without indication.	You attempted to exceed the J6Flag attribute during motion without the J6Flag indication. Change the J6Flag for the target point.		
4269	Attempt to exceed the particular wrist orientation attribute without indication.	You attempted to exceed the particular wrist orientation attribute during motion without the Wrist indication. Change the Wrist attribute for the target point. Change the target point to avoid a particular wrist orientation.		
4270	Attempt to exceed the particular arm orientation attribute without indication.	You attempted to exceed the particular hand orientation attribute during motion without the Hand indication. Change the Hand attribute for the target point. Change the target point to avoid a particular hand orientation.		
4271	Attempt to exceed the particular elbow orientation attribute without indication.	You attempted to exceed the particular elbow orientation attribute during motion without the Elbow indication. Change the Elbow attribute for the target point. Change the target point to avoid a particular elbow orientation.		
4272	Specified point flag is invalid.	For a CP motion command, the arm form at the target point is different from the point flag specified with the target point. Change the point flag for the target point.		
4273	J6Flag switched during the lift motion in coveyor tracking	Adjust the Tool orientation so that J6Flag will not switch		
4274	Manipulator motion did not match to J6Flag of the target point	For a CP motion command, the manipulator reached to the target point with J6Flag which differs from the one specified for the target point. Change J6Flag for the target point.		

No.	Message	Remedy	Note 1	Note 2
4275	Manipulator motion did not match to J4Flag of the target point	For a CP motion command, the manipulator reached to the target point with J4Flag which differs from the one specified for the target point. Change J4Flag for the target point.		
4276	Manipulator motion did not match to ArmFlag of the target point	For a CP motion command, the manipulator reached to the target point with ArmFlag which differs from the one specified for the target point. Change ArmFlag for the target point.		
4277	Manipulator motion did not match to ElbowFlag of the target point	For a CP motion command, the manipulator reached to the target point with ElbowFlag which differs from the one specified for the target point. Change ElbowFlag for the target point.		
4278	Manipulator motion did not match to WristFlag of the target point	For a CP motion command, the manipulator reached to the target point with WristFlag which differs from the one specified for the target point. Change WristFlag for the target point.		

Servo

No.	Message	Remedy	Note 1	Note 2
5000	Servo control gate array failure. Check the DMB.	Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency and I/O connectors) Replace the DMB. Replace the additional axis unit.		
5001	Disconnection of the parallel encoder signal. Check the signal cable connection or the robot internal wiring.	Check the M/C cable signal. Check the robot signal wiring. (Missing pin, disconnection, short-circuit) Replace the motor. Replace the DMB. Check the connector connection in the controller. (Loosening, connecting to the serial encoder terminal on the DMB) Check the model setting. Check the peripheral equipment wiring. (Emergency and I/O)		
5002	Motor driver is not installed. Install the motor driver. Check the DMB or the motor driver.	Check whether the motor driver is mounted. Check the model setting and hardware setting. Replace the motor driver. Replace the DMB.		
5003	Initialization communication failure of incremental encoder. Check the signal cable connection and the robot setting.	Check the model setting. Replace the motor. Replace the DMB.		
5004	Initialization failure of absolute encoder. Check the signal cable connection or the robot setting.	Check the model setting. Replace the motor. Replace the DMB.		
5005	Encoder division setting failure. Check the robot setting.	Check the model setting.		
5006	Data failure during absolute encoder initialization. Check the signal cable connection, the controller, or the motor.	Replace the motor. Replace the DMB. Check the noise countermeasures.		
5007	Absolute encoder multi-turn is beyond the maximum range. Reset the encoder.	Reset the encoder. Replace the motor.		
5008	Position is out of the range. Reset the encoder.	Reset the encoder. Replace the DMB. Replace the motor.		
5009	No response from the serial encoder. Check the signal cable connection, the motor, the DMB, or the encoder IF board.	Check the model setting. (Improperly setting of the parallel encoder model) Check the signal cable connection. Replace the DMB and encoder I/F board.		
5010	Serial encoder initialization failure. Reboot the controller. Check the motor, the DMB, or the encoder IF board.	Check the robot configuration. Check the signal cable connection. Replace the DMB and encoder I/F board.		

No.	Message	Remedy	Note 1	Note 2
5011	Serial encoder communication failure. Reboot the controller. Check the motor, the DMB, or the encoder IF board.	Check the robot configuration. Check the signal cable connection. Replace the DMB and encoder I/F board.		
5012	Servo CPU watchdog timer failure. Reboot the controller. Check the motor or the DMB.	Replace the DMB. Check the noise countermeasures.		
5013	Current control circuit WDT failure. Reboot the controller. Check the controller.	Check the power cable connection. Check the 15V power supply and cable connection. Replace the DMB. Check the noise countermeasures.		
5015	Encoder is reset. Reboot the controller.	Reboot the controller.		
5016	Power supply failure of the absolute encoder. Replace the battery. Check the robot internal wiring.	Reset the encoder. Check the signal cable connection.		
5017	Backup data failure of the absolute encoder. Reset the encoder.	Reset the encoder. Check the signal cable connection.		
5018	Absolute encoder battery alarm.	Replace the battery. Check the signal cable connection.		
5019	Position failure of the absolute encoder. Reset the encoder. Replace the motor.	Reset the encoder. Replace the motor.		
5020	Speed is too high at controller power ON. Stop the robot and reboot the controller.	Reboot the controller.		
5021	Absolute encoder overheat.	Lower the motion duty. Wait until the temperature of the encoder decreases.		
5022	R/D transducer failure. Check the resolver board.	Check the noise countermeasure. Replace the resolver board.		
5023	G sensor communication failure. Check the Gyro board.	Check the M/C signal cable. Check the robot signal wiring (for pin falling, disconnection, short). Check the noise countermeasure. Replace the Gyro board. Replace the DMB board.		
5024	G sensor data failure. Check the Gyro board.	Replace the Gyro board.		
5025	Resolver mixing failure. Reset the encoder.	Reset the resolver. Replace the resolver board.		
5026	Resolver signal disconnection. Check the motor and resolver board.	Check the robot signal wiring. Replace the resolver board.		
5027	S-DSP communication failure. Check-sum error, Free-run counter error	Reboot the controller. Replace the DMB. Check the noise countermeasure.		
5028	Current data failure. Data update stopped. Parity error.	Reboot the controller. Replace the DMB. Check the noise countermeasure.		
5029	D-DSP communication failure. Check-sum error, Free-run counter error	Reboot the controller. Replace the DMB. Check the noise countermeasure.		

No.	Message	Remedy	Note 1	Note 2
5032	Servo alarm A.			
5040	Motor torque output failure in high power state. Check the power cable connection, the robot, the driver or the motor.	Specify the Weight/Inertia setting. Check the load. Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		
5041	Motor torque output failure in low power state. Check the power cable connection, robot, brake, driver, or motor.	Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		
5042	Position error overflow in high power state. Check the power cable connection, the robot, the driver and the motor.	Specify the Weight/Inertia setting. Check the load. Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		

No.	Message	Remedy	Note 1	Note 2
5043	Position error overflow in low power state. Check the power cable connection, robot, brake, driver, or motor.	Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		
5044	Speed error overflow in high power state. Check the power cable connection, robot, brake, driver, or motor.	Specify the Weight/Inertia setting. Check the load. Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		
5045	Speed error overflow in low power state. Check the power cable connection, robot, brake, drive, or motor.	Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		

No.	Message	Remedy	Note 1	Note 2
5046	Over speed in high power state. Reduce SpeedS. Check the signal cable connection, robot, brake, driver or motor.	Reduce SpeedS of the CP motion. Change the orientation of the CP motion. Specify the Weight/Inertia setting. Check the load. Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		
5047	Over speed in low power state. Check the signal cable connection, robot, brake, driver, or motor.	Check the motion in high power state. Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		
5048	Over voltage of the main power circuit. Check the main power voltage or the regeneration module.	Specify the Weight/Inertia setting. Check the load. Check the robot. (Smoothness, backlash, non-smooth motion, loose belt tension, brake) Check the interference with the peripheral equipment. (Collision, contact) Check the model setting. Check the power cable connection. Check the robot power wiring. (Missing pin, disconnection, short-circuit) Check the power supply voltage. (Low power supply voltage) Replace the motor driver. Replace the DMB. Replace the motor.		

No.	Message	Remedy	Note 1	Note 2
5049	Over current of the motor driver. Check the power cable connection or the robot internal wiring.	Check the short-circuit and earth fault of the power line. Replace the motor driver. Replace the DMB.		
5050	Over speed during torque control. Check the work motion speed range.	Check the motion speed during torque control.		
5051	15V PWM drive power supply failure. Reboot the controller. Replace the 15V power supply.	Check the 15V power supply and cable connection. Replace the motor driver. Replace the DMB.		
5054	Overload of the motor. Decrease the motion duty and the Accel.	Lower the motion duty. Check the Weight/Inertia setting. Check the robot. (Backlash, large load, loose belt tension, brake)		
5055	Overload of the motor. Decrease the operation duty and the Accel.	Lower the motion duty. Check the Weight/Inertia setting. Check the robot. (Backlash, large load, loose belt tension, brake)		
5056	G sensor output failure. Check the Gyro board.	Check the noise countermeasures. Replace the Gyro board.		
5072	Servo alarm B.			
5080	Motor is overloaded. Decrease the duty and the Accel.	Lower the motion duty. Check the Weight/Inertia setting. Check the robot. (Backlash, large load, loose belt tension, brake)		
5098	High temperature of the encoder. Decrease the duty. Check the reduction gear unit of the robot.	Wait until the temperature of the encoder decreases. Lower the motion duty. Check the Weight/Inertia setting. Check the robot. (Backlash, large load, loose belt tension, brake)		
5099	High temperature of the motor driver . Clean the controller fan filter. Check the ambient temperature. Decrease the duty.	Clean the cooling fan filter. Lower the motion duty. Check the Weight/Inertia setting. Lower the ambient temperature.		
5112	Servo alarm C.			

Vision Calibration

No.	Message	Remedy	Note 1	Note 2
6001	Calibration number is out of range.			
6002	Calibration data is not defined.			
6003	Camera mounting direction is out of range.			
6004	2-point measurement flag is out of range.			
6005	There is an invalid data in the pose data.			
6006	Calibration failure: Invalid data prevents calculation.			
6007	Coordinate conversion: Invalid data prevent calculation.			
6009	Calibration file name is not correct.			
6010	Calibration file does not exist.			
6012	Failed to load the calibration file.			
6013	Failed to write into the calibration file.			
6014	Specify continuous 9 data for the Pixel coordinate.			
6015	Specify continuous 18 data for the Pixel coordinate.			
6016	Specify continuous 9 data for the Robot coordinate.			
6017	Specify continuous 18 data for the Robot coordinate.			
6018	Specify continuous 9 data and 1 reference point for the Robot coordinate.			
6019	Specify continuous 9 data and 2 reference points for the Robot coordinate.			

Points

No.	Message	Remedy	Note 1	Note 2
7003	The specified robot cannot be found.			
7004	Duplicate allocation of the point data area.			
7006	Specified point number cannot be found. Specify a valid point number.	Check the specified point number.		
7007	Specified point number was not defined. Specify a teach point number.	Check whether point data is registered in the specified point. Perform the teaching.		
7010	Cannot allocate the memory area for the pallet definition.			
7011	Cannot free the memory area for the pallet definition.			
7012	Specified pallet number cannot be found. Specify a valid pallet number.	Check the pallet number.		
7013	Specified pallet is not defined. Specify a defined pallet or define the pallet.	Check whether the specified pallet is defined by the Pallet statement. Declare the pallet.		
7014	Specified division number is beyond the pallet division number definition. Specify a valid division.	Check the specified division number.		
7015	Specified coordinate axis number does not exist.			
7016	Specified arm orientation number does not exist.			
7017	Cannot allocate the required memory.			
7018	Specified point label cannot be found. Specify a valid point label.	Check the specified point label.		
7019	Parameter setup in the initialization file is invalid.			
7021	Duplicate point label. Specified label name is already registered. Change the label name.	Change the point label.		
7022	Specified local coordinate system is not defined. Specify a valid local coordinate system number.	Check the specified local number. Define the Local coordinate system.		
7023	Specified string is not in the correct format.			
7024	Point data memory area for the specified robot is not allocated.			
7026	Cannot open the point file. Specify a valid point file name.	Check the point file name. Check whether the point file specified for the project exists.		
7027	Cannot read the point data from the point file.	Create the point file again.		
7028	Point area is allocated beyond the available point number.			

No.	Message	Remedy	Note 1	Note 2
7029	Specified point file name is not correct. Specify a valid point file name.	Check the file extension.		
7030	Specified point label is beyond the maximum length. Specify a valid point label.	Change the point label.		
7031	Description for the specified point is beyond the maximum length. Specify a valid description.	Change the comment.		
7032	Point file is corrupted. Check sum error.	Create the point file again.		
7033	Specified point file cannot be found. Specify a valid point file name.			
7034	Cannot save the point file.			
7035	Cannot save the point file.			
7036	Cannot save the point file.			
7037	Cannot save the point file.			
7038	Cannot save the point file.			
7039	Cannot save the point file.			
7040	The point label is not correct. Specify a valid point point label.			
7041	The point label is not correct. Specify a valid point point label.			

Fieldbus

No.	Message	Remedy	Note 1	Note 2
7101	Communication error occur during transform.	The module is broken or the controller software is damaged. Restore the controller firmware. (If Code 1 is 1, 2, 3, 4, or 10)		
		A communication data error was detected during communication. The communication cable has a problem. Check the communication cable and its related units. (If Code 1 is 11 or 12)		
		The module is broken or the controller software is damaged. Restore the controller firmware. (If Code 1 is 13, 14, or 15)		
7103	Timeout error occurs during transform.	The module is broken or the controller software is damaged. Restore the controller firmware. (If Code 1 is 1, 2, or 3)		
		A communication data error was detected during communication. The communication cable has a problem. Check the communication cable and its related units. (If Code 1 is 4)		

Vision

No.	Message	Remedy	Note 1	Note 2
7300	Vision Communication. Server mode not supported.			
7302	Vision Communication. Failed to read from the camera.	Check the connection with the camera.		
7303	Vision Communication. Read data overflow.			
7304	Vision Communication. Failed to open the Ethernet port.			
7305	Vision Communication. Invalid IP address of camera.	Rebuild the project. Check the camera configuration.		
7306	Vision Communication. No specification of Server/Client.			
7307	Vision Communication. Failed to send to the camera.	Check the connection with the camera.		
7308	Vision Communication. Camera version is old.			
7321	Vision Communication. Camera setting has not been set.	Rebuild the project. Check the camera configuration.		
7322	Vision Communication. Read timeout.			
7323	Vision Communication. Read invalid data.	Check the connection with the camera.		
7324	Vision Communication. Failed to send to the camera.	Check the connection with the camera.		
7325	Vision Communication. Connection is not completed.	Check the connection with the camera.		
7326	Vision Communication. Read data is too long.			
7327	Vision Communication. Undefined vision sequence.			
7328	Vision Communication. Camera setting has not been set.	Rebuild the project. Check the camera configuration.		
7329	Vision Communication. Vis file is not found.	Rebuild the project. Check the camera configuration.		
7330	Vision Communication. Failed to allocate memory.			
7341	Vision Communication. Out of max camera number.			
7342	Vision Communication. Invalid camera number.			
7343	Vision Communication. VSet parameter is too long.			
7344	Vision Communication: Too many parameters for VGet.			
7345	Vision Communication. Not enough data for VGet statement variable assignment.			
7346	Vision Communication. Cannot execute a Vision statement from the command window.			
7500	Smart camera. Out of memory.			
7501	Smart camera. Project does not exist.			

No.	Message	Remedy	Note 1	Note 2
7502	Smart camera. Project has not been set.			
7503	Smart camera. Vision property or result not supported.			
7504	Smart camera. Cannot open project file.			
7505	Undefined vision sequence.			
7506	Undefined vision object.			
7507	Smart camera. Critical error.			
7508	Smart camera. Invalid command.			
7509	Invalid vision property value.			
7510	Invalid vision property.			
7511	Vision model not trained.			
7512	Undefined vision calibration.			
7513	Vision model object not Self.			
7514	Invalid vision result.			
7515	Vision object not found.			
7516	No vision calibration.			
7517	Incomplete vision calibration.			
7518	Smart camera. Cannot connect with camera.			
7819	Smart camera. Communication error.			

GUI Builder

No.	Message	Remedy	Note 1	Note 2
7600	Cannot execute a GUI Builder statement from the command window.	GUI Builder commands are only available in the program.		
7602	GSet parameter is too long.	-		
7603	Too many parameters for GGet.	-		
7604	Not enough data for GGet statement variable assignment.	-		
7610	The event task cannot be executed. System in pause state and EventTaskType is Normal.	Specify NoEmgAbort for EventTaskType.		
7611	The event task cannot be executed. Safeguard is open and EventTaskType is Normal.	Specify NoEmgAbort for EventTaskType.		
7612	The event task cannot be executed. Estop is active and EventTaskType is not NoEmgAbort.	Specify NoEmgAbort for EventTaskType.		
7613	The event task cannot be executed. System in error state and EventTaskType is not NoEmgAbort.	Specify NoEmgAbort for EventTaskType.		
7650	Invalid property.	-		
7651	Invalid form.	-		
7652	Invalid control.	-		
7653	The specified form is already open.	-		
7654	Event function does not exist.	Create the event function.		

Hardware

No.	Message	Remedy	Note 1	Note 2
9001	Emergency stop circuit failure was detected. Disconnection or other failure was found in one of the redundant inputs.	Check whether no disconnection, earth fault, or short-circuit of the emergency stop input signal exits. Then reboot the controller.		
9002	Safeguard circuit failure was detected. Disconnection or other failure was found in one of the redundant inputs.	Check whether no disconnection, earth fault, or short-circuit of the safeguard input signal exits. Then reboot the controller.		
9011	Battery voltage of the CPU board backup is lower than the specified voltage. Replace the CPU board battery.			
9012	5V input voltage for CPU board is lower than the specified voltage.			
9013	24 V input voltage for the motor brake, encoder and fan is lower than the specified voltage.			
9014	Internal temperature of the Controller is higher than the specified temperature.	Stop the controller as soon as possible and check whether the ambient temperature of the controller is not high. Check whether the filter is not clogged up.	Current value	Boundary value
9015	Rotating speed of the controller fan is below the allowed speed. (FAN1)	Check whether the filter is not clogged up. If the warning is not cleared after the controller is rebooted, replace the fan.	Current value	Boundary value
9016	Rotating speed of the controller fan is below the allowed speed. (FAN2)	Check whether the filter is not clogged up. If the warning is not cleared after the controller is rebooted, replace the fan.	Current value	Boundary value
9017	Internal temperature of the Controller is higher than the specified temperature.			
9100	Initialization failure. Failed to allocate memory.	Reboot the controller.		
9101	Message queue has become full.			
9233	The Fieldbus I/O driver is in an abnormal state.	The module is broken or the controller software is damaged. Restore the controller firmware.		
9234	Fieldbus I/O driver initialization failure.	The module is broken or the controller software is damaged. Restore the controller firmware.		
9610	RAS circuit detected a servo system malfunction. Reboot the controller. Check for noise. Replace the controller.	Check the noise countermeasures. Replace the DMB.		
9611	Servo CPU internal RAM failure. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9612	RAM for the main and servo CPU communication failure. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9613	Servo CPU internal RAM failure. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9614	Initialization communication of main CPU and servo CPU failure. Reboot the Controller. Check for noise. Replace DMB.	Check the noise countermeasures. Replace the DMB.		

No.	Message	Remedy	Note 1	Note 2
9615	Initialization communication of the main and servo CPU failure. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9616	Communication of the main and servo CPU failure. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9617	Communication of the main and servo CPU failure. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9618	Servo long time command overrun.	Check the noise countermeasures. Replace the DMB.		
9619	Servo long time command check sum error.	Check the noise countermeasures. Replace the DMB.		
9620	System watchdog timer detected a failure. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9621	Drive unit check failure.	Check the noise countermeasures. Replace the DMB.		
9622	RAM failure of the servo CPU. Reboot the controller. Check for noise. Replace the DMB.	Check the noise countermeasures. Replace the DMB.		
9623	Failure of the redundant circuitry for the emergency stop or the safeguard. Check the wiring.	Check the noise countermeasures. Replace the DMB.		
9624	Low voltage of the main circuit power supply was detected. Check the power supply voltage. Reboot the controller.	Check the noise countermeasures. Replace the DMB.		
9625	Control relay contact of the main circuit power supply is welded closed. Replace the DPB.	Replace the DMB.		
9630	Servo real time status failure. Check sum error.	Reboot the controller. Replace the DMB. Check the noise countermeasures.		
9632	Servo real time status failure. Servo free running counter error	Reboot the controller. Replace the DMB. Check the noise countermeasures.		
9633	Servo real time status failure. Servo CPU communication error.	Reboot the controller. Replace the DMB. Check the noise countermeasures.		
9640	Irregular motion control interruption was detected. Interruption duplicate.	Reboot the controller. Replace the DMB. Check the noise countermeasures.		
9700	Servo control gate array failure. Check the DMB.	Check the short-circuit and improper connection of the peripheral equipment wiring. (Emergency and I/O connectors) Replace the DMB. Replace the additional axis unit.		

No.	Message	Remedy	Note 1	Note 2
9701	Disconnection of the parallel encoder signal. Check the signal cable connection or the robot internal wiring.	Check the M/C cable signal. Check the robot signal wiring. (Missing pin, disconnection, short-circuit) Replace the motor. (Encoder failure) Replace the DMB. (Detection circuit failure) Check the connector connection in the controller. (Loosening, connecting to the serial encoder terminal on the DMB) Check the model setting. (Improperly setting of the parallel encoder) Check the peripheral equipment wiring. (Emergency and I/O)		
9702	Motor driver is not installed. Install the motor driver. Check the DMB or the motor driver.	Check whether the motor driver is mounted. Check the model setting and hardware setting. Replace the motor driver. Replace the DMB.		
9703	Initialization communication failure of incremental encoder. Check the signal cable connection and the robot setting.	Check the model setting. Replace the motor. (Encoder failure) Replace the DMB.		
9704	Initialization failure of absolute encoder. Check the signal cable connection or the robot setting.	Check the model setting. Replace the motor. (Encoder failure) Replace the DMB.		
9705	Encoder division setting failure. Check the robot setting.	Check the model setting.		
9706	Data failure at the absolute encoder initialization. Check the signal cable connection, the controller, or the motor.	Replace the motor. (Encoder failure) Replace the DMB. Check the noise countermeasures.		
9707	Absolute encoder multi-turn is beyond the maximum range. Reset the encoder.	Reset the encoder. Replace the motor. (Encoder failure)		
9708	Position is out of the range. Reset the encoder.	Reset the encoder. Replace the DMB. Replace the motor. (Encoder failure)		
9709	No response from the serial encoder. Check the signal cable connection, the motor, the DMB, or the encoder IF board.	Check the model setting. (Improperly setting of the parallel encoder model) Check the signal cable connection. Replace the DMB and encoder I/F board.		
9710	Serial encoder initialization failure. Reboot the controller. Check the motor, the DMB, or the encoder IF board.	Check the robot configuration. Check the signal cable. Replace the DMB and encoder I/F board.		
9711	Serial encoder communication failure. Reboot the controller. Check the motor, the DMB, or the encoder IF board.	Check the robot configuration. Check the signal cable. Replace the DMB and encoder I/F board.		
9712	Servo CPU watchdog timer failure. Reboot the controller. Check the motor or the DMB.	Replace the DMB. Check the noise countermeasures.		
9713	Current control circuit WDT failure. Reboot the controller. Check the controller.	Check the power cable connection. Check the 15V power supply and cable connection. Replace the DMB. Check the noise countermeasures.		
9715	Encoder is reset. Reboot the controller.	Reboot the controller.		
9716	Power supply failure of the absolute encoder. Replace the battery to a new one. Check the robot internal wiring.	Reset the encoder. Check the signal cable connection.		

No.	Message	Remedy	Note 1	Note 2
9717	Backup data failure of the absolute encoder. Reset the encoder.	Reset the encoder. Check the signal cable connection.		
9718	Absolute encoder battery alarm.	Replace the battery. Check the signal cable connection.		
9719	Position failure of the absolute encoder. Reset the encoder. Replace the motor.	Reset the encoder. Replace the motor. (Encoder failure)		
9720	Speed is too high at controller power ON. Stop the robot and reboot the controller.	Reboot the controller.		
9721	Absolute encoder over heat.	Lower the motion duty. Wait until the temperature of the encoder decreases.		
9722	R/D transducer failure. Check the resolver board.	Check the noise countermeasure. Replace the resolver board.		
9723	G sensor communication failure. Check the Gyro board.	Check the M/C signal cable. Check the robot signal wiring (for pin falling, disconnection, short). Check the noise countermeasure. Replace the Gyro board. Replace the DMB board.		
9724	G sensor data failure. Check the Gyro board.	Replace the Gyro board.		
9725	Resolver mixing failure. Reset the encoder.	Reset the resolver. Replace the resolver board.		
9726	Resolver signal disconnection. Check the motor and resolver board.	Check the robot signal wiring. Replace the resolver board.		
9727	S-DSP communication failure. Check-sum error, Free-run counter error	Reboot the controller. Replace the DMB. Check the noise countermeasure.		
9728	Current data failure. Data update stopped. Parity error.	Reboot the controller. Replace the DMB. Check the noise countermeasure.		
9729	D-DSP communication failure. Check-sum error, Free-run counter error	Reboot the controller. Replace the DMB. Check the noise countermeasure.		
9732	Servo alarm A.			

EPSON RC+

No.	Message	Remedy	Note 1	Note 2
10000	Command aborted by user			
10001	Command timeout.			
10002	Bad point file line syntax			
10003	Project could not be built.			
10004	Cannot initialize Spel class instance.			
10005	Cannot initialize parser.			
10006	Cannot initialize wbproxy.			
10007	Project does not exist.			
10008	No project specified.			
10009	Cannot open file.			
10010	Cannot create file.			
10011	File not found			
10012	Option not enabled			
10013	Cannot execute LoadPoints with Robot Manager open.			
10014	Project cannot be locked. It is being used by another session.			
10015	Project could not be synchronized.			
10016	Drive not ready			
10017	Invalid IP address			
10018	Invalid IP mask			
10019	Invalid IP gateway			
10020	IP address or gateway cannot be the subnet address			
10021	IP address or gateway cannot be the broadcast address			
10022	Invalid DNS address			

Precaution of EPSON RC+ Ver.4.* Compatibility

Overview

This section contains information for customers using EPSON RC+ 5.0 with RC170/RC180 Controller that have already used EPSON RC+ Ver.4.* with RC520 or RC420.

EPSON RC+ 5.0 and EPSON RC+ Ver.4.* differs in such as harware, adaptable manipulators, number of joint allowance, and software execution enviornment. Please read this section and understand the contents for the safety use of the Robot system.

EPSON RC+ 5.0 is an improved software that has compatibility with products before EPSON RC+ 5.0 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+ 5.0.

General Differences

General differences of EPSON RC+ Ver.4.* and EPSON RC+ 5.0 are as follows.

Item	EPSON RC+ 5.0	EPSON RC+ Ver.4.*
Number of 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 NoPouse task
Special TRAP such as TRAP ERROR	Not supported	Supported
Task starts by TRAP number	Dedicated task number	Task number only using 1 to 32
Number of significant figure for Real type	6 digits	No specification
Number of significant figure for Double type	14 digits	No specification

Array elements number	Other than string variable
	  Local variable 1000
	  Global variable 10000
	  Module variable 10000
	  Global Preserve variable 1000
	
	String variable
	  Local variable 100
	  Global variable 1000
	  Module variable 1000
	  Global Preserve variable 100
		As far as the memory remains

Line number	Not supported	Supported
Device number	21:PC 22:REMOTE 23:OP	1:Controller 2:REMOTE 3:OP
Timer number range	0 to 15	0 to 63
Signal No range for SyncLock, SyncUnlock	0 to 15	1 to 32
Signal No range for WaitSig, Signal	0 to 5	0 to 127
Port No of Ethernet	201 to 208	128 to 147
Port No of RS-232C communication	1 to 8	1 to 16
OpenCom execution of RS-232C communication port	Mandatory	Optional
Input/output to files	Not supported	Supported
Fieldbus I/O	Use normal I/O commands	Use special commands
Group in the project	Not supported	Supported
Error number	New error numbers	

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	−	
	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	−	

	Command	Compatibility	Note
C	Calib Statement	×	
	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	!	Remaned to Cls Device ID can be spscified for argumants
	Cnv_**	×	
	Cont	!	Can be executed only from the event handler of GUI Builder
	Cont Statement	×	
	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	−	
	CX to CW Function	−	
D	Date Statement	−	
	Date$ Function	−	
	Declare Statement	×	
	DegToRad Function	−	
	Del Statement	×	
	Dir Statement	×	
	Dist Function	−	
	Do...Loop Statement	−	
	Double Statement	!	Significant figure is 14 digits
E	EClr Statement	×	
	ECP Statement	−	

	Command	Compatibility	Note
	ECP Function	–	
	ECPClr Statement	–	
	ECPSet Statement	–	
	ECPSet Function	–	
	Elbow Statement	–	
	Elbow Function	–	
	ENetIO_****	×	
	Eof Function	×	
	EPrint Statement	×	
	Era Function	+	Able to ommit the task number
	Erase Statement	×	
	EResume Statement	–	
	Erf$ Function	+	Able to ommit the task number
	Erl Function	+	Able to ommit 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	×	
	Exit Statement	–	
F	FbusIO_****	×	Normal I/O command avaiable
	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$ Function	×	
	Global Statement	–	
	Go Statement	–	
	Gosub...Return	–	
	Goto Statement	–	
H	Halt Statement	–	
	Hand Statement	–	
	Hand Function	–	
	Here Statement	–	
	Here Function	–	

	Command	Compatibility	Note
	Hex$ Function	−	
	Hofs Statement	×	
	Hofs Function	−	
	Home Statement	−	
	HomeSet Statement	−	
	HomeSet Function	−	
	HOrdr Statement	−	
	HOrdr Function	−	
	Hour Statement	−	
	Hour Function	−	
	HTest Statement	×	
	HTest Function	×	
I	If...EndIf	−	
	ImportPoints Statement	×	
	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	−	
	Jump3 Statement	−	
	Jump3CP Statement	−	
K	Kill Statement	×	
L	LCase$ Function	−	
	Left$ Function	−	
	Len Function	−	

	Command	Compatibility	Note
	LimZ Statement	–	
	LimZ Function	–	
	Line Input Statement	–	
	Line Input# Statement	+	Input is available from devices
	LoadPoints	!	Extension (.pnt) has changed to (.pts)
	Local Statement	!	Local number "0" is an error
	Local Function	!	Local number "0" is an error
	LocalClr Statement	–	
	Lof Function	–	
	LogIn Statement	×	
	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	–	

	Command	Compatibility	Note
	OP_*	×	
	OpBCD Statement	−	
	OpenCom Statement	!	OpenCom is mandatory
	OpenNet Statement	−	
	Oport Function	−	
	Or Operator	−	
	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	!	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

	Command	Compatibility	Note
	PTPTime Function	–	
	PTran Statement	–	
	PTRQ Statement	–	
	PTRQ Function	–	
	Pulse Statement	–	
	Pulse Function	–	
Q	QP Statement	–	
	Quit Statement	–	
R	RadToDeg Function	–	
	Randmize Statement	+	Seed value can be specified
	Range Statement	–	
	Read Statement	–	
	ReadBin Statement	+	Able to read mauliple bytes to array variable
	Real Statement	!	6 digit significant figure
	Recover Statement	×	
	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	–	
	Return Statement	–	
	Right$ Function	–	
	RmDir Statement	×	
	Rnd Function	–	
	Robot Statement	×	
	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	–	

	Command	Compatibility	Note
	Shutdown Statement	×	
	Signal Statement	–	
	Sin Function	–	
	SLock Statement	–	
	Space$ Function	–	
	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 retrival
	Str$ Function	–	
	String Statement	–	
	Sw Function	–	
	Sw($) Function	×	Replaced to MemSw
	SyncLock Statement	!	Error occurs by executing SyncLock repeatedly
	SyncUnlock Statement	–	
T	Tab$ 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	–	
	TillOn Function	–	
	Time Command	–	
	Time Function	–	
	Time$ Function	–	
	TLClr Statement	–	
	TLSet Statement	–	
	TLSet Function	–	
	TMOut Statement	–	
	TMove Statement	–	
	Tmr Function	–	
	TmReset Statement	–	
	Tool Statement	–	
	Tool Function	–	

	Command	Compatibility	Note
	Trap Statement	!!	Compatibility with Trap Goto Trap Gosub abolished and replaced to Trap Call Trap Call is renamed to Trap Xqt Trap Emergency, Trap Error, Trap Abort, Trap Pause, Trap SGOpen, Trap SGClose have been deleted
	Trim$ Function	−	
	Tw Function	!	Returns True/False
	Type Statement	×	
U	UBound Function	−	
	UCase$ Function	−	
	UOpen Statement	×	
V	Val Function	−	
	Ver Statement	×	Replaced to SysConfig
	Verinit Statement	×	
W	Wait Statement	−	
	WaitNet Statement	−	
	WaitPos Statement	−	
	WaitSig Statement	−	
	Weight Statement	−	
	Weight Function	−	
	Where Statement	!	Coordinate value always diplays 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	−	

List of New Commands

ArmDef Function	Joint	SetInReal
AccelMax Function	LJM Function	SysErr Function
Align Function	LocalDef Function	SoftCP
AlignECP Function	OLAccel	SoftCP Function
AtHome Function	OLAccel Function	TaskInfo$ Function
Base Function	OutReal	TaskInfo Function
Box	OutReal Function	TaskState
Box Function	Plane	TC
BoxClr	Plane Function	TCLim
BoxDef Function	PlaneClr	TCLim Function
Brake Function	PlaneDef Function	TCSpeed
Cls	QPDecelR Statement	TCSpeed Function
DispDev	QPDecelR Function	TlDef Function
DispDev Function	QPDecelS Statement	Toff
EcpDef Function	QPDecelS Function	Ton
ErrorOn Function	RealPls Function	VxCalib statement
HomeClr	RealPos Function	VxCalDelete Statement
HomeDef Function	RobotInfo$ Function	VxCalLoad Statement
InReal	RobotInfo Function	VxCalInfo Function
InsideBox Function	RobotName$ Function	VxCalSave Statement
InsidePlane Function	RobotSerial$ Function	VxTrans Function
IOLabel$ Function		XYLimClr
		XYLimDef Function