North American Distributor

Software Instructions

PSM-127 Rev. 20
2/27/2024
Author: Patrick Moreland
Modified By: Patrick Moreland
Approved By: Patrick Moreland

Table of Contents

KX-2 Robot – Software Instructions

1.0 Introduction

1.1 Reference Guide: This document contains specifications and instructions for configuring the control software for the Peak Robotics KX-2 four-axis cylindrical robot.

1.2 User's Manual: Please refer to the KX-2 User's Manual, Controlled Document PSM-126 for detailed information on installation, setup, teaching, preventive maintenance, etc.

1.3 Product Changes: It is the general policy of Peak Robotics to improve products constantly. We reserve the right to make changes to specifications at any time and without necessarily notifying existing customers.

2.0 Product Description

2.1 Description: The KX-2 robot is a tabletop cylindrical robot designed for laboratory and industrial applications.

2.2 Technology: The robot has four axes of motion and incorporates zero-backlash harmonic drive gearboxes & timing belts driven by DC servomotors with true absolute encoders. The vertical Z axis contains a counterweight to minimize required thrust, and to eliminate the need for a Z brake.

2.3 Controller: All robot joints and gripper are servo-driven with motor drives integrated into the robot, mounted next to each motor. These independent drives communicate with the controlling PC via an internal CAN/USB adapter.

2.4 User Inputs/Outputs: There are three digital inputs, two analog inputs, and four digital outputs available at the AUX connector on the robot base. These are described in more detail in the User's Manual.

2.5 User-Supplied PC & Network: A standard PC with Windows 7 or newer is required to communicate with and direct the robot. The robot connects to the PC via two USB 2.0 ports (one for robot control and another for the barcode reader).

2.6 Teach Pendant: For debugging and robot teaching purposes, a graphical user interface called "KX-2 Teach Pendant" is provided as part of the DLL.

2.7 Universal DLL: The robot control software is an x86 .NET 4.6.1 DLL (Dynamic Link Library) created in Visual Studio 2013. The end-user is responsible for supplying a host application for writing system sequences. The host application can be written using any software development tool capable of interfacing with a .NET DLL, such as VB, C++, C#, Java, etc. The DLL can also be accessed as a COM object by using the .TLB file that is installed in the same directory as the DLL.

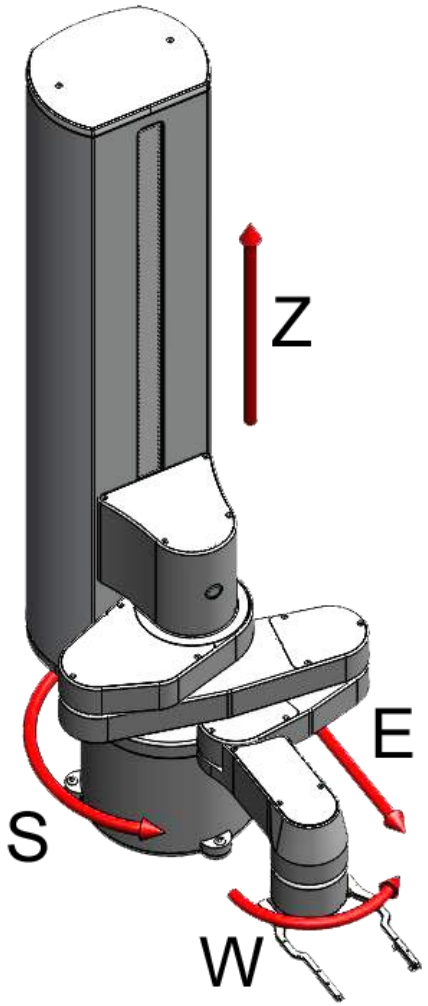
2.8 Scheduling Software: As an option, Overlord, a laboratory scheduling application, is available for developing system sequences. Please visit www.paa-automation.com for more information

2.9 Power: The robot uses 57 VDC for motor power, and 24VDC for logic power. The two DC power supplies are mounted inside the power supply box.

3.0 Description of Axes

3.1 Axis Naming Definitions:

Name	Designation	Description
Axis 1	“S”	Shoulder Motor, Shoulder Rotate
Axis 2	“Z”	Z Motor, Vertical Axis
Axis 3	“E”	Elbow Motor, Arm Extend
Axis 4	“W”	Wrist Motor, Wrist Rotate



KX-2 Axis Naming Definitions

3.2 Axis 1

3.2.1 The Shoulder is a rotary joint at the base of the robot. Axis 1 is used to rotate the entire robot about a vertical axis. Zero position is with the arm facing away from the side of the shoulder where the cables exit.

3.3 Axis 2

3.3.1 The Z axis is a belt-driven actuator mounted to the output of Axis 1. Axis 2 is used to move the arm up and down. Zero position is at the lower end of travel. Upward motion is in the positive direction. Two Z axis sizes are available: 500mm and 750mm travel. Axis 2 is equipped with a counterweight to prevent the Z axis from falling when power is removed.

3.4 Axis 3

3.4.1 The Elbow motor is a rotary joint that extends and retracts three arm links radially via a series of timing belts. Zero position is at the fully retracted position of the arm links.

3.5 Axis 4

3.5.1 The Wrist motor is a rotary joint that spins the gripper out at the end of the arm. Zero position is with the gripper facing straight forward away from the arm.

3.6 Rail

3.6.1 The robot can be mounted on a rail for larger systems. The rail is treated as an additional axis by the software (axis 5). When a rail is present, teachpoints will include the position of the rail, and its motion will be synchronized with the rest of the robot axes.

3.7 Gripper

3.7.1 The robot is equipped with an electric gripper that can handle portrait and landscape microplates. Top-grip and side-grip fingers are available, depending on the application. The gripper is numbered as axis 6 (was axis 5 in early robots released before the rail), although its position is not included in teachpoints.

3.7.2 A retro-reflective optical sensor is mounted on the gripper and is used for counting plates in a stack.

3.7.3 A 2D/3D camera-based barcode reader is located on the back of the gripper housing.

4.0 Description of Software

4.1 The file “KX2RobotControl.dll” is a Control program used for communicating with the robot.

4.2 The file “KX2RobotControl.dll” cannot run by itself. It must be referenced by a customer-supplied host application.

4.3 Any software language capable of interfacing with a .NET DLL, such as Visual Basic, C++, C#, Java, or Labview can be used to create a host application for referencing the methods and events of “KX2RobotControl.dll”, and should be used for writing complex robot motion sequences.

4.4 The following third-party host applications are available for controlling the KX-2 robot, and can simplify interfacing the robot to other devices.

- 4.4.1 Overlord, by Peak Analysis & Automation, Ltd. (www.paa-automation.com)
- 4.4.2 Agilent V-Works (www.agilent.com)
- 4.4.3 Retisoft Nexus (www.retisoft.ca)
- 4.4.4 Hamilton Company (www.hamiltoncompany.com)

4.5 A Teach Pendant executable is installed along with the DLL and can be opened from the Windows Start menu.

5.0 Hardware Requirements

5.1 A PC with one available USB 2.0 port is required for controlling the KX-2 robot.

5.2 A second USB port is required for the Serial/USB adapter provided for communicating with the barcode reader in the gripper.

5.3 The PC must be equipped with Windows 7 or newer. A minimum of a 2GHz dual-core processor and 4GB RAM is recommended. A higher-performance PC will improve software execution and will decrease time delays between moves.

6.0 Software Installation

6.1 Uninstall Previous Version

6.1.1 If a previous version of "KX-2 Robot Control" has been installed, backup the Teachpoints.ini and Sequences.ini files, and uninstall the DLL before installing the newer version (use “Start/Settings/Control Panel/Add/Remove Programs” to uninstall).

6.2 Install New Version

6.2.1 Run "KX2 Robot Control DLL vX.XX.X Setup.msi" to install the KX-2 robot software.

6.2.2 The PC must have Microsoft .NET 4.0 Framework installed (free download from Microsoft).

6.2.3 The following files will be installed in "C:\Program Files (x86)\Peak Robotics, Inc\KX2 Robot Control DLL":

KX2RobotControl.dll – This file contains the API for controlling the robot

PCANBasic.dll – This file is used by KX2RobotControl.dll to communicate with the CAN/USB adapter inside the robot

KX2RobotControl.tlb – This file can be used by older applications that require a COM interface to the DLL

KX2TeachPendent.exe – This executable proves a graphical user interface for teaching the robot. A shortcut to this file will be created in the “Start/All Programs/PAA” menu

6.2.4 The following text files are used by the DLL, and will be installed in the “C:\ProgramData\Peak Robotics, Inc\KX2 Robot Control DLL” directory:

TeachPoints.ini—Used for storing teach points.

Sequences.ini—Used for storing sequences created in the Sequence Editor.

ErrorLog.txt—Stores a history of all errors. The methods of “KX2RobotControl.dll” should now be available to the user’s control code application.

6.3 Adding a Reference

6.3.1 To add a reference to “KX2RobotControl.dll” in Visual Basic 2013:

6.3.1.1 Launch Visual Studio 2013.
6.3.1.2 Select File/New/Project…
6.3.1.3 Select .NET Framework 4

6.3.1.4 Expand “Templates” and select “Visual Basic”.
6.3.1.5 Select “Windows Forms Application”.
6.3.1.6 Give the application a name and click “Ok”.
6.3.1.7 Select “Project/MyApplication Properties…”. Select “Compile”, and select “x86” for “Target CPU:”
6.3.1.8 Place a copy of KX2RobotControl.dll inside the main project folder.
6.3.1.9 Place copies of PCANBasic.dll inside the bin/Debug and bin/Release folders.
6.3.1.10 Select “References”, then click “Add…”
6.3.1.11 In the “Reference Manager” window, click on the “Browse…” item, and then click the “Browse…” button.
6.3.1.12 Browse to KX2RobotControl.dll inside the main project folder and click “Add”. Click “OK”.
6.3.1.13 Right-click Form1.vb in the Solution Explorer and select “View Code”.
6.3.1.14 Add the following declaration inside Public Class Form1:
Dim WithEvents KX2 as KX2RobotControlNamespace.KX2RobotControl
6.3.1.15 Select “(Form1 Events)” from the “Class Name” drop-down list and select “Load” from the “Method Name” drop-down list. Add the following line inside the Form1_Load subroutine:
“KX2 = New KX2RobotControlNamespace.KX2RobotControl”
6.3.1.16 To call methods, use the following format:
“ReturnValue = KX2.MethodName(Parameter1, Parameter2,…)”
6.3.1.17 To startup the robot, call the Initialize() method. To turn off the robot, call the ShutDown() method. Be sure to call ShutDown() and PreClassTerminateCleanup() before closing your application to ensure the application will be able to connect to the robot the next time it is opened.

6.3.2 Create the KX2RobotControl object only once and use this same instance for the life of your application. The DLL must maintain a connection with the robot, and this connection will be lost if the object gets recreated.

6.3.3 Multiple instances of the KX2RobotControl object can be created for controlling multiple robots from one application. The SetRobotSerialNumber method must be used directly after creating each object to keep separate registry settings for each robot. See the KX-2 User’s Manual for information on assigning a unique hardware ID to each robot.

6.4 Accessing Events

6.4.1 To access events:

6.4.1.1 Select “KX2” in the “Class Name” drop-down menu.
6.4.1.2 Select the desired event in the “Method Name” drop-down menu.
6.4.1.3 The selected event subroutine code will then be added to the code text editor window.
6.4.1.4 The desired operation can then be added to the event subroutine.

6.5 User Forms

6.5.1 The DLL includes the following user forms:

6.5.1.1 TP_TeachPointsUsrCtrl
6.5.1.2 TP_JogControlsUsrCtrl
6.5.1.3 TP_InputsOutputsUsrCtrl
6.5.1.4 TP_MotionRulesUsrCtrl
6.5.1.5 TP_MenuStripUsrCtrl

6.5.2 These user forms represent the various components of the Teach Pendant, which can be inserted into the host application.

6.5.3 To insert a user control into a form, open the form Toolbox, right-click the Toolbox, and select “Choose items…”. Click the “Browse…” button and find KX2RobotControl.dll. The user controls will now be available in the Toolbox and can be selected and placed on the form as needed. The KX2ObjectSet method must be called for each user form, passing in the same KX2 object that was created in Form1_Load.

6.6 Registry Keys

6.6.1 The following values can be found in the Registry under “My Computer\HKEY_CURRENT_USER\Software\VB and VBA Program Settings\KX2 Robot Control\”:

6.6.1.1 DriveParamCache\—contains values stored by Initialize() for use by UseCachedDriveParams=True option.

6.6.1.2 Properties\SendEmailOnError—specifies whether emails containing error message information will be sent when the robot encounters an error (“False” or “True”).

6.6.1.3 Properties\ErrorEmailAddress—specifies the address to which error messages will be emailed

6.6.1.4 Properties\SMTPAcctName, SMTPPassword, SMTPPort, SMTPRequiresAuthentication, SMTPSecurePassword, and SMTPServer are settings used for configuring the email account used when SendEmailOnError = True.

6.6.1.5 Properties\LogCANData—If set to 1, a file will be created that contains a log of all communication data between the PC and robot. Set to 0 to disable logging.

6.6.1.6 Properties\LogCANDataNodeID—If set to 0, all communication will be logged. If set to a specific NodeID, only communication for that axis will be logged.

6.6.1.7 Properties\LogMovePathQueueErrors—If set to 1, a file will be created that contains a log of data points associated with each linear move that encounters a queue error. Set to 0 to disable logging. Enabling this feature will increase the likelihood of a queue error, so use only for debugging.

6.6.1.8 Properties\MasterVelocityScale—this value is used for scaling down the velocities of all move commands sent to the robot.

6.6.1.9 Properties\WriteToDebugTraceFile—If set to 1, a file will be created that contains a log of the functions that get called by the DLL.

6.6.1.10 Files\CANDataFile—specifies the file path used when LogCANData = 1. Specify a file path and a file name without a file type. A time stamp and .csv file type will be added to the file name.

6.6.1.11 Files\MovePathQueueErrorDataFile—specifies the file path used when LogMovePathQueueErrors = 1. Specify a file path and a file name without a file type. A time stamp and .csv file type will be added to the file name.

6.6.1.12 Files\DebugTraceFile—specifies the file path for the debug trace file.

6.6.1.13 Files\ErrorLogFile—specifies the file path for the ErrorLog.txt file where all robot errors are stored.

6.6.1.14 Files\SequenceFile—specifies the file path for the robot sequence file.

6.6.1.15 Files\TeachPointFile—specifies the file path for the robot teach point file.

6.6.1.16 Files\VisionCalibrationFile—specifies the path for the file that contains AprilTag calibration data used by the integrated vision system.

6.6.1.17 Joystick\JoyNum—Specifies which joystick to use if multiple joysticks are installed.

6.6.1.18 Joystick \Range0-15—These values store the range settings for analog controls on the jog-controlling joystick.

6.6.1.19 Joystick\Setting0-15—These values store the functions that are mapped to the joystick controls.

6.6.1.20 Joystick\Value0-15—These values store the analog control output levels that are required to activate the corresponding joystick function.

6.6.1.21 Joystick/ReverseJointJogElbow, ReverseJointJogShoulder, ReverseJointJogWrist, and ReverseJointJogZ reverse the directions that the axes rotate when in Joint Jog mode. Reversing the shoulder will have the benefit of causing the shoulder to move in the same direction in both Joint Jog and Cartesian Jog.

6.6.1.22 Settings\AutomaticMoveErrorRecovery—if set to “True”, then Automatic Move Error Recovery is enabled (“False” = disabled).

6.6.1.23 Settings\AutomaticMoveErrorRecoveryAttempts—specifies the number of times an automatic recovery will be attempted after a move error.

6.6.1.24 Settings\AutomaticMoveErrorRecoveryUseButton—specifies whether automatic move error recover will wait for the Teach Button to be pressed before recovering.

6.6.1.25 Settings\BarcodeReader – Specifies whether the barcode reader is enabled (1 = enabled, 0 = disabled).

6.6.1.26 Settings\BarcodeReaderBAUDRate – Specifies the BAUD rate of the barcode reader (default = 9600). Must match the internal setting in the barcode reader.

6.6.1.27 Settings\BarcodeReaderPortNum – Specifies the serial RS232 COM port number used by the barcode reader (default = 1).

6.6.1.28 Settings\CANDevice – Specifies the hardware ID of the CAN/USB adapter inside the robot (default = PCAN_USB 1 (51h)).

6.6.1.29 Settings\DefaultGripperState—specifies whether the gripper will be left closed (0) or open (1) after Initialize or ShutDown functions are called.

6.6.1.30 Settings\DefaultSequenceAcceleration, DefaultSequenceOperation, DefaultSequenceTeachPoint, and DefaultSequenceVelocity—specify the default values used when creating a sequence using the Sequence Editor.

6.6.1.31 Settings\EthernetPort – Specifies the port number to be used if the Interface Type is set to Ethernet.

6.6.1.32 Settings\InterfaceType – Specifies wither the interface is “USB” or “Ethernet”.

6.6.1.33 Settings\JogModeJoint, IncrementalJog, JogVelocity, and JointJogStep—stores the current settings in the Teach Pendant Jog window.

6.6.1.34 Settings\JogVelocityScaleShoulder, Z, Elbow, Wrist, Rail, Gripper – specify scaling factors used by the Teach Pendant jog controls (default = 100).

6.6.1.35 Settings\Joystick—specifies whether a joystick is being used (1) or not (0).

6.6.1.36 Settings\JoystickSpeech—specifies whether audio prompts will be used (1) or not (0) while operating a joystick.

6.6.1.37 Settings\LastSequenceOpen—stores the last sequence open for editing in the Sequence Editor so that the same sequence will be displayed the next time the Sequence Editor is opened.

6.6.1.38 Settings\MaintainGripAfterShutdown—specifies whether to leave gripper motor energized (1) or de-energized (0) after ShutDown function is called.

6.6.1.39 Settings\ModelessSubWindows—specifies whether Teach Pendant sub-windows should be modeless or modal. A value of “True” specifies modeless, and “False” specifies modal.

6.6.1.40 Settings\MotionRulesEnabled—specifies whether motion rules will be used.

6.6.1.41 Settings\RecentOperation0-4 – Used in Sequence Editor right-click menu for adding recently used operations.

6.6.1.42 Settings\ReferenceFrameAngle—stores the last value specified in the Cartesian Jog settings in the Teach Pendant.

6.6.1.43 Settings\RobotIPAddress – Speciies the IP address for the Ethernet gateway if the Interface Type is set to Ethernet.

6.6.1.44 Settings\SaveMoveDataOnShutDown—specifies whether move cycle data will be saved to the robot motor drives when ShutDown() function is called (1 = enabled, 0 = disabled)

6.6.1.45 Settings\SequenceEditorHeight, Width – stores the size of the Sequence Editor window.

6.6.1.46 Settings\SequenceEditorTop, Left – stores the position of the Sequence Editor window.

6.6.1.47 Settings\ServoGripperClosedPosition – specifies the default closed position for the gripper.

6.6.1.48 Settings\ServoGripperOpenPosition – specifies the default open position of the gripper.

6.6.1.49 Settings\ServoGripperForcePercent – specifies the default force percent (10-100) of the gripper.

6.6.1.50 Settings\TeachPendantAccessLevel—specifies the current access level for the Teach Pendant (0=Full, 1=Technician, 2=Limited).

6.6.1.51 Settings\TeachPendantFullAccessPW—stores the encrypted Full Access password.

6.6.1.52 Settings\TeachPendantHeight, Width – stores the size of the Teach Pendant window.

6.6.1.53 Settings\TeachPendantTop, Left – stores the position of the Teach Pendant window.

6.6.1.54 Settings\TeachPendantTechnicianAccessPW—stores the encrypted Technician Access password.

6.6.1.55 Settings\ToolOffset—stores the last value specified in the Cartesian Jog settings in the Teach Pendant.

6.6.1.56 Settings\UseDefaultGripperShutdownState—specifies whether the position of the gripper will be altered during ShutDown based on the default gripper shutdown state settings listed above (1=enabled, 0=disabled).

6.6.1.57 Settings\UseDefaultGripperState—specifies whether the position of the gripper will be altered after homing during Initialize based on the default gripper state settings listed above (1=enabled, 0=disabled).

6.6.1.58 Settings\VelLimEnabled—if set to a value of 1, then the specified digital input will be monitored. The velocity of the robot will be limited to a specified value for all moves when the input is in the specified state. If set to a value of 0, then velocity will not be limited.

6.6.1.59 Settings\VelLimInputAxis—specifies the axis of the digital input to be monitored for limiting the robot velocity.

6.6.1.60 Settings\VelLimInputNum—specifies the digital input number to be monitored for limiting the robot velocity.

6.6.1.61 Settings\VelLimInputState—specifies the state in which the velocity-limiting digital input must be in order to limit the velocity of the robot.

6.6.1.62 Settings\VelLinearLimMax—specifies the maximum linear velocity at which the robot will be allowed to operate when the velocity-limiting input is in the state that limits the velocity. Expressed as a percentage of maximum.

6.6.1.63 Settings\VelLimMax—specifies the maximum velocity at which the robot will be allowed to operate when the velocity-limiting input is in the state that limits the velocity. Expressed as a percentage of maximum.

6.6.1.64 Settings\WarningAfterIdleTime—length of time, in seconds, that must elapse before a move warning will occur.

6.6.1.65 Settings\WarningBeforeMove—when enabled (1), the robot buzzer will sound, and the indicator will flash if the robot has been idle for a period longer than specified by WarningAfterIdleTime. Set to 0 to disable the warning.

6.6.1.66 Settings\WarningDuration—duration, in seconds, of the warning.

6.6.1.67 MoveData\AxisTravelElbow—stores the total number of full arm extensions completed by the elbow. If the arm starts in the fully retracted position, extends completely, and then returns to the fully retracted position, then this value will increase by 2.

6.6.1.68 MoveData\AxisTravelGripper—stores the total motion, in meters, completed by the gripper.

6.6.1.69 MoveData\AxisTravelRail—stores the total motion, in meters, completed by the rail.

6.6.1.70 MoveData\AxisTravelShoulder—stores the total number of full rotations completed by the shoulder.

6.6.1.71 MoveData\AxisTravelWrist—stores the total number of full rotations completed by the wrist.

6.6.1.72 MoveData\AxisTravelZ—stores the total motion, in meters, completed by the Z axis.

6.6.1.73 MoveData\LastMaintenancePerformed—stores the total Z motion, in meters, at the time when Z axis maintenance was last performed.

6.6.1.74 MoveData\LastMaintenancePerformedDate—stores the date when maintenance was last performed on the Z axis (20200618 = June 18th, 2020).

6.6.1.75 MoveData\LastMaintenancePerformedRail—stores the total Rail motion, in meters, at the time when rail maintenance was last performed.

6.6.1.76 MoveData\LastMaintenancePerformedDateRail—stores the date when maintenance was last performed on the rail (20200618 = June 18th, 2020).

6.6.1.77 MoveData\MaintenanceInterval—stores the interval, in meters, at which maintenance must be performed on the Z axis.

6.6.1.78 MoveData\MaintenanceIntervalRail—stores the interval, in meters, at which maintenance must be performed on the rail.

6.6.1.79 MoveData\MaintenanceRequired—indicates when Z axis maintenance is required (1 = required, 0 = not required).

6.6.1.80 MoveData\MaintenanceRequiredRail—indicates when rail maintenance is required (1 = required, 0 = not required).

6.6.1.81 MoveData\MoveCount—stores the total number of moves, regardless of the distance moved, executed by the robot.

6.6.1.82 MoveData\SerialNumber—stores the serial number of the robot that is connected to the PC.

6.6.1.83 MoveData\SerialNumberRail—stores the serial number of the rail that is connected to the PC.

6.6.1.84 VisionOffset\OffsetRef3DCoordX,Y,Z, OffsetRefQuaternionW,X,Y,Z, ZeroRef3DCoordX,Y,Z, ZeroRefQuaternionW,X,Y,Z—store the settings specified by the last call to the SetVisionOffset function.

6.6.1.85 VisionOffset\VisionOffsetEnabled – stores the setting specified by the last call to EnableVisionOffset.

Note: All values stored in the MoveData section are temporary values that are saved to the robot non-volatile internal memory each time the ShutDown() method is called.

6.6.2 Most of the values listed above can be changed in the “Teach Pendant/Tools/Options…” window.

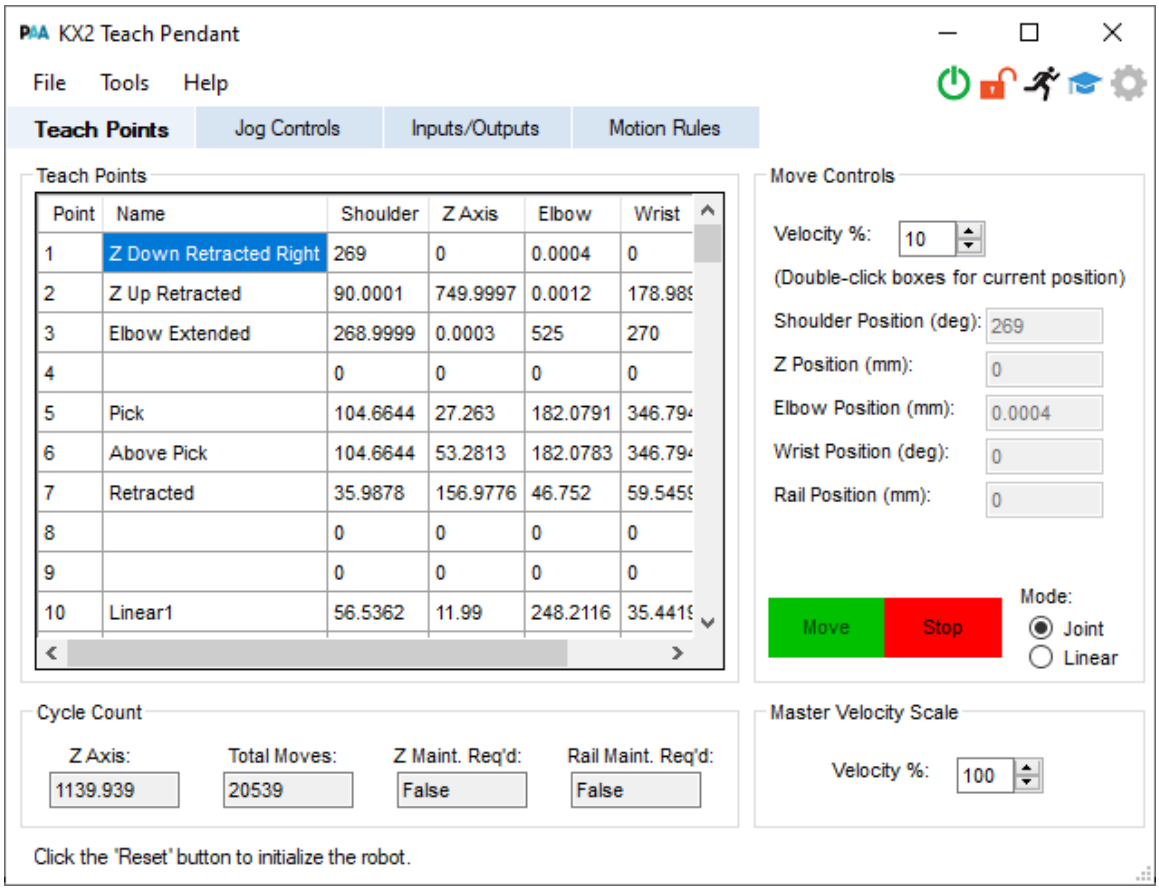
7.0 Teach Pendant Dialog

7.1 A dialog window is included in the "KX2RobotControl.dll" program. The dialog provides the operator with functionality to perform the following actions:

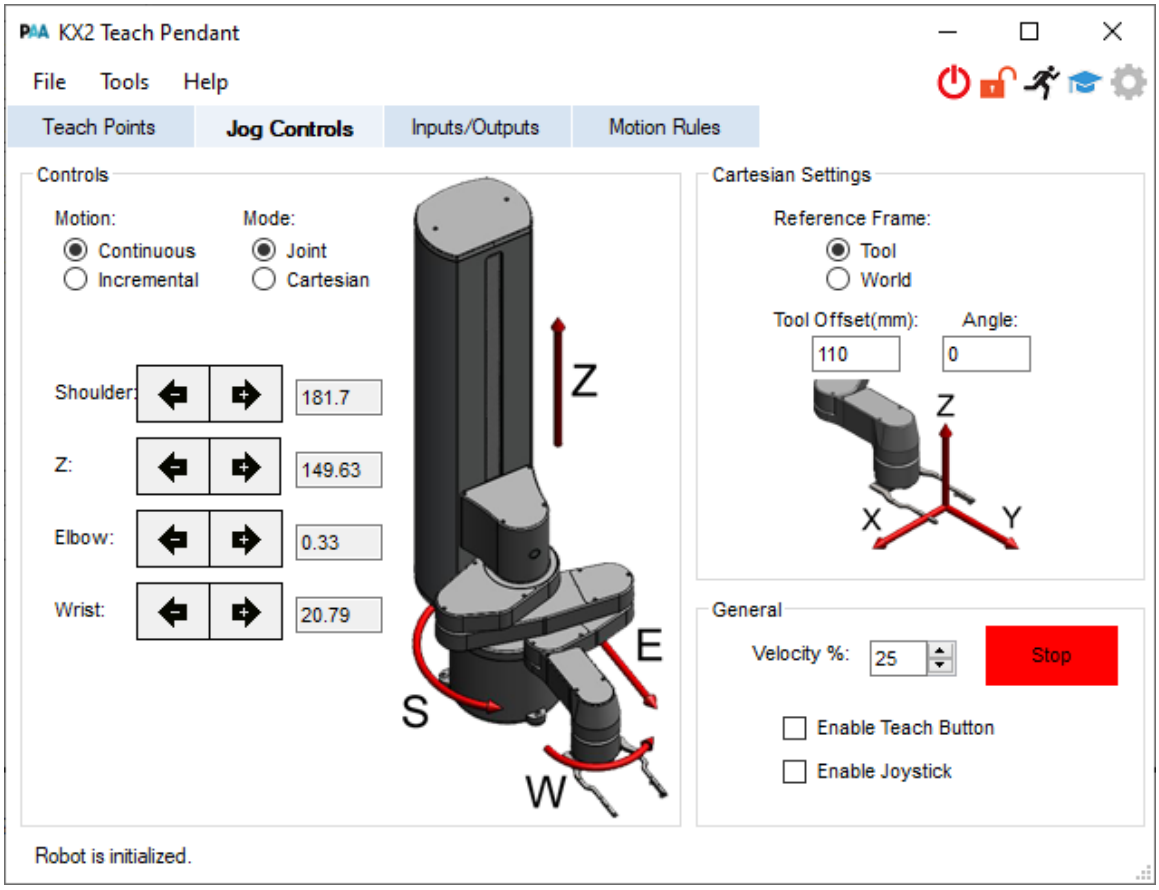
- 7.1.1 Jog individual motors
- 7.1.2 Open and close the gripper
- 7.1.3 Move the robot to Teach Points
- 7.1.4 Load, name, teach, save, delete, and reorder Teach Points
- 7.1.5 Read digital and analog inputs
- 7.1.6 Set outputs
- 7.1.7 Operate the barcode reader
- 7.1.8 Define motion rules to reduce the chance of accidental collisions
- 7.1.9 Calibrate the integrated vision system
- 7.1.10 Create simple robot sequences

7.2 Refer to the "TeachPendantShow" method (later in this document) for instructions on showing the Teach Pendant Dialog.

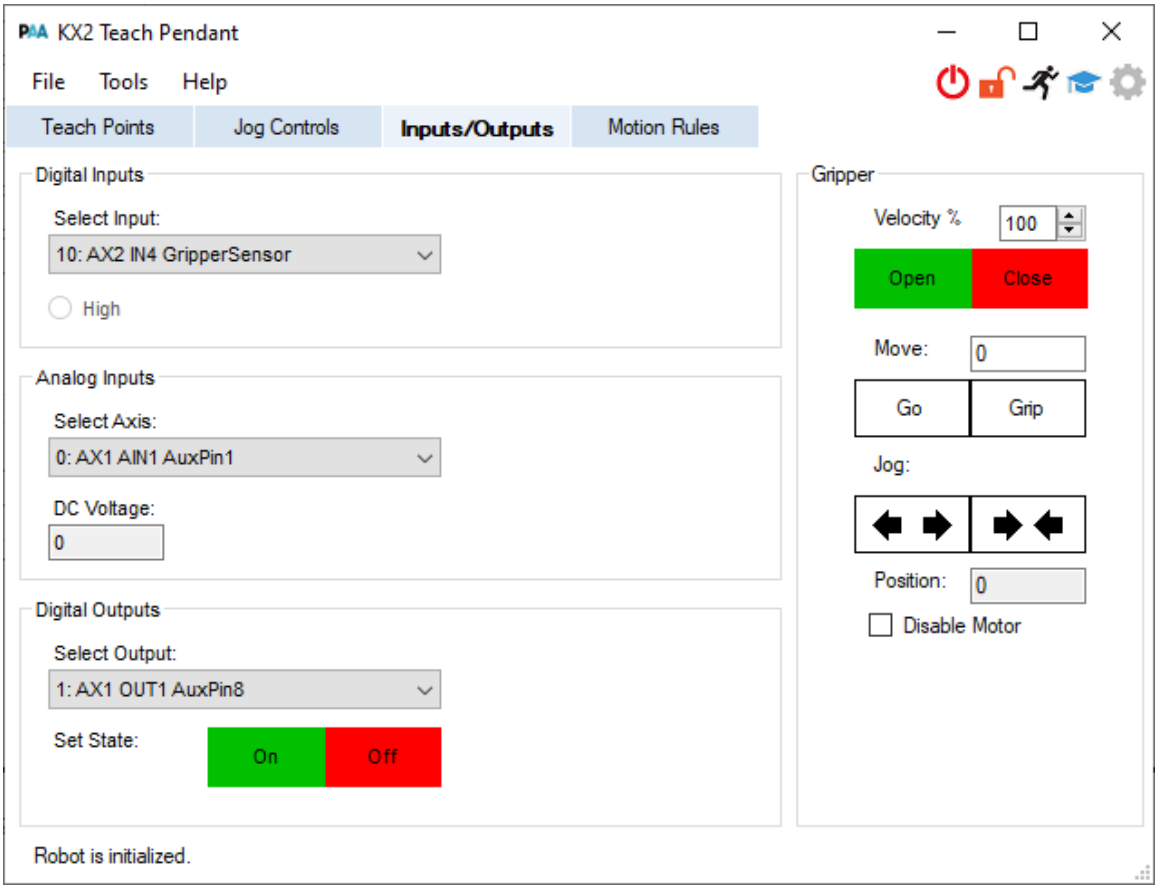
7.3 Refer to the Robot Maintenance Manual for instructions on using the Teach Pendant.



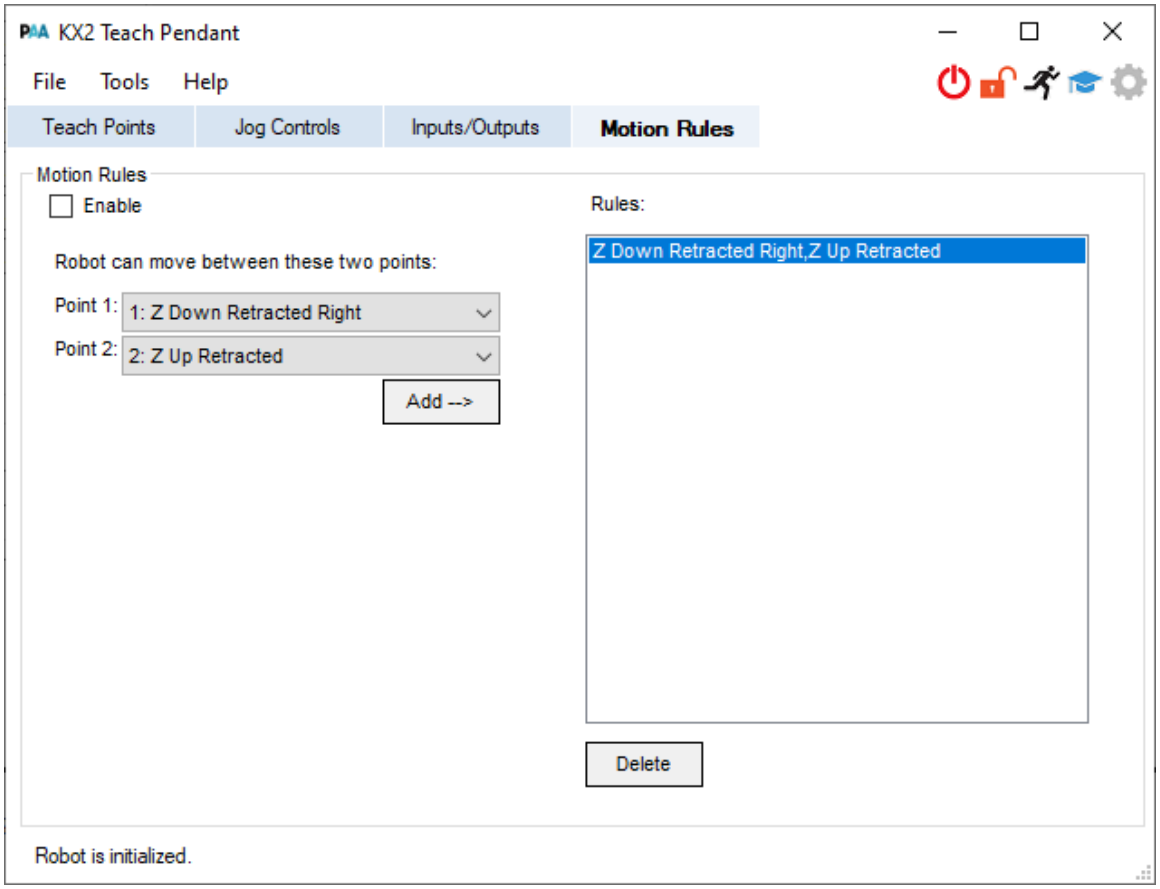
Teach Pendant – Teach Points Tab



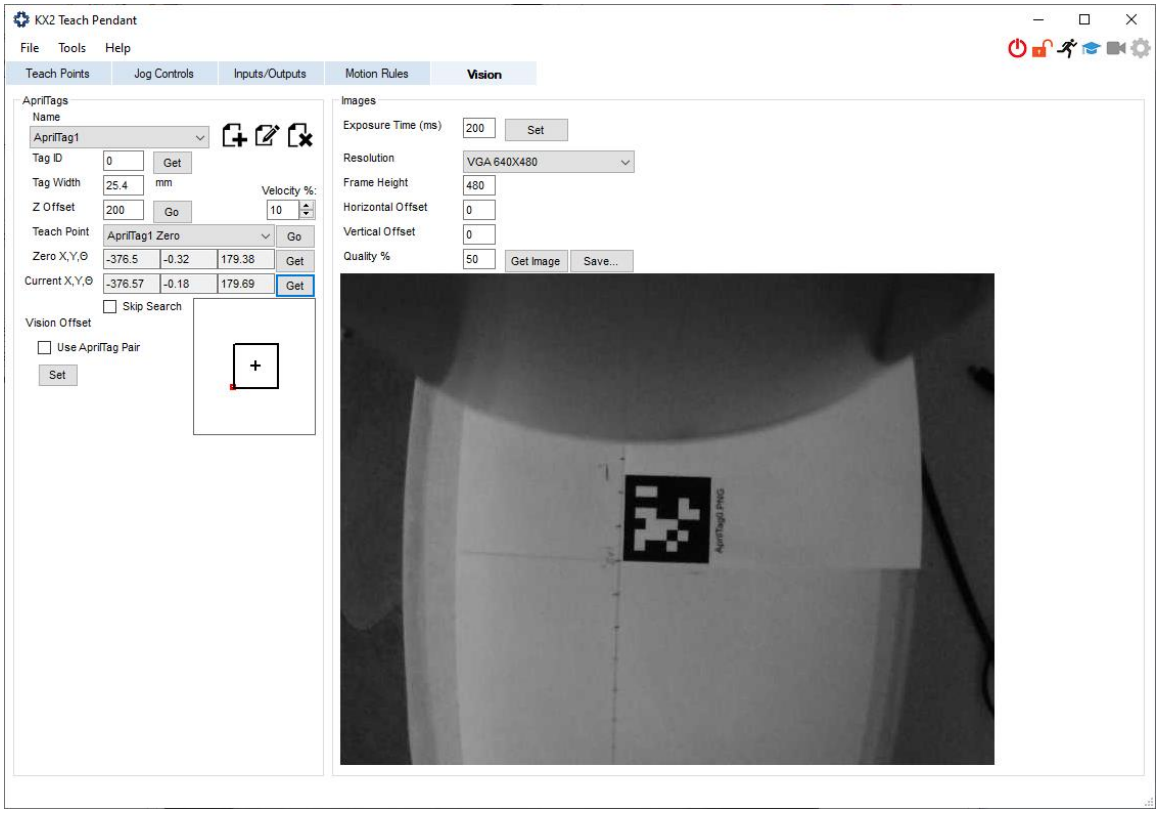
Teach Pendant – Jog Controls Tab



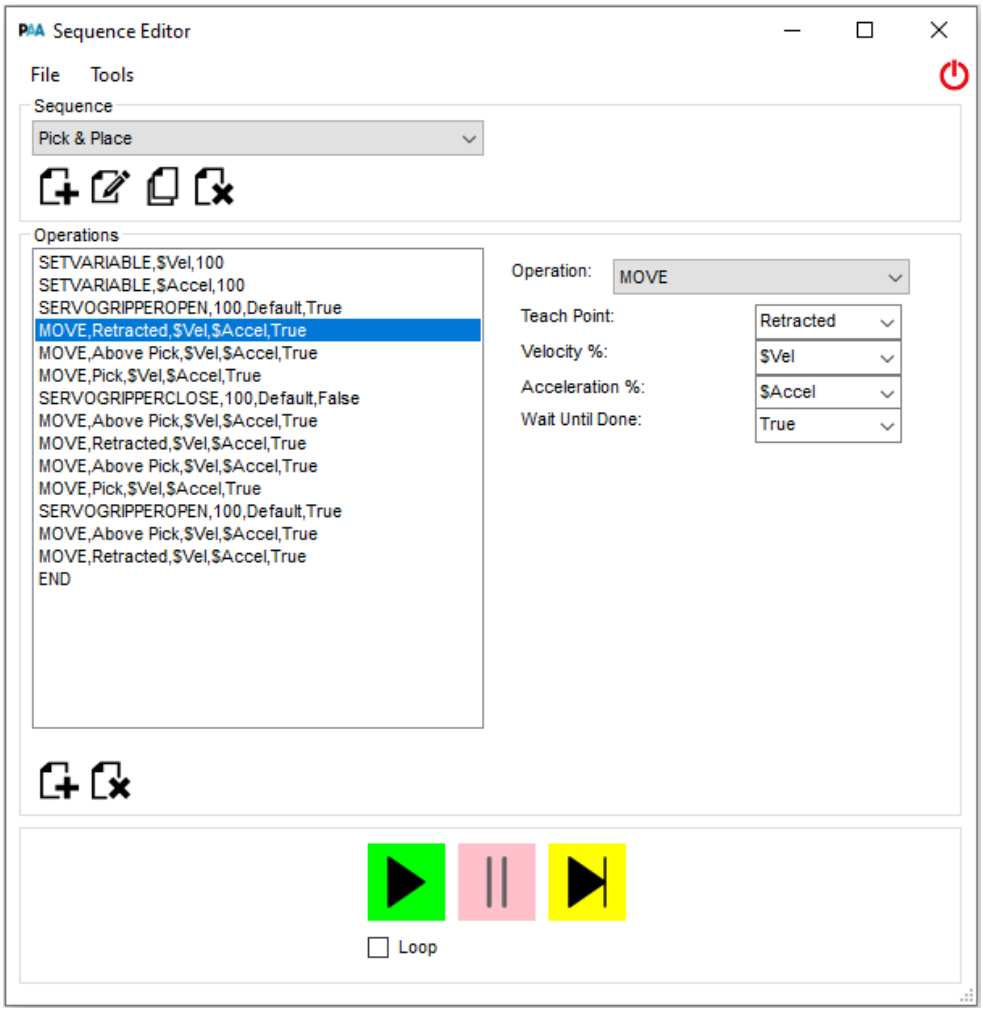
Teach Pendant – Inputs/Outputs Tab



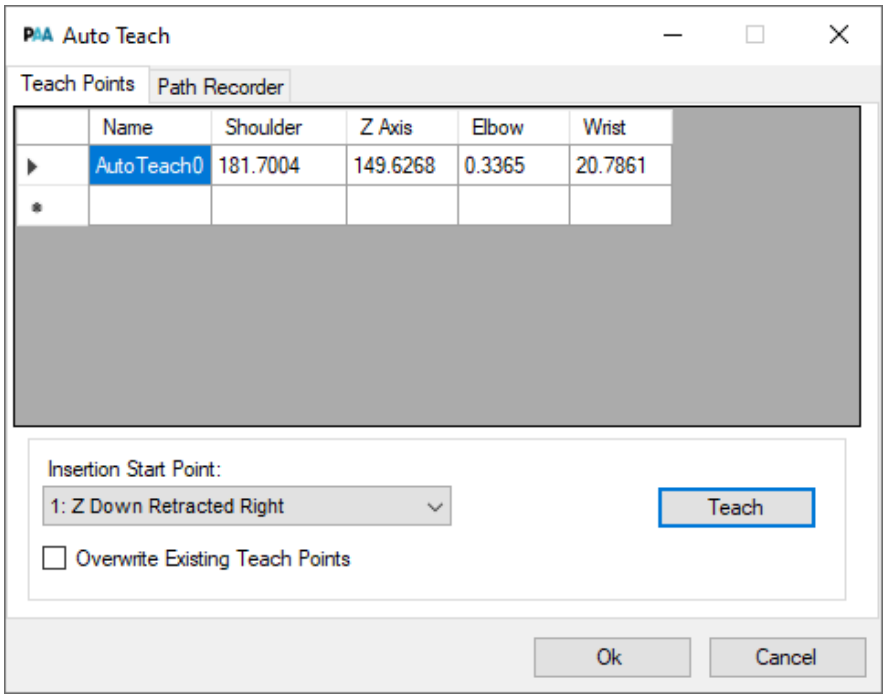
Teach Pendant – Motion Rules Tab



Teach Pendant – Vision Tab



Sequence Editor



AutoTeach Window

8.0 Error Codes Returned by Methods

Note: The following error codes are passed as the return value for most methods. A zero indicates no error

```
ErrorCode(1) = "" 'Reserved for passing through axis-specific error information.
ErrorCode(2) = "" 'Reserved for passing through resume step for stack/hotel
functions
ErrorCode(3) = "" 'Reserved for passing through CAN errors
ErrorCode(10) = "Communication failure (see Error Code 1 for more information)."
ErrorCode(11) = "Failed to establish communication with motor drives. Verify power
supply is turned on."
ErrorCode(15) = "Robot failed to complete absolute move."
ErrorCode(16) = "Motor finished moving sooner than expected (see Error Code 1 for
axis number)."
ErrorCode(17) = "Servo gripper failed to close."
ErrorCode(20) = "Motor failed to find home position."
ErrorCode(25) = "Z brake output cannot be controlled manually."
ErrorCode(30) = "Invalid Axis number."
ErrorCode(35) = "Invalid Velocity value (0.1-100)."
ErrorCode(40) = "Invalid Acceleration value (0.1-100)."
ErrorCode(45) = "Move exceeds maximum travel limit (see Error Code 1 for axis
number)."
ErrorCode(46) = "Move exceeds maximum velocity limit (see Error Code 1 for axis
number)."
ErrorCode(47) = "Move exceeds maximum acceleration limit (see Error Code 1 for
axis number)."
ErrorCode(50) = "No Teach Point exists with specified name."
ErrorCode(55) = "The specified Teach Point name is already in use."
ErrorCode(60) = "MotorsMoveLinear queue error."
ErrorCode(69) = "MoveToArrayPoint TeachPoint array parameter must have two (1D),
three (2D) or four (3D) indices (see Error Code 1 for more information)."
ErrorCode(70) = "MoveToArrayPoint NumPoints array parameter number of indices must
be one less than TeachPoint array parameter (see Error Code 1 for more
information)."
ErrorCode(71) = "MoveToArrayPoint MovePoint array parameter number of indices must
be the same as NumPoints array parameter (see Error Code 1 for more information)."
ErrorCode(72) = "MoveToArrayPoint NumPoints array parameter must contain only
positive values (see Error Code 1 for more information)."
ErrorCode(73) = "MoveToArrayPoint MovePoint array parameter violates range defined
by NumPoints array parameter (see Error Code 1 for more information)."
ErrorCode(75) = "Motor failed to complete homing routine (see Error Code 1 for
axis number)."
ErrorCode(80) = "Robot is in motion.  A new move cannot be started until it has
been verified that the previous move finished.  Robot may need to be
reinitialized."
ErrorCode(85) = "If Teach Point name is three characters long or less, it must not
be numeric."
ErrorCode(86) = "Teach Point name must contain no commas or semicolons."
ErrorCode(90) = "Invalid Teach Point name."
ErrorCode(95) = "Invalid Teach Point number (must be between 1 and 100)."
ErrorCode(100) = "Failed to read cached drive parameters from Registry (see Error
Code 1 for more information)."
```

```
ErrorCode(105) = "Combined Elbow and Shoulder Move will exceed the Shoulder travel
limit."
ErrorCode(110) = "Illegal Move. Arm would hit base."
ErrorCode(115) = "Object not found by sensor."
ErrorCode(120) = "Robot is not initialized."
ErrorCode(125) = "Unable to read Parameter file."
ErrorCode(126) = "Please specify the location of the rail parameter file in Teach
Pendant/Tools/Options."
ErrorCode(127) = "Missing Vision Calibration file name. Must be specified in
Options window."
ErrorCode(130) = "Gripper sensor is already tripped.  Cannot perform search."
ErrorCode(135) = "Input failed to change to desired state."
ErrorCode(150) = "Sensor failed to change state during object search."
ErrorCode(155) = ""
ErrorCode(160) = "A Move Sequence is already running."
ErrorCode(165) = "Unable to write to Parameter file."
ErrorCode(170) = "Failed to connect to Barcode Reader (see Error Code 1 for more
information)."
ErrorCode(175) = "Failed to set Barcode Reader BAUD Rate (see Error Code 1 for
more information)."
ErrorCode(180) = "Failed to receive software version from Barcode Reader (see
Error Code 1 for more information)."
ErrorCode(185) = "Failed to send command to Barcode Reader (see Error Code 1 for
more information)."
ErrorCode(186) = "Failed to receive response from Barcode Reader."
ErrorCode(187) = ""
ErrorCode(188) = ""
ErrorCode(190) = "Gripper sensor failed to detect object in gripper."
ErrorCode(195) = "Gripper sensor detected object in gripper."
ErrorCode(200) = "Z Axis must be above stack before moving to stack."
ErrorCode(205) = "Unable to read Script file."
ErrorCode(210) = "Invalid script command (see Error Code 1 for more information)."
ErrorCode(215) = "Missing script parameter (see Error Code 1 for more
information)."
ErrorCode(220) = "Input is in incorrect state (see Error Code 1 for more
information)."
ErrorCode(225) = "A Script is already running."
ErrorCode(230) = "INI File does not exist."
ErrorCode(235) = "INI File contains no headers."
ErrorCode(240) = "Hardstop search can only be performed on servo gripper."
ErrorCode(241) = "Index search can only be performed on servo gripper."
ErrorCode(245) = "Robot power has been cycled.  Robot must be reinitialized."
ErrorCode(250) = "Unable to read Teach Point file."
ErrorCode(255) = "Unable to read INI file (see error code 1 for more
information)."
ErrorCode(260) = "Robot needs to be reinitialized. Verify the following:
       -Robot power is on
       -USB cable is connected to the PC
       -Emergency stop button is pulled out"
ErrorCode(265) = "Robot on rail not supported by this function."
ErrorCode(270) = "Gripper sensor is already tripped.  Cannot perform search.
SecondSearchStartHeightMM may be too small."
ErrorCode(275) = "Robot failed to stop at top of stack."
ErrorCode(280) = "No default file specified."
ErrorCode(285) = "Plate stack is empty."
```

```
ErrorCode(290) = "Plate stack is full."
ErrorCode(291) = "PlateNumber must be greater than 0 and no greater than
StackCapacity."
ErrorCode(292) = "PlateNumber must be greater than 0 and no greater than
HotelCapacity."
ErrorCode(295) = "Emergency Stop circuit has been activated.  Verify the
following:
       -Emergency stop button is pulled out"
ErrorCode(298) = "Emergency Stop circuit has been activated. Verify the following:
       -Emergency stop button is pulled out"
ErrorCode(300) = "Motion rule already exists."
ErrorCode(301) = "Invalid Emergency Stop circuit state. STO1=High STO2=Low"
ErrorCode(302) = "Invalid Emergency Stop circuit state. STO1=Low STO2=High"
ErrorCode(305) = "A motion rule must consist of two separate teach points."
ErrorCode(310) = "Robot is not currently positioned at a teach point."
ErrorCode(311) = "Robot is not at the desired teach point (see Error Code 1 for
more information)."
ErrorCode(315) = "Serial Port error (check ErrorLog.txt for details)."
ErrorCode(320) = "Serial Port Wait Timeout."
ErrorCode(325) = "Motor did not reach commanded position (see Error Code 1 for
more information)."
ErrorCode(330) = "Servo Gripper was able to move all the way to the closed
position, which indicates the absence of an object in the gripper.  The closed
position value may need to be decreased."
ErrorCode(331) = "Servo Gripper did not reach the expected gripping force while
closing."
ErrorCode(335) = "Servo Gripper is already positioned beyond the defined Closed
position."
ErrorCode(336) = "Servo Gripper is already positioned beyond the defined Open
position."
ErrorCode(337) = "Servo Gripper Close command timed out."
ErrorCode(345) = "Infinite loop encountered in sequence."
ErrorCode(350) = "Failed to read Analog Input (see Error Code 1 for axis number)."
ErrorCode(355) = "Commanded move exceeds the travel limits of the robot.  If the
move path is linear, then it is likely that the path passes too close to the
center of the robot (see Error Code 1 for more information)."
ErrorCode(361) = "Robot must be homed before absolute encoders can be calibrated."
ErrorCode(362) = "The specified axis is not equipped with an absolute encoder."
ErrorCode(363) = "Absolute encoder calibration data is invalid (see Error Code 1
for more information)."
ErrorCode(364) = "Failed to read absolute encoder on socket 2 (see Error Code 1
for more information)."
ErrorCode(370) = "Script variable must be assigned a name."
ErrorCode(371) = "Script variable must be assigned a value."
ErrorCode(372) = "The name 'Variables' is reserved."
ErrorCode(373) = "The first character of variable name must be a '$'."
ErrorCode(374) = "The variable name cannot contain any '+' or '-' symbols."
ErrorCode(375) = "Flag not found in script (see Error Code 1 for more
information)."
ErrorCode(380) = "Unable to find registry subkey."
ErrorCode(385) = "Unable to overwrite existing file."
ErrorCode(390) = "Command not compatible with current robot type (see Error Code 1
for more information)."
ErrorCode(394) = "Image download error (see Error Code 1 for more information)."
```

```
ErrorCode(395) = "Vision offset must be specified using API SetVisionOffset
function call prior to being enabled."
ErrorCode(396) = "Camera is not present.
ErrorCode(397) = "No AprilTag found."
ErrorCode(398) = "Failed to calculate AprilTag edge length.”
ErrorCode(399) = "Failed to find AprilTag with ID that matches specified ID."
```

Note: The following error codes are passed via ErrorCode(3):

```
CanErrorCode(4) = "Unexpected CAN response: " 'More descriptive information will
be added to end of message
CanErrorCode (5) = "CAN Status: "  'More descriptive information will be added to
end of message
CanErrorCode (10) = "Motor fault.  "  'More descriptive information will be added
to end of message
CanErrorCode (15) = "PVT fault.  "  'More descriptive information will be added to
end of message
CanErrorCode (20) = "Motor failed to enable."
CanErrorCode (25) = "Motor failed to disable."
CanErrorCode (30) = "Motion timed out."
CanErrorCode (35) = "Homing Failure 1: Failed to set Hard Stop Search
Deceleration."
CanErrorCode (36) = "Homing Failure 2: Failed to find hard stop."
CanErrorCode (37) = "Homing Failure 3: Failed to stop after finding hard stop."
CanErrorCode (38) = "Homing Failure 4: Hard stop search timed out."
CanErrorCode (39) = "Homing Failure 5: Failed to move away from hard stop."
CanErrorCode (40) = "Homing Failure 6: Failed to finish index pulse search."
CanErrorCode (41) = "Homing Failure 7: Failed to move to index offset."
CanErrorCode (43) = "Homing an axis equipped with a BiSS absolute encoder is not
allowed."
CanErrorCode (44) = "No CAN devices found."
CanErrorCode (45) = "Failed to receive response from motor."
CanErrorCode (46) = "Connection to CAN device is closed."
CanErrorCode (50) = "Total linear move time cannot exceed 16 seconds (255msec per
time interval)."
CanErrorCode (55) = "Unable to find the library PCANBasic.dll."
CanErrorCode (60) = "Maximum allowed command length is 500 characters."
CanErrorCode (65) = "Unexpected SDO Transfer Toggle value."
CanErrorCode (70) = "Move aborted due to preexisting error state."
CanErrorCode (75) = "Failed to receive response from camera."
CanErrorCode (76) = "Received incorrect response from camera."
CanErrorCode (77) = "Camera not available."
CanErrorCode (78) = "No AprilTag found."
CanErrorCode (79) = "Camera error."
```

9.0 Descriptions of Methods

9.1 Use of Arrays

9.1.1 Some method parameters are arrays.

9.1.2 When an array has indices that correspond to robot axes, index 0 is ignored, and axis 1 is index 1.

9.1.3 When an array has indices that correspond to Cartesian coordinates, the following rules are to be followed:

Index 0 = X
Index 1 = Y
Index 2 = Z
Index 3 = Theta (absolute wrist position)
Index 4 = Rail Axis (if present)

9.2 ApplyVisionOffset

(ByVal zeroJointPos() As Double, ByRef offsetJointPos() As Double)

Parameters:

zeroJointPos—An array containing robot joint positions

offsetJointPos—Returns an array of robot joint positions that represents zeroJointPos with the vision offset applied to it

Description:

Applies the vision offset specified by SetVisionOffset to a robot joint position array. This function is called automatically by the DLL internal functions when EnableVisionOffset(True) has been called.

9.3 AprilTagDeleteCalibrationFromFile

(Filename As String, Name As String)

Parameters:

Filename—Specify the path and name of the file to be read

Name—Specify the name of the AprilTag record to be deleted

Description:

Deletes the specified AprilTag calibration record from the specified file.

9.4 AprilTagReadCalibrationFromFile

(Filename As String, Name As String, ByRef AprilTag As tAprilTag) As Short

Parameters:

Filename—Specify the path and name of the file to be read

Name—Specify the name of the AprilTag record to be read

AprilTag—Retrieves a tAprilTag structure containing the following values: ID, Width, ZOffset, TeachPoint, ZeroX, ZeroY, ZeroTheta, CurrX, CurrY, CurrTheta

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Retrieves the calibration data for specified AprilTag calibration record from the specified file.

9.5 AprilTagsGetNamesFromFile

(Filename As String, ByRef Names() As String) As Short

Parameters:

Filename—Specify the path and name of the file to be read

Names()—Retrieves an array containing all of the AprilTag calibration record names in the file

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Retrieves a list of all AprilTag calibration records contained in the vision calibration file.

9.6 AprilTagWriteCalibrationToFile

(Filename As String, Name As String, AprilTag As tAprilTag) As Short

Parameters:

Filename—Specify the path and name of the file to be read

Name—Specify the name of the AprilTag record to be read

AprilTag—Specifies the calibration settings using a tAprilTag structure containing the following values: ID, Width, ZOffset, TeachPoint, ZeroX, ZeroY, ZeroTheta, CurrX, CurrY, CurrTheta

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Sets the calibration data for specified AprilTag calibration record in the specified file.

9.7 AssignStringToDoubleArray

(CrSeparatedString As String, DoubleArray() As Double, LowerBound As Short, Optional Separator As String)

Parameters:

CrSeparatedString—A string of decimal numbers separated by carriage returns or semicolons.

DoubleArray()—An unbound array. Returned containing the values found in CrSeparatedString

LowerBound—This optional parameter specifies the lower bound of DoubleArray. The default value is zero

Separator—This optional parameter specifies the character that is used to separate the values in the array. Either a semicolon or a carriage return can be used as a separator without having to specify the character with the Separator parameter.

Description:

Converts a carriage return-separated (or semicolon-separated) string of decimal numbers into an array.

9.8 AssignStringToIntegerArray

(CrSeparatedString As String, IntegerArray() As Short, LowerBound As Short)

Parameters:

CrSeparatedString—A string of Short numbers separated by carriage returns or semicolons.

IntegerArray()—An unbound array. Returned containing the values found in CrSeparatedString

LowerBound—This optional parameter specifies the lower bound of ShortArray. The default value is zero

Description:

Converts a carriage return-separated (or semicolon-separated) string of Integer numbers into an array.

9.9 AssignStringToStringArray

(CrSeparatedString As String, StringArray() As Short, LowerBound As Short)

Parameters:

CrSeparatedString—A string of individual strings separated by carriage returns or semicolons.

StringArray()—An unbound array. Returned containing the values found in CrSeparatedString

LowerBound—This optional parameter specifies the lower bound of StringArray. The default value is zero

Description:

Converts a carriage return-separated (or semicolon-separated) string of individual strings into an array.

9.10 AutoTeachWindowShow

(Show As Boolean, Optional WindowTop As Short, Optional WindowLeft As Short)

Parameters:

Show—Specifies whether window should be shown or hidden.

WindowTop—This optional parameter specifies the startup top position of the window. The default is zero (Windows controls the location).

WindowLeft—This optional parameter specifies the startup left position of the window. The default is zero (Windows controls the location).

Description:

Can be used to show or hide the AutoTeach window. The user may not want the window to be shown at all times. This window is shown in Modal state only.

9.11 BarcodeRead

(WaitUntilDone As Boolean, Optional ReadMode As eBCRReadMode, Optional ReadTime As eBCRReadTime, Optional Barcode As String) As Short

Parameters:

WaitUntilDone—If set to True, then the function call will be blocking until either a barcode has been read successfully or the time specified by ReadTime has elapsed (60 seconds max.). If set to False, then the function call will activate the barcode reader, but it will return prior to receiving a reading. In this case, the reading can be received via the BarcodeReaderData event.

ReadMode—Specifies one of the following read modes:

SingleRead (default)—Only one barcode will be read

MultipleRead—Will read continuously until the period specified by ReadTime has elapsed

ContinuousRead—Will read continuously without timeout. Can be canceled by calling BarcodeReaderSetReadMode(SingleRead)

ReadTime—Specifies the time interval either for SingleRead timeout or for MultipleRead duration. Available periods are OneSecond – NineSeconds or Indefinite.

Barcode—Returned containing the value of the first successfully read barcode if WaitUntilDone = True.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Call this function to read a 1D or 2D barcode with the barcode reader that is mounted in the rear of the robot gripper. To read a single barcode, set WaitUntilDone = True, and the barcode value will be passed back via the Barcode parameter. To read multiple barcodes while moving the robot, set WaitUntilDone = False, ReadMode = MultipleRead and ReadTime to an adequate time that is longer than the duration of the move. Retrieve the barcodes via the BarcodeReaderData event.

9.12 BarcodeReaderAutoTrigger

(Enable As Boolean) As Short

Parameter:

Enable—A value of True will enable AutoTrigger, and a value of False will disable AutoTrigger.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

When AutoTrigger is enabled, then the barcode reader will read automatically when a barcode is in view. Otherwise, BarcodeReaderTrigger must be called to trigger a reading. BarcodeRead is recommended to be used instead of these lower-level commands.

9.13 BarcodeReaderConnect

(Connect As Boolean) As Short

Parameter:

Connect—If equals True, then the serial port will be opened and communication with the barcode reader will be attempted. If equals False, then the serial port will be closed.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function is called automatically by the Initialize function, so it should not be necessary to call this function directly. One practical use of this function is as part of a search through all available COM ports to locate the robot barcode reader.

9.14 BarcodeReaderGetSoftwareVersion

(SWVersion As String) As Short

Parameter:

SWVersion—Passes back the barcode reader software version, such as ” BD01J08”.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method returns the firmware version of the robot barcode reader.

9.15 BarcodeReaderSendCommand

(Cmd As String, Optional TimeoutMsec As Long, Optional Response As String) As Short

Parameters:

Cmd—Specifies the command to be sent to the barcode reader.

TimeoutMsec—Specifies the duration, in milliseconds, that the function will wait for a response from the barcode reader. Timeout is active only if Response is passed into the method as a non-empty initial value.

Response—If the command being sent elicits a response from the barcode reader, and this parameter is initialized as a non-empty string (set it to “?” or similar) then the response will be passed back through this parameter.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This method provides a means to send any command supported by the MDI-4100 barcode reader. Refer to Opticon MDI-4000 Series Serial Interface Manual for a list of available commands (available online).

9.16 BarcodeReaderSetReadMode

(ReadMode As eBCRReadMode) As Short

Parameter:
ReadMode—Specifies one of the following read modes:

SingleRead (default)—Only one barcode will be read

MultipleRead—Will read continuously until the period specified by ReadTime has elapsed

ContinuousRead—Will read continuously without timeout. Can be canceled by calling BarcodeReaderSetReadMode(SingleRead)

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This method sets the read mode of the robot barcode reader to single, multiple or continuous read. The read mode can also be set via the BarcodeRead method.

9.17 BarcodeReaderSetReadTime

(ReadTime As eBCRReadTime) As Short

Parameter:
ReadTime—Specifies the time interval either for SingleRead timeout or for MultipleRead duration. Available periods are OneSecond – NineSeconds or Indefinite.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method sets the read time of the robot barcode reader to anywhere between one and nine seconds or indefinite. The read time can also be set via the BarcodeRead method.

9.18 BarcodeReaderTrigger

(Trigger As Boolean) As Short

Parameter:

Trigger—If True, then the barcode reader will be activated to take a reading. If False, then an active reading will be aborted.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method activates the robot barcode reader to take a reading, based on prior calls of the BarcodeReaderSetReadMode and BarcodeReaderSetReadTime. Read results will be passed back via the BarcodeReaderData event. This method is called by BarcodeRead.

9.19 CameraCaptureImage

(ByRef AprilTagCount As Byte) As Short

Parameter:

AprilTagCount—Retrieves the number of AprilTags found in the image

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Commands the integrated camera to acquire an image. The image is then stored in the camera memory. The camera analyzes the image and returns the number of AprilTags present.

9.20 CameraGetAprilTag

(AprilTagNum As Byte, ByRef AprilTagData As CameraAprilTagData) As Short

Parameters:

AprilTagNum—Specifies the index of the AprilTag. For example, if CameraCaptureImage() returns AprilTagCount = 2, then AprilTagNum should be set to either 0 or 1

AprilTagData—Retrieves the values associated with the AprilTag as a class with the following properties: x, y, w, h, id, family, cx, cy, rotation, decision_margin, hamming, goodness, x_translation, y_translation, z_translation, x_rotation, y_rotation, z_rotation

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Retrieves the properties of an AprilTag that has been identified by the integrated camera as a result of calling CameraCaptureImage().

9.21 CameraGetAprilTagCornerCoordinates

(AprilTagNum As Byte, ByRef AprilTagCoord As t2dCoord()) As Short

Parameters:

AprilTagNum—Specifies the index of the AprilTag. For example, if CameraCaptureImage() returns AprilTagCount = 2, then AprilTagNum should be set to either 0 or 1

AprilTagCoord()—Retrieves the four corner coordinates (x,y) of the AprilTag in units of camera pixels relative to the upper left corner of the camera image

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Retrieves the corner coordinates of an AprilTag that has been identified by the integrated camera as a result of calling CameraCaptureImage().

9.22 CameraGetError

(ByRef Msg As String) As Short

Parameter:

Msg—Retrieves a message that describes the last exception thrown by the camera

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

If the program running on the integrated camera throws an exception, then this function can be used to retrieve a description of the exception.

9.23 **CameraGetAprilTagID**

(AprilTagNum As Byte, ByRef AprilTagID As Short) As Short

Parameters:

AprilTagNum— Specifies the index of the AprilTag. For example, if CameraCaptureImage() returns AprilTagCount = 2, then AprilTagNum should be set to either 0 or 1

AprilTagID—Retrieves the ID encoded into the AprilTag

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

The integrated camera reads AprilTags with tag36h11 format. The information encoded in each AprilTag is an ID that is any value in the range of 0 – 586. This function retrieves the ID of an AprilTag found by CameraCaptureImage().

9.24 **CameraGetAprilTagNum**

(AprilTagID As Short, ByRef AprilTagNum As Byte) As Short

Parameters:

AprilTagID—Specifies the ID encoded into the desired AprilTag

AprilTagNum— Retrieves the index associated with the specified AprilTagID.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Determines if a specific AprilTag ID is present in the AprilTags found by calling CameraCaptureImage().

9.25 **CameraGetAprilTagPosition**

(AprilTagNum As Byte, AprilTagWidthMM As Double, ByRef AprilTagCoord As t2dCoord, ByRef mmPerPixel As Double, ByRef AprilTagAngle As Double) As Short

Parameters:

AprilTagNum—Specifies the index of the AprilTag. For example, if CameraCaptureImage() returns AprilTagCount = 2, then AprilTagNum should be set to either 0 or 1

AprilTagWidthMM—Specifies the physical width of the AprilTag, in millimeters. When printing new AprilTags for use with the integrated vision system, the recommended width is 25mm

AprilTagCoord—Retrieves the coordinates (x,y) of the center of the AprilTag in units of camera pixels relative to the center of the camera image

mmPerPixel—Retrieves the factor for converting pixel coordinates to millimeters

AprilTagAngle--Retrieves the angle of the AprilTag in degrees, relative to the camera. Counterclockwise is positive

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Retrieves the location of the center of the AprilTag and the rotation angle within the image captured by calling CameraCaptureImage().

9.26 **CameraGetImage, *overload 1***

(FrameHeight As UShort, Resolution As UShort, HorizontalOffset As Short, VerticalOffset As Short, JPEGQuality As Byte, JPEGResolution As eCameraResolution, ByRef Img() As Byte) As Short

Parameters:

FrameHeight—Specifies the vertical size, in pixels, of a frame to crop out within the specified Resolution. The frame width will be calculated at a 4:3 ratio.

Resolution—Specifies the vertical line count, in pixels, of the full-sized image prior to cropping based on FrameHeight. Used only if JPEGQuality = 0 (bitmap).

HorizontalOffset—Shifts the horizontal position of the cropped frame relative to the center of the image

Vertical Offset—Shifts the vertical position of the cropped frame relative to the center of the image

JPEGQuality—If set to 0, then the image is captured as a bitmap. If Set to 1-100, then the image is captured as a JPEG with the corresponding quality percent

JPEGResolution—If JPEGQuality > 0, then this parameter specifies the resolution via the eCameraResolution enum that specifies the camera image format (VGA = 11 WQXGA2 = 38, etc.)

Img()—Retrieves an array containing the pixel grayscale values of the image

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Commands the integrated camera to capture an image in the specified format. The image is then transferred to the PC via the robot CAN bus. The pixel values are returned as a byte array.

9.27 **CameraGetImage, *overload 2***

(FrameHeight As UShort, VerticalResolution As UShort, HorizontalOffset As Short, VerticalOffset As Short, ByRef bm As System.Drawing.Bitmap) As Short

Parameters:

FrameHeight—Specifies the vertical size, in pixels, of a frame to crop out within the specified Resolution. The frame width will be calculated at a 4:3 ratio.

Resolution—Specifies the vertical line count, in pixels, of the full-sized image prior to cropping based on FrameHeight.

HorizontalOffset—Shifts the horizontal position of the cropped frame relative to the center of the image

Vertical Offset—Shifts the vertical position of the cropped frame relative to the center of the image

bm—Retrieves the image as a System.Drawing.Bitmap object

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Commands the integrated camera to capture an image in the specified format. The image is then transferred to the PC via the robot CAN bus. The image is returned as a bitmap.

9.28 **CameraGetImage, *overload 3***

(FrameHeight As UShort, HorizontalOffset As Short, VerticalOffset As Short, JPEGQuality As Byte, JPEGResolution As eCameraResolution, ByRef jpg As System.Drawing.Image) As Short

Parameters:

FrameHeight—Specifies the vertical size, in pixels, of a frame to crop out within the specified Resolution. The frame width will be calculated at a 4:3 ratio.

HorizontalOffset—Shifts the horizontal position of the cropped frame relative to the center of the image

Vertical Offset—Shifts the vertical position of the cropped frame relative to the center of the image

JPEGQuality—Specifies the quality percent (1 – 100) of the captured JPEG image

JPEGResolution—Specifies the resolution via the eCameraResolution enum that specifies the camera image format (VGA = 11 WQXGA2 = 38, etc.)

jpg—Retrieves the image as a System.Drawing.Image object

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Commands the integrated camera to capture an image in the specified format. The image is then transferred to the PC via the robot CAN bus. The image is returned as a JPEG.

9.29 **CameraGetMaximumResolution**

(ByRef HorizontalResolution As Short, ByRef VerticalResolution As Short) As Short

Parameters:

HorizontalResolution—Retrieves the maximum native horizontal resolution of the camera. The standard value is 2592 pixels

VerticalResolution—Retrieves the maximum native vertical resolution of the camera. The standard value is 1944 pixels

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Retrieves the maximum horizontal and vertical native resolution of the integrated camera.

9.30 **CameraGetWindowing**

(ByRef HorizontalWindowing As Short, ByRef VerticalWindowing As Short) As Short

Parameters:

HorizontalWindowing—Retrieves the horizontal size of the cropped window used by CameraCaptureImage(). The standard value is 240 pixels

VerticalWindowing— Retrieves the vertical size of the cropped window used by CameraCaptureImage(). The standard value is 260 pixels

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Retrieves the horizontal and vertical size, in pixels, of the cropped window used by the CameraCaptureImage() function for the integrated camera. This is the maximum window size allowed by the camera for detecting AprilTags.

9.31 CameraSetExposureTime

(ExposureMS As UShort) As Short

Parameter:

ExposureMS—Specifies the exposure time in milliseconds

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Sets the exposure time used by the integrated camera when capturing or getting images. This value may need to be adjusted, based on ambient light levels, to improve the ability of the camera to detect AprilTags. The default value is 200ms. This setting is volatile, so it will return to the default value when robot power is cycled.

9.32 CalculateMoveAbsAllAxes

(CmdPos() As Double, CmdVel As Double, CmdAccel As Double, EncPos() As Double, EncVel() As Double, EncAccel() As Double, CalculatedMoveTimeMsec As Integer, SkipAx() As Boolean, StartPos() As Double, UseCurrPos As Boolean, EncMoveDist() As Double) As Short

Parameters:

CmdPos()—Specifies the absolute commanded axis positions for the move, expressed in millimeters and degrees.

CmdVel—Specifies the velocity for the move, expressed in percentage of maximum.

CmdAccel—Specifies the acceleration for the move, expressed in percentage of maximum.

EncPos()—Returned containing CmdPos values converted to encoder counts.

EncVel()—Returned containing CmdVel converted to encoder counts per second.

EncAccel()—Returned containing CmdAccel converted to encoder counts per second per second.

CalculatedMoveTimeMsec—Returned containing the expected duration of the specified move.

SkipAx()—Returns a list of which axes will be moving. A value of “True” is returned for an axis when CmdPos equals StartPos, or if UseCurrPos equals “True” and CmdPos equals the current position of the axis.

StartPos()—Specifies the starting position of the robot if UseCurrPos is set to “False”.

UseCurrPos—A “False” value specifies that the values in StartPos should be used in calculating EncVel and EncAccel instead of the current position of the robot.

EncMoveDist()—Returns the relative move distance, in encoder counts, of each axis.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

MoveAbsoluteAllAxes uses this function to calculate velocities and accelerations for each motor so that the motors will complete their motions simultaneously. The motion profile can either start from the current position of the robot or from a specified future position.

9.33 ConfigureInputLogic

(Axis As Short, InputNumber As Short, LogicHigh As Boolean, LogicType As eInputLogic) As Short

Parameters:

Axis—Specifies the motor drive containing the input to be configured.

InputNumber—Specifies the number of the input (1-6) to be configured.

LogicHigh—Specifies the input state at which the action specified by LogicType will occur (High=True=Activated, Low=False=Deactivated)

LogicType—Specifies the action to take place when the input is in the state specified by LogicHigh. Available actions are as follows:

- Freewheel = 0
- SoftAndAuxStop = 2
- None = 4
- GeneralPurpose = 6
- EnableForwardOnly = 8 (used by RemovePlateFromStack)
- EnableReverseOnly = 10
- BeginMotion = 12

- SoftStop = 14
- EnableMainHomeSequence = 16
- EnableAuxHomeSequence = 18
- StopUnderControl = 20
- AbortMotion = 22

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function configures a digital input to carry out a motor command automatically when the input is tripped. An example is the ability of the RemovePlateFromStack function to stop the Z axis when using the gripper sensor to scan for the top of the plate stack.

9.34 ConvertCartesianToJointPosition

(Coordinate() As Double, JointPosition() As Double)

Parameters:

Coordinate()—Cartesian coordinate representing the position of the robot.
JointPosition()—This parameter returns the joint positions equal to the coordinate position passed in.

Description:

Converts a Cartesian coordinate (X,Y,Z,Theta) to the corresponding joint positions.

9.35 ConvertCartesianToolOffsetToWorldCoordinates

(StartCoordinate() As Double, ToolOffset() As Double, WorldCoordinates() As Double, RefFrameRotate As Double, ToolOffset As Double) As Short

Parameters:

StartCoordinate()—World Cartesian coordinate representing the initial position of the robot.

ToolOffset()—Specifies the distance from StartCoordinate to offset the robot position, in the tool Cartesian reference frame defined by StartCoordinate. ToolOffset is defined as follows:

Coordinate 0 = Tool X Position (gripper left to gripper right)
Coordinate 1 = Tool Y Position (gripper front to gripper back)
Coordinate 2 = Tool Z Position (up and down)
Coordinate 3 = Tool Theta (wrist rotation)

WorldCoordinates()—Returns the world Cartesian coordinates generated by applying ToolOffset to StartCoordinate.

RefFrameRotate—Specifies a rotation, in degrees, for StartCoordinate in the world coordinate system.

ToolOffset—Specifies a linear tool offset for StartCoordinate in the world coordinate system.

Description:

This function is used to offset a world Cartesian coordinate by a desired tool coordinate amount. This can be used for adjusting the coordinates of a group of teachpoints that need to be translated or rotated as a group.

9.36 ConvertElbowAngleToPosition

(ElbowAngle As Double, ElbowPos As Double)

Parameters:

ElbowAngle— Specifies the rotary position of the elbow actuator that corresponds to a linear position of the arm. Expressed in units of degrees, and ranges from -4 to 180.

ElbowPos—Returns the linear amount the arm will be extended away from its fully retracted position, and ranges from 0 to 525. Specified in units of millimeters.

Description:

This function is used to convert the rotary position of the elbow actuator into the linear extension position of the arm.

9.37 ConvertElbowPositionToAngle

(ElbowPos As Double, ElbowAngle As Double)

Parameters:

ElbowPos— This value must be passed in. Specified in units of millimeters. It is the linear amount the arm is extended away from its fully retracted position.

ElbowAngle—Returns a value calculated from the value of ElbowPos. Indicates the rotary position of the elbow actuator that corresponds to a linear position of the arm. Expressed in units of degrees.

Description:

This function is used to convert the linear extension position of the arm into the rotary position of the elbow actuator.

9.38 ConvertEncoderToJointPositions

(EncoderPositions() As Double, JointPositions() As Double)

Parameters:

EncoderPositions()—Specifies the encoder positions of the robot motors.

JointPositions()—Returns the joint positions equal to the encoder positions passed in.

Description:

Converts encoder positions to the corresponding joint positions.

9.39 ConvertJointPositionToCartesian, *overload 1*

(JointPosition() As Double, Coordinate() As Double)

Parameters:

JointPosition()—Specifies the joint positions of the robot.

Coordinate()—Returns the Cartesian coordinate equal to the joint positions passed in.

Description:

Converts joint positions to the corresponding Cartesian coordinate (X,Y,Z,Theta).

9.40 ConvertJointPositionToCartesian, *overload 2*

(JointPosition() As Double, Coordinate() As Double, WristOffset As Double)

Parameters:

JointPosition()—Specifies the joint positions of the robot.

Coordinate()—Returns the Cartesian coordinate equal to the joint positions passed in.

WristOffset—Specifies the length of the wrist housing, in millimeters. Measured from the center of the last pulley in the arm links to the center of the wrist joint. The default value is 145.0mm. The integrated camera is located at 98.89mm

Description:

Converts joint positions to the corresponding Cartesian coordinate (X,Y,Z,Theta). This overload provides an extra parameter to specify the wrist offset.

9.41 ConvertJointPositionToToolCoordinate

(JointPosition() As Double, RefFrameRotate As Double, ToolOffset As Double, ToolCoordinate() As Double)

Parameters:

JointPosition()—Specifies the joint positions of the robot.

RefFrameRotate—Compensates for a gripper that does not point straight forward when the wrist is at the 0.0 position. Specified in degrees.

ToolOffset—Specifies the distance from the center of the wrist axis to the center of the object being handled by the gripper.

ToolCoordinate()—Returns the Cartesian coordinate of the point that results from applying the RefFrameRotate and ToolOffset values.

Description:

Converts joint positions to the corresponding Cartesian coordinate of a point located at the position in space that results from applying the RefFramRotate and ToolOffset values.

9.42 ConvertJointToEncoderPositions

(JointPositions() As Double, EncoderPositions() As Double)

Parameters:

JointPositions()—Specifies the joint positions of the robot.

EncoderPositions()—Returns the encoder positions equal to the joint positions passed in.

Description:

Converts joint positions to the corresponding encoder positions.

9.43 ConvertTeachPointToCartesian

(TeachPoint As String, Coordinate() As Double) As Short

Parameters:

TeachPoint—Specifies the name of the Teach Point whose value will be converted to a Cartesian coordinate.

Coordinate()—Returns the value of the Cartesian coordinate corresponding to the Teach Point specified.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function calculates the Cartesian coordinate value of a Teach Point.

9.44 ConvertToolCoordinateToJointPosition

(ToolCoordinate() As Double, RefFrameRotate As Double, ToolOffset As Double, JointPosition() As Double)

Parameters:

ToolCoordinate()—Specifies the Cartesian coordinate of a point in space that represents the center of the object being handled by the gripper.

RefFrameRotate—Compensates for a gripper that does not point straight forward when the wrist is at the 0.0 position. Specified in degrees.

ToolOffset—Specifies the distance from the center of the wrist axis to the center of the object being handled by the gripper.

JointPosition()—Returns the joint positions of the robot that result from applying the RefFrameRotate and ToolOffset values to ToolCoordinate.

Description:

Converts the Cartesian coordinate of a point located at the position in space that results from applying the RefFramRotate and ToolOffset values.

9.45 CycleCountGet

(CycleCount As Integer) As Short

Parameters:

CycleCount—Returns the current cycle count (total number of moves) for the robot.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function retrieves the “Move Count” value stored in the robot parameter file under the [Cycle Data] section. This value represents the total number of moves executed by the robot.

9.46 DelayMsec

(MSec As Integer)

Parameters:

Msec—Specifies the number of milliseconds to pause.

Description:

Pauses execution for the number of milliseconds specified by Msec. Events that occur during the pause will not be lost. The amount of time paused is based on the system clock (absolute, real-time), not on the calling application's processor time. For this reason, it is not recommended that this function be used for measuring robot communication or motion timeout periods.

9.47 DetermineTeachPoint

(TeachPoint As String, Optional Position() As Double, Optional Tolerance As Double)

Parameters:

TeachPoint—Returns the name of the teachpoint that corresponds to Position.

Position()—Specifies the joint position being evaluated. If this parameter is passed in with a value of "Nothing", then the current position of the robot will be used.

Tolerance—Specifies how close to a teachpoint Position() must be. If omitted, a value of 0.1 (degrees or mm) will be used.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Determines whether a teachpoint exists that is equal to the joint values specified by Position() +/- the specified tolerance. This function can also be used to determine if the robot is currently positioned at a teachpoint.

9.48 EmergencyStop

Description:

Attempts to stop motion on all axes and attempts to disable the motors. To restart the robot after calling this subroutine, call the "Initialize" function. Please note that this is not to be used as a safety device, as electricity is not shut off to the motors, and this subroutine may fail, or fail to be called, depending on the state of the PC and controller. There is a separate hard-wired mechanical emergency stop button built into the system that, when pressed, disables the motors. This subroutine is provided merely as a matter of convenience; it gives the operator a quick way of stopping the robot, without killing power, if the wrong command has been sent to the robot. .

9.49 EnableVisionOffset

(ByVal enabled As Boolean) As Short

Parameter:

enabled – If set to True, then the offset specified by SetVisionOffset will be applied to all multi-axis moves that are commanded via MotorsMoveJoint or MotorsMoveLinear, or by any parent function of these functions, unless the IgnoreVisionOffset parameter of these functions is set to True.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Enables or disables the positional offset specified by SetVisionOffset.

9.50 GetAnalogInputAssignments

As String(,)

Return Value:

Returns a two-dimensional array (first index specifies an axis, second index specifies the digital input number) containing a list of the analog input assignments.

Description:

Retrieves a list of the assignments for the analog inputs. Each axis has two analog inputs. The assignment value (name) will be displayed in the Teach Pendant Inputs/Outputs tab, in the list of analog inputs.

9.51 GetAvailableCANDevices

(ByRef DeviceCode() As Byte, ByRef DeviceDescription() As String)

Parameters:

DeviceCode()—Returns an array containing the codes of the available devices.

DeviceDescription()—Returns an array containing the descriptions of the available devices. Use these values with SetCANDevice.

Description:

This method retrieves the available CAN/USB devices (internal to the robot) that currently are connected to the PC.

9.52 GetBarcodeReaderSerialPort

As Byte

Return Value:

Returns the port number.

Description:

This method retrieves the PC COM port number currently set to be used for the robot barcode reader located in the gripper.

9.53 GetBaseToGripperClearance

(ZClearance As Double, ArmClearance As Double)

Parameters:

ZClearance—Returns the current value of Z Clearance, which is stored in UF[6] in the shoulder motor drive. This value limits how close the Z axis can get to the bottom end of travel when the arm is retracted to within the distance specified by Arm Clearance. The default value is 0.

ArmClearance—Returns the current value of Arm Clearance, which is stored in UF[7] in the shoulder motor drive. This value limits how close the arm can get to its fully retracted position when the Z axis is within the distance specified by ZClearance to the bottom end of travel. The default value is 0.

Description:

This method retrieves the current values of Z Clearance and Arm Clearance. Z Clearance and Arm Clearance define a rectangular region where the arm and Z are not allowed to go. This prevents the gripper or an object in the gripper from colliding with the robot base. The values are stored in the shoulder motor drive, and can be accessed via the Teach Pendant Tools/Terminal window by sending UF[6].1 and UF[7].1.

9.54 GetBuzzerAxis

As Short

Return Value:

Returns the axis number of the motor controller that controls the buzzer.

Description:

This function returns the axis number of the motor controller that controls the buzzer with a digital output (use GetBuzzerOutput to retrieve the output number). The default value is axis 4 (Wrist). It is recommended to use this function instead of hard coding an axis number. This approach will help accommodate future changes to the robot design.

9.55 GetBuzzerOutput

As Short

Return Value:
Returns the input number that controls the buzzer.

Description:

This function returns the input number that controls the buzzer (use GetBuzzerAxis to retrieve the axis number). The default value is output 4. It is recommended to use this function instead of hard coding an output number. This approach will help accommodate future changes to the robot design.

9.56 GetCameraPresent

As Boolean

Return Value:
Will return a value of True if the robot has an integrated camera, or False if camera is absent

Description:
This function is used to determine the presence or absence of an integrated camera. The robot must be initialized prior to calling this function.

9.57 GetCameraWristOffset()

As Double

Return Value:
Will return the distance from the rear of the wrist housing to the center of the integrated camera

Description:
This function is used to retrieve the physical location of the integrated camera inside the robot wrist housing. The robot must be initialized prior to calling this function. The default value is 98.89mm. This value is stored in the wrist motor drive as UF[6].

9.58 GetCANDevice

As String

Return Value:
Returns the description of the currently selected CAN device.

Description:
This method retrieves the CAN/USB device (internal to the robot) currently selected.

9.59 GetDefaultErrorLogFile

As String

Return Value:
Will return the directory and file name for the default error log file.

Description:
This method retrieves the current setting for the default error log file from the registry.

9.60 GetDefaultGripperShutdownState

As Boolean

Return Value:
Will return a value of True if the default gripper shutdown state is open, or False if the default state is closed.

Description:
This method retrieves the current setting for the default gripper shutdown state from the registry, as set by the SetDefaultGripperShutdownState method. If set to True, the gripper will be opened when the ShutDown function is called. Otherwise the gripper will be closed. Call UseDefaultGripperShutdownState(False) to leave gripper in its current position.

9.61 GetDefaultGripperState

As Boolean

Return Value:
Will return a value of True if the default gripper state is open, or False if the default state is closed.

Description:
This method retrieves the current setting for the default gripper state from the registry, as set by the SetDefaultGripperState method. If set to True, the gripper will be opened after the gripper is homed during robot

initialization. Otherwise, the gripper will be closed. Call UseDefaultGripperState(False) to leave gripper in its current position.

9.62 GetDefaultSequenceFile

As String

Return Value:
Will return the directory and file name of the default sequence file.

Description:
This method retrieves the file path and name of the default sequence file from the registry.

9.63 GetDefaultServoGripperClosedPosition()

As Double

Return Value:
Will return the default closed position, in millimeters, of the servo gripper.

Description:
This method retrieves the current setting for the default servo gripper closed position from the registry.

9.64 GetDefaultServoGripperOpenPosition()

As Double

Return Value:
Will return the default open position, in millimeters, of the servo gripper.

Description:
This method retrieves the current setting for the default servo gripper open position from the registry.

9.65 GetDefaultTeachpointFile

As String

Return Value:
Will return the directory and file name for the default teachpoint file.

Description:
This method retrieves the current setting for the default teachpoint file from the registry.

9.66 GetDefaultVisionCalibrationFile

As String

Return Value:
Will return the directory and file name for the default vision calibration file.

Description:
This method retrieves the current setting for the default vision calibration file from the registry. The vision calibration file stores AprilTag calibration data used by the integrated camera.

9.67 GetDigitalInputAssignments

As String(,)

Return Value:
Returns a two-dimensional array (first index specifies an axis, second index specifies the digital input number.

Description:
Retrieves a list of the assignments for the digital inputs. Each axis has six digital inputs. The assignments are listed in the robot parameter file, and can be modified, if desired (user inputs only). The assignment value (name) will be displayed in the Teach Pendant Inputs/Outputs tab, in the list of digital inputs.

9.68 GetElbowAxisNumber

As Short

Return Value:
Returns the axis number for the elbow.

Description:
Retrieves the axis number for the elbow. The default elbow axis number is 3, but this function provides a method of ensuring compatibility with future robots that may have the axes in a different order.

9.69 GetErrorCode

(ErrorNumber As Short) As String

Parameters:
ErrorNumber—The number of an error from the errors listed above.

Return Value:
Will return the text description of the error number.

Description:
Provides a method of retrieving the text of an error message. If a function returns an error code, the error code can be passed into this function to retrieve the corresponding text description of the error.

9.70 GetEStopState

(State As eEStopState) As Short

Parameters:
State—Returns the state of the E-Stop circuitry. The following is a list of the possible states:
1 – Normal
2 – Estop

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs. If the robot is in an E-Stop state, then one of the following error codes will be returned:

298—Both e-stop inputs are activated
301—STO1=High, STO2=Low
302—STO1=Low, STO2=High

Description:
Retrieves the state of the E-Stop circuitry.

9.71 GetEthernetPort

() As Integer

Return Value:
Will return the port number specified for the Ethernet interface.

Description:
Retrieves the Ethernet port number specified for use with the CAN/Ethernet gateway. The robot hardware must match the specified interface type.

9.72 GetGripperSensorAxis

As Short

Return Value:
Returns the digital input number on the axis that has an input connected to the gripper sensor.

Description:
This method determines the digital input number on the axis that is monitoring the gripper sensor (default = axis 2).

9.73 GetGripperSensorInput

As Short

Return Value:
Returns the axis that has an input connected to the gripper sensor.

Description:
This method determines the axis that is monitoring the gripper sensor (default = input 4).

9.74 GetHeadersFromINIFile

(Filename As String, Header() As String) As Short

Parameters:
Filename—Specifies the full file path and filename for the INI file to be read.

Header()—Returns the names of all headers in the INI file.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Retrieves a list of all section headers in an INI file.

9.75 GetIndicatorLightAxis

As Short

Return Value:
Returns the axis that has an output controlling the indicator light.

Description:
This method determines the axis that is controlling the indicator light on the Teach button.

9.76 GetIndicatorLightBlueAxis

As Short

Return Value:
Returns the axis that has an output controlling the blue indicator light.

Description:
This method determines the digital output number on the axis that is controlling the blue circuit of the RGB indicator light.

9.77 GetIndicatorLightBlueOutput

As Short

Return Value:
Returns the output number that is controlling to the blue indicator light.

Description:
This method determines the output is controlling the blue circuit of the RGB indicator light.

9.78 GetIndicatorLightGreenAxis

As Short

Return Value:
Returns the axis that has an output controlling the green indicator light.

Description:
This method determines the digital output number on the axis that is controlling the green circuit of the RGB indicator light.

9.79 GetIndicatorLightGreenOutput

As Short

Return Value:
Returns the output number that is controlling to the green indicator light.

Description:
This method determines the output is controlling the green circuit of the RGB indicator light.

9.80 GetIndicatorLightOutput

As Short

Return Value:
Returns the output number that is controlling the indicator light.

Description:
This method determines the output that is controlling the indicator light on the teach button.

9.81 GetIndicatorLightRedAxis

As Short

Return Value:
Returns the axis that has an output controlling the red indicator light.

Description:
This method determines the digital output number on the axis that is controlling the red circuit of the RGB indicator light.

9.82 GetIndicatorLightRedOutput

As Short

Return Value:
Returns the output number that is controlling to the red indicator light.

Description:

This method determines the output is controlling the red circuit of the RGB indicator light.

9.83 GetInterfaceType

As eInterfaceType

Return Value:

One of the following values will be returned:

USB—A CAN/USB adapter is specified

Ethernet—A CAN/Ethernet gateway is specified

Description:

Returns the interface type being used by the PC to communicate with the robot. The robot hardware must match the interface type setting.

9.84 GetJointMoveDirection

(Axis As Short) As eJointMoveDirection

Parameter:

Axis—The robot axis for which the direction setting is being queried.

Return Value:

One of the following direction values will be returned:

Normal—The axis will move the direction that does not require passing through the 0°/360° rollover point

Clockwise—The axis will always move in the clockwise (negative) direction

Counterclockwise—The axis will always move in the counterclockwise (positive) direction

ShortestWay—The axis will always move the shortest direction, even if it must pass through the 0°/360° rollover point

Description:

Returns the move direction for an axis that has unlimited travel, such as the Shoulder or Wrist.

9.85 GetMaintainGripAfterShutdown

As Boolean

Return Value:
Will return a value of True if the gripper motor will be left enabled after shutdown, or False if the motor will be disabled.

Description:
This method retrieves the current setting for the post-shutdown gripper motor state from the registry, as set by the SetMaintainGripAfterShutdown method. If set to True, the gripper motor will be left in an energized state after the ShutDown function is called. Otherwise, the gripper motor will be de-energized.

9.86 GetMasterVelocityScale

() As Double

Return Value:
Returns a value between 0 and 100.

Description:
Will return the current value of the Master Velocity Scale in the registry. The Master Velocity Scale is used to scale down all commanded velocities for testing purposes.

9.87 GetMotorLimits

(MaxTravel() As Double, MinTravel() As Double, MaxVel() As Double, MaxAccel() As Double) As Short

Parameters:
MaxTravel()—Returns the maximum travel limits of the motors. Expressed in degrees (rotary) or millimeters (linear).

MinTravel()—Returns the minimum travel limits of the four motors. Expressed in degrees (rotary) or millimeters (linear).

MaxVel()—Returns the maximum velocities of the four motors. Expressed in degrees per second (rotary) or millimeters per second (linear).

MaxAccel()—Returns the maximum accelerations of the four motors. Expressed in degrees per second per second (rotary) or millimeters per second per second (linear).

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Retrieves absolute maximum settings for position, velocity, and acceleration of the four motors, as specified in the robot parameter file.

9.88 GetMovePathMode

As eMovePathMode

Return Value:
eMovePathMode —The following is a list of possible values:
0 – Joint
1 – Linear

Description:

This method returns the setting that controls whether the robot will execute joint or linear moves when commanded via the MoveAbsoluteAllAxes method, as well as other methods that use the MoveAbsoluteAllAxes, like the stack and hotel methods.

9.89 GetNumAxes

() As Short

Return Value:
Will return 4 if RobotOnRail = False, or 5 if RobotOnRail = True.

Description:
Can be used for determining the total number of robot axes, which typically is 4, but will be 5 if the robot is mounted on a rail. The gripper is not counted as an axis.

9.90 GetOutputAssignments

As String(,)

Return Value:
Returns a two-dimensional array (first index specifies an axis, second index specifies the output number).

Description:

Retrieves a list of the assignments for the outputs. Each axis has four digital outputs. The assignment value (name) will be displayed in the Teach Pendant Inputs/Outputs tab, in the list of digital outputs.

9.91 GetRailAxisNumber

As Short

Return Value:
Returns the axis number for the rail.

Description:
Retrieves the axis number for the rail. The default axis number for the rail is 5. This function provides a method of ensuring compatibility with future robots that may have the axes in a different order.

9.92 GetRegistryAppName

() As String

Return Value:
Will return “KX2 Robot Control“ + value of RegSerialNumber, which was specified by SetRobotSerialNumber().

Description:
Retrieves the value of the App name that will be used in the Windows Registry.

9.93 GetRegistrySerialNumber

() As String

Return Value:
Will return the value of RegSerialNumber, which was specified by SetRobotSerialNumber().

Description:
Retrieves the value of the robot serial number that will be used in the Windows Registry.

9.94 GetRobotIPAddress

() As String

Return Value:
Will return the IP address specified for the Ethernet interface.

Description:
Retrieves the IP address specified for use with the CAN/Ethernet gateway. The robot hardware must match the specified interface type.

9.95 GetSaveMoveDataOnShutDown

() As Boolean

Return Value:
Will return True if setting is enabled. Otherwise, False is returned.

Description:
Retrieves the value of the SaveMoveDataOnShutDown registry setting that controls whether move cycle data will be saved to the motor drives when the ShutDown function is called.

9.96 GetServoGripperAxisNumber

() As Short

Return Value:
Will return the axis number of the servo gripper.

Description:
Retrieves the axis number of the servo gripper. The default axis number for the servo gripper is 6. Earlier KX-2 robots built prior to the release of the KX-2 rail had the gripper axis number set to 5.

9.97 GetServoGripperLimits

(MaxTravel() As Double, MinTravel() As Double, MaxVel() As Double, MaxAccel As Double) As Short

Parameters:
MaxTravel—Returns the maximum travel limit of the servo gripper motor. Expressed in millimeters (linear).

MinTravel—Returns the minimum travel limits of the servo gripper motor. Expressed in millimeters (linear).

MaxVel—Returns the maximum velocity of the servo gripper motor. Expressed in millimeters per second (linear).

MaxAccel—Returns the maximum acceleration of the servo gripper motor. Expressed in millimeters per second per second (linear).

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Retrieves absolute maximum settings for position, velocity, and acceleration of the servo gripper motor.

9.98 GetShoulderAxisNumber

As Short

Return Value:
Returns the axis number for the shoulder.

Description:
Retrieves the axis number for the shoulder. The shoulder is usually axis 1. This function provides a method of ensuring compatibility with future robots that may have the axes in a different order.

9.99 GetSubWindowMode

() As Short

Return Value:
0 = Modeless (allows switching freely between multiple windows)
1 = Modal (sub-window must be closed to access parent window)

Description:
Retrieves the value of ModelessSubWindows from the Windows Registry. The value can be changed using SetSubWindowMode() or by selecting “Modeless Sub-Windows” in the Teach Pendant/Tools/Miscellaneous window.

9.100 GetTeachButtonAxis

As Short

Return Value:
Returns the digital input number on the axis that has an input connected to the teach button.

Description:
This method determines the digital input number on the axis that is monitoring the teach button (default = axis 3).

9.101 GetTeachButtonInput

As Short

Return Value:
Returns the input that is connected to the teach button.

Description:
This method determines the input that is monitoring the teach button (default = input 6).

9.102 GetTeachPendantAccessLevel

() As eTeachPendantAccessLevel

Return Value:
Full = 0
Technician = 1
Limited = 2

Description:
Retrieves the current access level for the Teach Pendant, as specified in the Windows Registry.

9.103 GetTeachPendantAccessPassword

(AccessLevel As eTeachPendantAccessLevel) As String

Parameter:
AccessLevel—Specifies the access level for which to retrieve the password.
Full = 0
Technician = 1
Note: Limited access has no password.

Return Value:
Returns the decrypted password.

Description:
Retrieves the current password for the specified access level for the Teach Pendant, as specified in the Windows Registry, but decrypted.

9.104 GetUsingDefaultGripperShutdownState

As Boolean

Return Value:
Returns “True” if gripper will be moved to the default state during Shutdown. Otherwise, “False” is returned.

Description:
Determines whether gripper is configured to move to the default shutdown state during Shutdown, as specified in the Windows Registry.

9.105 GetUsingDefaultGripperState

As Boolean

Return Value:
Returns “True” if gripper will be moved to the default state during initialization. Otherwise, “False” is returned.

Description:
Determines whether gripper is configured to move to the default state during initialization, as specified in the Windows Registry.

9.106 GetVersionNumber

(Optional ByRef VersionDate As String) As String

Parameters:
VersionDate—Returns the release date for the DLL.

Return Value:
Returns the version number for the DLL.

Description:
Reports the release date and the version number for the DLL.

9.107 GetVisionOffsetEnabled

(Optional ByRef name As String = “”) As Boolean

Parameter:
name – Returns the value of the name specified for the vision offset via SetVisionOffset

Return Value:
Will return True if the vision offset is enabled.

Description:
Retrieves the enabled state of the vision offset that was last set using EnableVisionOffset and provides the name of the current vision offset.

9.108 GetWarningAfterIdleTime

As Long

Return Value:
Returns the idle time setting, in seconds.

Description:
Returns the number of seconds of idle time required to activate a warning (buzzer beeps, indicator light flashes) at the beginning of the next move.

9.109 GetWarningBeforeMove

As Boolean

Return Value:
Returns True if activated or False if not.

Description:
Retrieves the state of the WarningBeforeMove setting. If activated, the buzzer beeps and the indicator light flashes at the beginning of a move that is executed after the robot has sat idle for a time period equal to or greater than WarningAfterIdleTime.

9.110 GetWarningDuration

As Short

Return Value:
Returns the warning duration, in seconds.

Description:
Retrieves the duration of buzzer beeping and indicator light flashing at the beginning of a move that is executed after the robot has sat idle.

9.111 GetWristAxisNumber

As Short

Return Value:
Returns the axis number for the wrist.

Description:
Retrieves the axis number for the wrist. The wrist is usually axis 4. This function provides a method of ensuring compatibility with future robots that may have the axes in a different order.

9.112 GetZAxisNumber

As Short

Return Value:
Returns the axis number for the Z axis.

Description:
Retrieves the axis number for the Z axis. The Z axis is usually axis 2. This function provides a method of ensuring compatibility with future robots that may have the axes in a different order.

9.113 HomeMotor

(Axis As Short) As Short

Parameters:
Axis—Specifies motor to be homed.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This method is used primarily for homing the servo gripper since it has an incremental encoder that loses its position reference when power is cycled. The gripper will move to the fully open hard stop, then search for the index pulse, and then move to the home position. For any axis other than the servo gripper, the axis will move to the zero position at 10% velocity and acceleration. The main robot axes have absolute encoders, so homing is not required. If a specific robot orientation is required at startup, use MoveAbsoluteSingleAxis() to position each robot joint individually, as needed.

9.114 Initialize

(Optional HomeGripper As Boolean = True, Optional SkipHomeIfAlreadyHomed As Boolean = True, Optional UseCachedDriveParams As Boolean = False, Optional ByRef CachedDriveParams As tDriveParam) As Short

Parameters:

HomeGripper—This optional parameter specifies whether the gripper is to be homed. The default value is "true". If the gripper was previously homed successfully and power has not been turned off since then, and SkipHomeIfAlreadyHomed=True, then homing will be skipped even if HomeGripper=True.

SkipHomeIfAlreadyHomed—This optional parameter specifies whether the gripper is to be homed again, even if it has already been homed. When the power to the robot is turned off, the gripper encoder loses its position, which makes homing a requirement. If the gripper has already been homed, and power has not been cycled since then, the gripper encoder will retain its position, and homing will not be required. If this parameter is set to “true”, the application will check if the encoder position has been reset due to power cycling. If the encoder position has not been reset, then the gripper will not be homed unnecessarily. If the encoder position has been reset, then the gripper will be homed.

HomeGripper	SkipHomeIfAlreadyHomed	Action
True	True	Home only if needed
True	False	Home always
False	True or False	Home never

UseCachedDriveParams—This optional parameter specifies whether cached drive parameter values will be used in place of querying the values from the motor drives. Using this mechanism speeds up initialization significantly. If drive parameters have not been cached previously, then values in the drive parameters will be queried and cached the first time Initialize() is called. There are two options for caching. The first option is accessed by managing and providing cached values via the CachedDriveParams parameter. The section option is to pass in an empty CachedDriveParams parameter, which directs Initialize() to store and retrieve cached parameters from the Windows Registry.

CachedDriveParams—This optional parameter will return a set of motor drive parameter values that are queried during initialization if UseCachedDriveParams is set to False. This set of values can then be passed back in on subsequent Initialize() calls to speed up the initialization process. The initialization process can be further shortened by setting the Shutdown() DisableMotors parameter to False.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Opens a connection to the robot, reads the TeachPoint file, enables the motors, homes the gripper, establishes a connection with the barcode reader, turns on the blue indicator light, and enables controls on the “Teach Pendant” dialog window. This function must be called before the robot can be operated. An overload is available that excludes UseCachedDriveParams and CachedDriveParams.

9.115 InitializeNoWait

This method is the same as Initialize(), except that it is non-blocking, and runs in the background.

9.116 IsInitialized

() As Boolean

Return Value:

Will return "true" if the robot is initialized, otherwise "false" is returned.

Description:

Determines whether the robot is initialized.

9.117 IsMaintenanceRequired

(Required As Boolean) As Short

Parameter:

Required—Returns a value of “True” if Z maintenance is required. Otherwise, a value of “False” is returned.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Determines whether the Z axis has traveled farther than the maintenance interval since the last time maintenance was performed. If this method returns a “True” value, then the Z axis needs periodic maintenance, which includes greasing the two linear bearings.

9.118 IsRailMaintenanceRequired

(Required As Boolean) As Short

Parameter:

Required—Returns a value of “True” if Rail maintenance is required. Otherwise a value of “False” is returned.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Determines whether the Rail axis has traveled farther than the maintenance interval since the last time maintenance was performed. If this method returns a “True” value, then the Rail axis needs periodic maintenance, which includes greasing the two linear bearing blocks.

9.119 IsRobotOnRail

() As Boolean

Return Value:

Will return True if a Rail is present, or False if not.

Description:

Can be used to determine the presence of a Rail, which is carried out during Initialize() by querying the CAN network for all motor drives present.

9.120 IsScriptRunning

() As Boolean

Return Value:

Will return "true" if a script is running, otherwise "false" is returned.

Description:

Can be used to determine whether a script started with ScriptRun() is still running.

9.121 IsServoGripperPresent

() As Boolean

Return Value:

True—Servo Gripper is present
False—Servo Gripper is not present

Description:

Can be used to determine the presence of a Servo Gripper, which is carried out during Initialize() by querying the CAN network for all motor drives present.

9.122 Jog

(Axis As Short, Vel As Double, Accel As Double, Optional ByRef Index As Byte) As Short

Parameters:

Axis—Specifies the motor to be jogged.

Vel—Specifies the jog velocity, in terms of percent of maximum. A negative value will jog the motor in the negative direction.

Accel—Specifies the jog acceleration in terms of percent of maximum.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteSingleAxisDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Will move the specified axis at the specified velocity and acceleration to the end of travel. The axis can be stopped prematurely using the "JogStop" function. For axes with unlimited rotation, 10 rotations will be commanded.

9.123 JogStop

(Axis As Short) As Short

Parameters:

Axis—Specifies the motor to be stopped.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Stops the axis at the maximum allowed rate of deceleration, and can be called during any single axis move to stop the axis.

9.124 LockOutput

(Locked As Boolean, Axis As Short, OutputNumber As Short, State As Short) As Short

Parameters:

Locked—If set to True, the output will be locked. If set to False, the output will no longer be locked.

Axis—Specifies the axis of the output to be locked.

OutputNumber—Specifies the output to be locked.

State—Specifies the output level at which the output will be locked (1=high, 0=low).

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Locks the state of an output at a desired level. Once an output is locked, the state of the output cannot be changed by other function calls. This is useful for altering the behavior of methods that toggle the state of an output in an unwanted manner. To unlock an output, call this method again, with the value of the “Locked” parameter set to False.

9.125 MaintenancePerformed

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Resets Axis 2 Maintenance Required to “FALSE” (stored in the Z motor drive). Axis 2 Maintenance Required parameter is set to “TRUE” every 100,000 meters of Z-axis travel, at which time maintenance is required (see User’s Manual). The MaintenanceRequired event will be triggered each time the Z axis reaches the maintenance interval.

9.126 MotionRuleCreate

(TeachPoint1 As String, TeachPoint2 As String) As Short

Parameters:

TeachPoint1—Specifies the name of the first teachpoint that will be part of a new motion rule.

TeachPoint1—Specifies the name of the second teachpoint that will be part of a new motion rule.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A motion rule is a pair of teachpoints between which the robot can move safely. Motion rules can be defined under the “Motion Rules” tab of the Teach Pendant. If motion rules are enabled, then a warning message will be displayed when using the Teach Pendant to command the robot to perform a move that violates the motion rules. This method can be used for defining new motion rules. Motion rules are stored in the teachpoints.ini file.

9.127 MotionRuleDelete

(TeachPoint1 As String, TeachPoint2 As String) As Short

Parameters:

TeachPoint1—Specifies the first teachpoint of the motion rule to be deleted.

TeachPoint2—Specifies the second teachpoint of the motion rule to be deleted.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Deletes the motion rule for the two specified teachpoints.

9.128 MotionRulesDeleteTeachPoint

(ByVal TeachPoint As String) As Short

Parameter:

TeachPoint—All motion rules containing this teachpoint name will be deleted.

Return Value:

Will return False if motion rules are disabled, or True if motion rules are enabled.

Description:

Deletes all motion rules that contain the specified teachpoint.

9.129 MotionRulesEnable

(Enabled As Boolean)

Parameters:

Enabled—Specifies whether motion rules will be used or ignored.

Description:

A motion rule is a pair of teachpoints between which it is safe to move. Motion rules can be defined under the “Motion Rules” tab of the Teach Pendant. If motion rules are enabled, then a warning message will be displayed when using the Teach Pendant to command the robot to perform a move that violates the motion rules. Motion rules can be enabled from the Teach Pendant, or by using this method.

9.130 MotionRulesGet

(Rules() As String) As Short

Parameter:

MotionRules()—Returns the list of all motion rules. Each rule is a pair of teachpoint names separated by a comma.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Returns all existing motion rules.

9.131 MotionRulesGetState

() As Boolean

Return Value:

Will return False if motion rules are disabled, or True if motion rules are enabled.

Description:

A motion rule is a pair of teachpoints between which it is safe to move. Motion rules can be defined under the “Motion Rules” tab of the Teach Pendant. If motion rules are enabled, then a warning message will be displayed when using the Teach Pendant to command the robot to perform a move that violates the motion rules. This method can be used to determine whether motion rules are enabled.

9.132 MotionRulesRenameTeachPoint

(CurrentName As String, NewName As String) As Short

Parameters:

CurrentName—Specifies the current name of a teachpoint that is being used in the existing motion rules.

NewName—Specifies the name that will be used for overwriting all occurrences of CurrentName in the existing motion rules.

Return Value:

Will return False if motion rules are disabled, or True if motion rules are enabled.

Description:

Can be used to update all motion rules that contain the name of one teachpoint with the name of another teachpoint.

9.133 MotionRuleVerify

(TeachPoint1 As String, TeachPoint2 As String, PermittedMove As Boolean) As Short

Parameters:

TeachPoint1—Specifies the first teachpoint name in the rule being verified.

TeachPoint2—Specifies the second teachpoint name in the rule being verified.

PermittedMove—Returns whether the motion rule exists.

Return Value:

Will return False if motion rules are disabled, or True if motion rules are enabled.

Description:

Can be used to determine whether a move from one teachpoint to another will violate motion rules. Use DetermineTeachPoint() to determine the starting position if the current position of the robot is unknown.

9.134 MotorCheckIfMoveDone

(Axis As Short, DoneStatus As Boolean) As Short

Parameters:

Axis—Specifies the motor to be checked.

DoneStatus—Return value of “True” if move is done, and “False” if not done..

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function determines if the previous move of the specified axis is completed. This function is to be used only for single axis moves.

9.135 MotorEnable

(Axis As Short, EnableState As Boolean, Optional ByVal LEDIndicatorOn As Boolean = True) As Short

Parameters:

Axis—Specifies the motor to be enabled or disabled.

EnableState—A value of “True” will enable the motor, and a value of “False” will disable the motor.

LEDIndicatorOn—This optional parameter can be set to False to prevent the blue LED indicator light from being turned on.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Enables or disables the specified axis. When disabled, a motor is free and can be moved by hand.

9.136 MotorGetCurrentPosition

(Axis As Short, Position As Double) As Short

Parameters:

Axis—Specifies the motor.

Position—Returns the position of the axis, in millimeters or degrees.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Provides a method for reading the current position of an axis, specified in millimeters or degrees.

9.137 MotorGetHomedStatus

(Axis As Short, Status As Boolean) As Short

Parameters:

Axis—Specifies the motor.

Status—Returns “true” if axis has been homed successfully.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Determines whether the axis has been homed successfully since the last power cycle. This method is to be used with the Servo Gripper only, as the robot axes have absolute encoders and never need to be homed. Robot axes will always return “true”.

9.138 MotorResetEncoderPosition

(Axis As Short, EncoderPosition As Double) As Short

Parameters:

Axis—Specifies the motor.

EncoderPosition—Specifies the new position of the axis, in encoder counts.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function sets the current position of the specified axis to the specified value. This is useful for setting the position of the Servo Gripper manually after the power has been cycled, if it is necessary to operate the gripper before homing it.

9.139 MotorSendCommand

(Axis As Short, MotorCommand As String, Index As Short, Value As String, ValFloat As Boolean, ReturnedData As String) As Short

Parameters:

Axis—Specifies the motor.

MotorCommand—Contains the text of a low-level command for a motor. Refer to Elmo Command Reference for Gold Line Drives.

Index—Used by some array-based commands. Non-array-based commands use a value of -1.

Value—Contains data to accompany the command, such as the velocity value of a speed command.

ValFloat—Specifies whether the Value parameter represents a Float value (“true”) or an Integer value (“false”).

ReturnedData—Returns any information sent by the motor, such as an encoder position or a digital input state. If a command does not return data, an empty string must be passed in. If a command does return data, a non-empty string must be passed in (the value is not important).

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function is used to send low-level commands directly to the motors to achieve custom functionality. This should not be necessary for typical robot control, as the pertinent commands are written into this software.

9.140 MotorsMoveJoint

(CmdPos() As Double, CmdVel As Double, CmdAccel As Double, WaitUntilDone As Boolean, CalculatedMoveTimeMsec As Integer, SendEventWhenMoveDone As Boolean , Optional ByRef Index As Byte, Optional ByVal IgnoreVisionOffset As Boolean = False) As Short

Parameters:

CmdPos()—Specifies the absolute commanded joint position for the move, expressed in millimeters and degrees.

CmdVel—Specifies the velocity for the move, expressed in percentage of maximum.

CmdAccel—Specifies the acceleration for the move, expressed in percentage of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the move command has been loaded into the move buffer. Either the function “MotorWaitForMoveDone” will have to be called once for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped, or the host application can simply wait for the MoveAbsoluteAllAxesDone event.

If this method is called multiple times before the robot has completed the previous moves, it will be loaded into the move buffer and will be executed as soon as the previous moves have finished. Preloading

multiple moves into the move buffer in this manner will allow each subsequent move to be preloaded into the robot, which reduces the pause time between moves.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "MoveAbsoluteAllAxesDone" event will be triggered when the move is done.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

IgnoreVisionOffset—If set to true, then the vision offset specified by SetVisionOffset() will be ignored even if the vision offset has been enabled via EnableVisionOffset().

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to move the robot to a specific absolute position. Each motor will be rotated the minimum required distance to reach the commanded position, thus the end of the arm will not follow a predictable path. It is recommended that the function “TeachPointMoveTo” be used instead of this function, in conjunction with the SetMovePathMode command, since use of this function would require management of axis position values by the host application.

9.141 MotorsMoveLinear

(CmdPos() As Double, CmdVel As Double, CmdAccel As Double, WaitUntilDone As Boolean, Optional CalculatedMoveTimeMsec As Integer = 0, Optional UseLinearVelAccelUnits As Boolean = False, Optional SendEventWhenMoveDone As Boolean = False, Optional MoveToLimit As Boolean = False, Optional RefFrameRotate As Double = 0, Optional ToolOffset As Double = 0, Optional ByRef Index As Byte, Optional ByVal IgnoreVisionOffset as Boolean = False) As Short

Parameters:

CmdPos()—Specifies the absolute commanded joint position for the move, expressed in millimeters and degrees.

CmdVel—Specifies the velocity for the move, expressed in percentage of maximum, unless UseLinearVelAccelUnits = True, in which case the units of CmdVel are mm/sec.

CmdAccel—Specifies the acceleration for the move, expressed in percentage of maximum, unless UseLinearVelAccelUnits = True, in which case the units of CmdAccel are mm/sec/sec.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the motors start to move. Either the function “MotorsWaitForMoveDone” will have to be called once for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped, or the host application can simply wait for the MoveAbsoluteAllAxesDone event.

If this method is called multiple times before the robot has completed the previous moves, it will be loaded into the move buffer and will be executed as soon as the previous moves have finished. Preloading multiple moves into the move buffer in this manner will allow each subsequent move to be preloaded into the robot, which reduces the pause time between moves.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

UseLinearVelAccelUnits—If set to “True”, the units for CmdVel will be mm/sec, and the units for CmdAccel will be mm/sec/sec.

SendEventWhenMoveDone—If set to “True”, the "MoveAbsoluteAllAxesDone" event will be triggered when the move is done.

MoveToLimit—If set to “True”, and the commanded move would exceed the travel limits of the robot, the robot is commanded to move as far as it can go, even though the original CmdPos will not actually be reached.

RefFrameRotate—Specifies a rotary offset, specified in degrees, for the gripper. This provides compensation for a gripper that is attached so that it does not face straight forward when the wrist is at zero.

ToolOffset—Specifies a linear offset from the center of the wrist to the center of the object being held by the gripper. The robot will move the object in the gripper along the path, instead of moving the center of the wrist along the path.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

IgnoreVisionOffset—If set to true, then the vision offset specified by SetVisionOffset() will be ignored even if the vision offset has been enabled via EnableVisionOffset().

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to move the robot along a linear path to a specific absolute position. It is recommended that the function “TeachPointMoveTo” be used instead of this function, in conjunction with the SetMovePathMode command, since this function would require hard coding of position values.

9.142 MotorWaitForMoveDone

(Axis As Short, TimeoutMsec As Integer) As Short

Parameters:

Axis—Specifies the motor.

TimeoutMsec—Specifies the number of milliseconds allowed to pass before the function fails.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function halts execution until the motor finishes its current move or the function times out. This function should be called for each motor that is moving, to determine when all motors have finished.

9.143 MoveAbsoluteAllAxes

(CmdPos() As Double, CmdVel As Double, CmdAccel As Double, WaitUntilDone As Boolean, Optional CalculatedMoveTimeMsec As Integer, Optional SendEventWhenMoveDone As Boolean, Optional ByRef Index As Byte) As Short

Parameters:

CmdPos()—Specifies the absolute commanded joint position for the move, expressed in millimeters and degrees.

CmdVel—Specifies the velocity for the move, expressed in percentage of maximum.

CmdAccel—Specifies the acceleration for the move, expressed in percentage of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the motors start to move. Either the function “MotorWaitForMoveDone” will have to be called once for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped, or the host application can simply wait for the MoveAbsoluteAllAxesDone event.

If this method is called multiple times before the robot has completed the previous moves, it will be loaded into the move buffer and will be executed as soon as the previous moves have finished. Preloading multiple moves into the move buffer in this manner will allow each subsequent move to be preloaded into the robot, which reduces the pause time between moves.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "MoveAbsoluteAllAxesDone" event will be triggered when the move is done.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to move the robot to a specific absolute position. The move will be either a Joint move or a Linear move, depending on the mode set by a prior SetMovePathMode call. It is recommended that the function “TeachPointMoveTo” be used instead of this function, since this function would require the host application to manage axis position values.

9.144 MoveAbsoluteAllAxesStop

(Optional Deceleration As Single = -1, Optional WaitUntilDone As Boolean) As Short

Parameters:

Deceleration—Not implemented. Deceleration is performed at the maximum allowed rate.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the motor has been commanded to stop.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to stop a multi-axis move prematurely.

9.145 MoveAbsoluteSingleAxis

(Axis As Short, Position As Double, Vel As Double, Accel As Double, WaitUntilDone As Boolean, Optional CalculatedMoveTimeMsec As Integer, Optional SendEventWhenMoveDone As Boolean, Optional ByRef Index As Byte) As Short

Parameters:

Axis—Specifies the motor.

Position—Specifies the absolute destination position of the move, expressed in degrees or millimeters.

Vel—Specifies the velocity, expressed in percent of maximum.

Accel—Specifies the acceleration, espressed in percent of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the motor starts to move. Either the function “MotorWaitForMoveDone” will have to be called once, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify that the motor has stopped.

If this method is called multiple times before the robot has completed the previous moves, it will be loaded into the move buffer and will be executed as soon as the previous moves have finished.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "MoveAbsoluteSingleAxisDone" event will be triggered when the move is done.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteSingleAxisDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to move a motor to a specific absolute position. It is recommended that the function “TeachPointMoveTo” be used instead of this function, since this function would require hard coding of the position value.

9.146 MoveCountGet

(MoveCount As Integer) As Short

Parameter:

MoveCount—Returns the total accumulated number of moves executed by the robot.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function retrieves the “MoveCount” value from the Windows Registry. This value represents the total number of moves executed by the robot, regardless of the move distance. Refer to Registry Keys section for more information.

9.147 MoveDistanceCountGet

(Axis As Short, Travel As Double, Optional Key As String) As Short

Parameters:

Axis—Specifies the axis for which the move distance count will be retrieved.

Travel—Returns the accumulated travel of the specified axis.

Key—Returns the name of the setting from the WindowsRegistry, such as “AxisTravelShoulder” or “AxisTravelGripper”.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function retrieves the “Axis?Travel” value stored in Windows Registry. This value represents the total accumulated distance moved by the specified axis. Refer to Registry Keys section for more information.

9.148 MoveDistanceCountIncrement

(Axis As Short, Travel As Double) As Short

Parameters:

Axis—Specifies the axis for which the accumulated travel will be incremented.

Travel—Specifies the move distance that will be added to the current “AxisTravel?” value stored in the Windows Registry.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function adds the value specified by the Travel parameter to the “AxisTravel?” value stored in the Windows Registry. This value represents the total accumulated distance moved by the specified axis. Refer to Registry Keys section for more information.

9.149 MoveRelativeCartesian

(MoveDist() As Double, CmdVel As Double, CmdAccel As Double, WaitUntilDone As Boolean, Optional CalculatedMoveTimeMsec As Integer, Optional SendEventWhenMoveDone As Boolean , Optional RefFrame As Short, Optional RefFrameRotate As Double, ToolOffset As Double, Optional MoveToLimit As Boolean = False , Optional ByRef Index As Byte) As Short

Parameters:

MoveDist()—Specifies the Cartesian coordinate offsets from the current position of the robot.

CmdVel—Specifies the velocity for the move, expressed in percentage of maximum.

CmdAccel—Specifies the acceleration for the move, expressed in percentage of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the move will be added to the move buffer, and the function will

return without waiting for motion to be executed. Either the function “MotorWaitForMoveDone” will have to be called once for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "MoveAbsoluteAllAxesDone" event will be triggered when the move is done.

RefFrame—This optional parameter can have the following values:
0 = World Reference Frame (relative to base of robot, +X = right, +Y = forward)
1 = Tool Reference Frame (relative to robot gripper, +X = right, +Y = forward)
2 = Rotated World Reference Frame (rotates the world reference frame by the number of degrees specified by RefFrameRotate parameter)

RefFrameRotate—This optional parameter specifies the number of degrees to rotate the World reference frame. If RefFrame=1, then the Tool Reference Frame will be rotated.

ToolOffset—This optional parameter specifies a linear offset in millimeters for the tool center of rotation from the center of the wrist.

MoveToLimit—If set to “True”, and the commanded move would exceed the travel limits of the robot, the robot is commanded to move as far as it can go, even though the original CmdPos will not actually be reached. Vision offset will not be applied to move if MoveToLimit = True.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Moves the robot a relative distance in a Cartesian coordinate direction from the current position of the robot.

9.150 MoveRelativeSingleAxis

(Axis As Short, Distance As Double, Vel As Double, Accel As Double, WaitUntilDone As Boolean, Optional CalculatedMoveTimeMsec As Integer,

Optional SendEventWhenMoveDone As Boolean, Optional ByRef Index As Byte) As Short

Parameters:

Axis—Specifies the motor.

Distance—Specifies the relative destination position of the move, expressed in degrees or millimeters. The motor will move the distance specified.

Vel—Specifies the velocity, expressed in percent of maximum.

Accel—Specifies the acceleration, espressed in percent of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the move will be added to the move buffer, and the function will return without waiting for motion to be executed. Either the function “MotorWaitForMoveDone” will have to be called once, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify that the motor has stopped.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application to be used as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "MoveRelativeSingleAxisDone" event will be triggered when the move is done.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to move a motor a specific relative distance from its current position. This can be useful when incrementing a motor repeatedly. This function can also be used to move the servo gripper.

9.151 MoveToArrayPoint

(TeachPoint() As String, NumPoints() As Short, MovePoint() As Short, CoordinateOffset() As Double, CmdVel As Double, CmdAccel As Double, WaitUntilDone As Boolean, Optional CalculatedMoveTimeMsec As Integer, Optional SendEventWhenMoveDone As Boolean, Optional ByRef Index As Byte, Optional RefFrame As Short, Optional RefFrameRotate As Double, Optional ToolOffset As Double) As Short

Parameters:

TeachPoint()—Specifies the Teach Points that define the corners of the array. A one-dimensional array is defined by two Teach Points—one at

each end (line of points). A two-dimensional array is defined by three Teach Points—one at three of the four corners (flat grid of points). A three-dimensional array is defined by four Teach Points—one at three of the four corners of a flat grid of points that define the first and second dimensions, and a fourth point directly offset from the third point in the direction of the third dimension. Data starts at index 0.

NumPoints()—Specifies the number of array points along each dimension of the array. Data starts at index 0, with one element less than TeachPoint.

MovePoint()—Specifies the point in the array to which the robot will move. Data starts at index 0, with one element less than TeachPoint.

CoordinateOffset()—Specifies a Cartesian coordinate offset from the array point. If no offset is desired, set CoordinateOffset = Nothing.

CmdVel—Specifies the velocity for the move, expressed in percentage of maximum.

CmdAccel—Specifies the acceleration for the move, expressed in percentage of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the move will be added to the move buffer, and the function will return without waiting for motion to be executed. Either the function “MotorWaitForMoveDone” will have to be called once for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "MoveAbsoluteAllAxesDone" event will be triggered when the move is done.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveRelativeSingleAxisDone event by comparing the move Index parameter to the event Index parameter.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

RefFrame—This optional parameter can have the following values:

0 = World Reference Frame (relative to base of robot, +X = right, +Y = forward)

1 = Tool Reference Frame (relative to robot gripper, +X = right, +Y = forward)

2 = Rotated World Reference Frame (rotates the world reference frame by the number of degrees specified by RefFrameRotate parameter)

RefFrameRotate—Specifies a rotary offset, specified in degrees, for the gripper. This provides compensation for a gripper that is attached so that it does not face straight forward when the wrist is at zero.

ToolOffset—Specifies a linear offset from the center of the wrist to the center of the object being held by the gripper. The offset will be calculated from the center of the object in the gripper instead of the center of the wrist.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to move the robot to a point in a 1D, 2D or 3D array. The array is defined by teach points taught at the corners of the array. A one-dimensional array can be used for placing objects in a stack, where the first teach point is the top of the stack and the second teach point is the bottom of the stack. This is especially useful if there are many items in the stack and if the stack is not perfectly parallel with the Z axis of the robot.

9.152 MoveToCartesian

(Coordinate() As Double, CmdVel As Double, CmdAccel As Double, WaitUntilDone As Boolean, Optional CalculatedMoveTimeMsec As Integer, Optional SendEventWhenMoveDone As Boolean, Optional ByRef Index As Byte) As Short

Parameters:

Coordinate()—Specifies the X, Y, Z and Theta values to which the robot will be moved.

CmdVel—Specifies the velocity for the move, expressed in percentage of maximum.

CmdAccel—Specifies the acceleration for the move, expressed in percentage of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the move will be added to the move buffer, and the function will return without waiting for motion to be executed. Either the function

"MotorWaitForMoveDone" will have to be called once for each motor, or the function "MotorCheckIfMotionDone" will have to be called repeatedly to verify each motor has stopped.

CalculatedMoveTimeMsec—Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to "True", the "MoveAbsoluteAllAxesDone" event will be triggered when the move is done.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to move the robot to an absolute Cartesian coordinate.

9.153 PlacePlateInHotel

(TopTeachPoint As String, BottomTeachPoint As String, RetractTeachPoint As String, HotelCapacity As Short, PlateNumber As Short, LiftHeightMM As Double, Velocity As Double, GripperTimeDelayMsec As Integer, IgnoreSensor As Boolean, DepartureHeightMM As Double, ApproachVelocity As Double, WayPoint As String, FirstMoveJointPathMode As Boolean, PreloadMoves As Boolean) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top slot of the hotel.

BottomTeachPoint—Specifies the teach point that has been taught at the bottom slot of the hotel. An acceleration value can be added onto the end of this parameter, separated by a semicolon.

RetractTeachPoint—Specifies the teach point that has been taught at the retract position. The retract position should be taught at the same Z height as the TopTeachPoint, with the robot retracted out of the hotel. A grip height Z offset value can be added onto the end of this parameter, separated by a semicolon.

HotelCapacity—Specifies the total number of slots in the hotel.

PlateNumber—Specifies the slot in the hotel where the plate currently in the gripper will be placed.

LiftHeightMM—Specifies the height in millimeters the robot must move above the current hotel slot when moving into and out of the slot.

Velocity— The desired velocity, expressed in percent of maximum.

GripperTimeDelayMsec—Specifies the number of milliseconds to wait after opening the gripper (give the gripper enough time to open before the robot moves away).

IgnoreSensor—If set to “true”, the gripper sensor will not be used for plate presence/absence verification. This is useful for testing a sequence without actually handling any plates.

DepartureHeightMM—When retracting out of the hotel, the gripper will be at the height specified by LiftHeightMM plus the value specified by DepartureHeightMM.

ApproachVelocity—Specifies the velocity of the initial move to in front of the hotel slot.

WayPoint—Specifies the teach point that has been taught in between the TopTeachPoint and the RetractTeachPoint. The robot will move to this intermediate position before moving into or out of the hotel. This is useful when using hotels that have little lateral clearance.

FirstMoveJointPathMode—If set to True, then the first move to the RetractTeachPoint will be executed in Joint MovePath Mode even if the MovePath Mode is set to Linear.

PreloadMoves—If set to True, then the moves associated with this function will be batch-loaded into the robot. This approach reduces pause time between moves.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function will place a plate in the desired slot of a random-access hotel. The hotel is defined by three teach points and by the number of plates that can be loaded into the hotel. The robot will move to above and retracted from the hotel slot, into the slot, down to the slot, and release the plate from the gripper. The robot will then move to above the plate, and then retracts out of the slot. A ResumeLine value will be passed back via ErrorCode(2), which can be used with PlacePlateInHotelResume(). An overload is available that excludes FirstMoveJointPathMode and PreloadMoves.

9.154 PlacePlateInHotelResume

Parameters:

Has all the same parameters as PlacePlateInHotel plus the following:

Resumeline As Short—Specifies the point at which to resume running PlacePlateInHotel

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

If an error occurs during the execution of PlacePlateInHotel, the point at which the error occurs will be saved in ErrorCode(2), which can be retrieved using the GetErrorCode method. This value can be passed into PlacePlateInHotelResume after recovering from the error.

9.155 PlacePlateInPitchStack

(TopTeachPoint As String, BottomTeachPoint As String, PlateNumber As Short, TopClearanceMM As Double, GripHeightOffset As Double, StackingPitch As Double, StackEntryVelocity As Double, Velocity As Double, Acceleration As Double, GripperTimeDelayMsec As Integer, Optional IgnoreSensor As Boolean) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top position in the stack. The height of this position is not important.

BottomTeachPoint—Specifies the teach point that has been taught at the bottom position in the stack.

PlateNumber—Specifies the position in the stack at which the plate is to be placed.

TopClearanceMM—Specifies the height in millimeters that the robot must move above the top of the stack when approaching and departing from the stack.

GripHeightOffset—Specifies the offset in millimeters above the bottom teachpoint at which the plate will be gripped. This offset is applied to all plates in the stack.

StackingPitch—Specifies the vertical distance in millimeters from the bottom of one plate to the bottom of the plate on top of it, when stacked.

StackEntryVelocity—The desired velocity while entering the top of the stack, expressed in percent of maximum. The plate is less likely to be bumped out of the gripper if the robot enters the top of the stack at a

reduced velocity. Once the robot has entered the top of the stack, it will switch to the velocity specified by the Velocity parameter.

Velocity— The desired velocity, expressed in percent of maximum.

Acceleration—The desired acceleration, expressed in percent of maximum.

GripperTimeDelayMsec—Specifies the number of milliseconds to wait after closing the gripper (give the gripper enough time to close before the robot moves away). A time delay is not necessary when using a servo gripper.

IgnoreSensor—If set to “true”, the gripper sensor will not be used for plate presence/absence verification. This is useful for testing a sequence without actually handling any plates.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs. The step at which an error occurs will be saved to ErrorCode(2), which can be used in PlacePlateInPitchStackResume.

Description:

This function will place a plate at a desired height on a stack of plates. The plate stack is defined by two teach points and by the stacking pitch of the plates being used. This allows the teach points to be used for multiple thicknesses of plates. The robot will move a plate to above the stack, move down into the stack, release the plate, and move out of the stack.

9.156 PlacePlateInPitchStackResume

Parameters:

Has all the same parameters as PlacePlateFromPitchStack plus the following:

Resumeline As Short—Specifies the point at which to resume running RemovePlateFromStack

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

If an error occurs during the execution of PlacePlateInPitchStack, the point at which the error occurs will be saved in ErrorCode(2), which can be retrieved using the GetErrorCode method. This value can be passed into PlacePlateInPitchStackResume after recovering from the error.

9.157 PlacePlateInStack

(TopTeachPoint As String, BottomTeachPoint As String, StackCapacity As Short, PlateNumber As Short, TopClearanceMM As Double, Velocity As Double, GripperTimeDelayMsec As Integer, Optional IgnoreSensor As Boolean) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top position in the stack. An initial velocity can be appended to the end of this parameter to cause the robot to move at a slower speed while first entering the stack (for example, TopTeachPoint = “Stack1Top;25” would move at 25% velocity until inside the stack).

BottomTeachPoint—Specifies the teach point that has been taught at the bottom position in the stack. An acceleration value can be added onto the end of this parameter, separated by a semicolon.

StackCapacity—Specifies the maximum number of plates that can be placed in the stack.

PlateNumber—Specifies the position in the stack where the plate currently in the gripper will be placed.

TopClearanceMM—Specifies the height in millimeters that the robot must move above the top of the stack when approaching and departing from the stack.

Velocity— The desired velocity, expressed in percent of maximum.

GripperTimeDelayMsec—Specifies the number of milliseconds to wait after opening the gripper (give the gripper enough time to open before the robot moves away).

IgnoreSensor—If set to “true”, the gripper sensor will not be used for plate presence/absence verification. This is useful for testing a sequence without actually handling any plates.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function will place a plate at a desired height on a stack of plates. The plate stack is defined by two teach points and by the number of plates that are in the stack. The robot will move to above the stack, move down to the specified height, and release the plate from the gripper. The robot will then move to above the stack.

9.158 PlacePlateInStackResume

Parameters:

Has all the same parameters as PlacePlateInStack plus the following:

Resumeline As Short—Specifies the point at which to resume running PlacePlateInStack

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

If an error occurs during the execution of PlacePlateInStack, the point at which the error occurs will be saved in ErrorCode(2), which can be retrieved using the GetErrorCode method. This value can be passed into PlacePlateInStack after recovering from the error.

9.159 PreClassTerminateCleanup

Description:

Call this method just prior to closing the connection to the DLL to allow the DLL to close all forms and to terminate all classes. If this method is not used, a memory-write error may occur.

9.160 RailMaintenancePerformed

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Resets Rail Maintenance Required to “FALSE” (stored in the Rail motor drive). Rail Maintenance Required parameter is set to “TRUE” every 5,000,000 meters of Rail travel, at which time maintenance is required (see User’s Manual). The MaintenanceRequired event will be triggered each time the Rail axis reaches the maintenance interval.

9.161 ReadAnalogInput

(Axis As Short, InputNumber As Short, Voltage As Double) As Short

Parameters:

Axis—Specifies the motor.

InputNumber—Specifies the input number (1 or 2).

Voltage—Returns a value between 0.0 and 10.0 for input 1, or a value between 0 and 4096 for input 2 (0-3VDC).

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Each motor has two analog inputs. Input 1 has a range of 0-10 VDC, and input 2 has a range of 0-3VDC. Both have a resolution of 12 bits. Use this method to read the voltage of the desired analog input.

9.162 ReadAnalogInputAveraged

(Axis As Short, InputNumber As Short, Voltage As Double, Optional SampleCount As Short) As Short

Parameters:
Axis—Specifies the motor.

InputNumber—Specifies the input number (1 or 2).

Voltage— Returns a value between 0.0 and 10.0 for input 1, or a value between 0 and 4096 for input 2 (0-3VDC).

SampleCount—Specifies the number of readings that will be collected and averaged.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Each motor has two analog inputs. Input 1 has a range of 0-10 VDC, and input 2 has a range of 0-3VDC. Both have a resolution of 12 bits. A more stable result can be produced by averaging several readings.

9.163 ReadInput

(Axis As Short, InputNumber As Short, State As Short) As Short

Parameters:
Axis—Specifies the motor.

InputNumber—Specifies the input number on the specified motor. The gripper sensor is Axis 2, input 4

State—Returns state of the input. A state of “1” means the sensor is activated. A state of “0” means the sensor is not activated.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Each motor drive has several digital inputs. Some of these can be accessed at the Auxiliary connector. This function can be used to read the state of any one of these inputs.

9.164 ReadStringFromFile

(ReadString As String, FileName As String, LineNumber As Short)

Parameters:
ReadString—Returns the data read from the file.

FileName—Specify the path and name of the file to be read.

LineNumber—Specify the line number to be read.

Description:
This is a non-robot-specific function provided as an extra utility for reading data from a text file.

9.165 ReadStringFromINIFile

(Filename As String, Heading As String, Item As String, Value As String, Optional Default As String = "??????", Optional ReportError As Boolean = True) As Short

Parameters:
Filename—Specifies the path and name of the INI file.

Heading—Specifies the section under which the value is to be read.

Item—Specifies the name of the value to be read.

Value—Returns the value of the item being read.

Default—If item does not exist, then the value specified by Default will be written.

ReportError—If set to True, then any error encountered will be saved in the ErrorLog file.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method is used for reading the value of an item in an INI file.

9.166 RemovePlateFromHotel

(TopTeachPoint As String, BottomTeachPoint As String, RetractTeachPoint As String, HotelCapacity As Short, PlateNumber As Short, LiftHeightMM As Double, Velocity As Double, GripperTimeDelayMsec As Integer, IgnoreSensor As Boolean, ApproachHeightMM As Double, ApproachVelocity As Double, WayPoint As String, FirstMoveJointPathMode As Boolean, PreloadMoves As Boolean) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top slot of the hotel.

BottomTeachPoint—Specifies the teach point that has been taught at the bottom slot of the hotel. An acceleration value can be added onto the end of this parameter, separated by a semicolon.

RetractTeachPoint—Specifies the teach point that has been taught at the retract position. The retract position should be taught at the same Z height as the TopTeachPoint, with the robot retracted out of the hotel. A grip height Z offset value can be added onto the end of this parameter, separated by a semicolon.

HotelCapacity—Specifies the total number of slots in the hotel.

PlateNumber—Specifies the slot in the hotel that contains the plate to be removed.

LiftHeightMM—Specifies the height in millimeters the robot must move above the current hotel slot when moving into and out of the slot.

Velocity— The desired velocity, expressed in percent of maximum.

GripperTimeDelayMsec—Specifies the number of milliseconds to wait after closing the gripper (give the gripper enough time to close before the robot lifts the plate).

IgnoreSensor—If set to “true”, the gripper sensor will not be used for plate presence/absence verification. This is useful for testing a sequence without actually handling any plates.

ApproachHeightMM—When extending into the hotel, the gripper will be at the height specified by LiftHeightMM plus the value specified by ApproachHeightMM.

ApproachVelocity—Specifies the velocity of the initial move to in front of the hotel slot.

WayPoint—Specifies the teach point that has been taught in between the TopTeachPoint and the RetractTeachPoint. The robot will move to this intermediate position before moving into or out of the hotel. This is useful when using hotels that have little lateral clearance.

FirstMoveJointPathMode—If set to True, then the first move to the RetractTeachPoint will be executed in Joint MovePath Mode even if the MovePath Mode is set to Linear.

PreloadMoves—If set to True, then the moves associated with this function will be batch-loaded into the robot. This approach reduces pause time between moves.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function will remove a plate from the desired slot of a random-access hotel. The hotel is defined by three teach points and by the number of plates that fit in the hotel. The robot will move to above and retracted from the slot, into the slot, down to the slot, and will grip the plate in the slot. The robot will then lift the plate, and then retract out of the slot. If an error occurs during the execution of RemovePlateFromHotel, the point at which the error occurs will be saved in ErrorCode(2), which can be retrieved using the GetErrorCode method. This value can be passed into RemovePlateFromHotelResume after recovering from the error. An overload is available that excludes FirstMoveJointPathMode and PreloadMoves.

9.167 RemovePlateFromHotelResume

Parameters:

Has all the same parameters as RemovePlateFromHotel plus the following:

Resumeline As Short—Specifies the point at which to resume running RemovePlateFromHotel

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

If an error occurs during the execution of RemovePlateFromHotel, the point at which the error occurs will be saved in ErrorCode(2), which can be retrieved using the GetErrorCode method. This value can be passed into RemovePlateFromHotelResume after recovering from the error.

9.168 RemovePlateFromPitchStack

(TopTeachPoint As String, BottomTeachPoint As String, PlateNumber As Short, TopClearanceMM As Double, GripHeightOffset As Double, StackingPitch As Double, Velocity As Double, Acceleration As Double, SearchForPlate As Boolean, SensorOffsetMM As Double, GripperTimeDelayMsec As Integer, Optional IgnoreSensor As Boolean, Optional GripPlate As Boolean, Optional SearchVelocity As Double, Optional YOffset As Double, Optional FlipWrist As Boolean, Optional MeasuredStackHeightMM As Double) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top position in the stack. The height of this position is not important.

BottomTeachPoint—Specifies the teach point that has been taught at the bottom position in the stack.

PlateNumber—Specifies the position in the stack where the plate to be removed is currently located. If SearchForPlate=True, then the measured plate count will be returned back through this parameter.

TopClearanceMM—Specifies the height in millimeters that the robot must move above the top of the stack when approaching and departing from the stack.

GripHeightOffset—Specifies the offset in millimeters above the bottom teachpoint at which the plate will be gripped. This offset is applied to all plates in the stack.

StackingPitch—Specifies the vertical distance in millimeters from the bottom of one plate to the bottom of the plate on top of it, when stacked.

Velocity— The desired velocity, expressed in percent of maximum.

Acceleration—The desired acceleration, expressed in percent of maximum.

SearchForPlate—If this parameter is set to “true” then the sensor mounted on the gripper will be used to locate the top plate in the stack. If this parameter is set to “false”, then the robot will move directly to the height calculated from the value of PlateNumber.

SensorOffsetMM—If SearchForPlate is set to “true”, once the gripper sensor detects the plate and the robot stops, the robot will factor in the value of this parameter when calculating the precise height to which the robot will move before gripping the plate. The robot may over- or under-shoot the right grip height by enough that the plate count may be calculated incorrectly. This parameter can be used to account for this potential error.

GripperTimeDelayMsec—Specifies the number of milliseconds to wait after closing the gripper (give the gripper enough time to close before the

robot moves away). A time delay is not necessary when using a servo gripper.

IgnoreSensor—If set to “true”, the gripper sensor will not be used for plate presence/absence verification. This is useful for testing a sequence without actually handling any plates.

GripPlate—If set to “false” and with SearchForPlate set to “true”, the quantity of plates in a stack can be found without removing the top plate. This is useful for determining the height at which to place a plate on a stack of unknown height.

SearchVelocity—Specifies the velocity for searching for the top of the stack (percentage of maximum velocity). The robot may collide with the top of the stack if the search velocity is set too high. The accuracy of the search is worse at higher speeds, although for many applications this is not a problem.

YOffset—Specifies a distance, in millimeters, to offset the gripper in the Cartesian Y direction. This can be used in situations where the plate sensor needs to be farther from or closer to the stack in order to improve search results. This parameter must be used when scanning with Top-Grip fingers. The recommended value is -125mm.

FlipWrist—If set to True, then the wrist will be positioned 180° from its normal position. This allows the use of a rear-facing plate sensor, which can be advantageous when using dropped fingers that strike the bottom of the stack when performing a conventional search.

MeasuredStackHeightMM—Returns the distance, in millimeters, from the bottom teach point to the position at which the robot stopped due to SearchForPlate=True plus the value of SensorOffsetMM.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs. The step at which an error occurs will be saved to ErrorCode(2), which can be used in RemovePlateFromPitchStackResume.

Description:

This function will remove a plate at a desired height from a stack of plates, or the gripper sensor can be used to detect the position of the top plate. The plate stack is defined by two teach points and by the stacking pitch of the plates being used. This allows the teach points to be used for multiple thicknesses of plates. The robot will move to above the stack, move down to the plate, grip it, and lift it out of the stack.

9.169 RemovePlateFromPitchStackResume

Parameters:

Has all the same parameters as RemovePlateFromPitchStack plus the following:

Resumeline As Short—Specifies the point at which to resume running RemovePlateFromStack

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

If an error occurs during the execution of RemovePlateFromPitchStack, the point at which the error occurs will be saved in ErrorCode(2), which can be retrieved using the GetErrorCode method. This value can be passed into RemovePlateFromPitchStackResume after recovering from the error.

9.170 RemovePlateFromStack

(TopTeachPoint As String, BottomTeachPoint As String, StackCapacity As Short, PlateNumber As Short, TopClearanceMM As Double, Velocity As Double, SearchForPlate As Boolean, SensorOffsetMM As Double, MeasuredPlateNumber As Short, GripperTimeDelayMsec As Integer, Optional IgnoreSensor As Boolean, Optional GripPlate As Boolean, Optional SearchVelocity As Double, Optional SecondSearchStartHeightMM As Double, Optional SecondSearchVelocity As Double, Optional MoveToCalculatedHeightAfterSearch As Boolean, Optional InitialLiftDistMM As Double) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top position in the stack.

BottomTeachPoint—Specifies the teach point that has been taught at the bottom position in the stack. An acceleration value can be added onto the end of this parameter, separated by a semicolon.

StackCapacity—Specifies the maximum number of plates that can be placed in the stack.

PlateNumber—Specifies the position in the stack where the plate to be removed is currently located.

TopClearanceMM—Specifies the height in millimeters that the robot must move above the top of the stack when approaching and departing from the stack.

Velocity— The desired velocity, expressed in percent of maximum.

SearchForPlate—If this parameter is set to “true” then the sensor mounted on the gripper will be used to locate the height of the plate. If this parameter is set to “false”, then the robot will move directly to the height calculated from the value of PlateNumber.

SensorOffsetMM—If SearchForPlate is set to “true”, once the gripper sensor detects the plate and the robot stops, the robot will move the amount specified by this parameter. The robot may not stop at exactly the right grip height, so this parameter can be used to adjust the height of the robot before gripping the plate.

MeasuredPlateNumber—If SearchForPlate is set to “true”, this parameter returns the number of plates in the stack once the top plate has been found.

GripperTimeDelayMsec—Specifies the number of milliseconds to wait after closing the gripper (give the gripper enough time to close before the robot moves away).

IgnoreSensor—If set to “true”, the gripper sensor will not be used for plate presence/absence verification. This is useful for testing a sequence without actually handling any plates.

GripPlate—If set to “false” and with SearchForPlate set to “true”, the quantity of plates in a stack can be found without removing the top plate. This is useful for determining the height at which to place a plate on a stack of unknown height.

SearchVelocity—Specifies the velocity for searching for the top of the stack (percentage of maximum velocity). The robot may collide with the top of the stack if the search velocity is set too high. The accuracy of the search is worse at higher speeds, although for many applications this is not a problem.

SecondSearchStartHeightMM—If given a non-zero value, once the initial search for the stack has been completed, the robot will move back up by the value specified here, and will perform a second search.

SecondSearchVelocity—Specifies the velocity for the second search for the top of the stack (percentage of maximum velocity). This value is used only if SecondSearchStartHeightMM has a non-zero value. Performing a second search at a greatly reduced velocity can improve the accuracy of the search.

MoveToCalculatedHeightAfterSearch—After searching and finding the top of the stack, this parameter specifies whether the robot will then move to the theoretical calculated height of the plate found, based on the top and bottom teach points and the calculated number of plates in the stack. If this parameter is set to False, then the robot will move the distance specified by SensorOffsetMM.

InitialLiftDistMM—After gripping the plate on the top of the stack, the robot will move upward at 100% velocity and 100% acceleration the

distance specified by this parameter. This can help shake loose any plates that might be stuck to the bottom of the top plate.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function will remove a plate at a desired height from a stack of plates, or the gripper sensor can be used to detect the position of the top plate. The plate stack is defined by two teach points and by the number of plates that are in a full stack. The robot will move to above the stack, move down to the plate, grip it, and lift it out of the stack.

9.171 RemovePlateFromStackResume

Parameters:

Has all the same parameters as RemovePlateFromStack plus the following:

Resumeline As Short—Specifies the point at which to resume running RemovePlateFromStack

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

If an error occurs during the execution of RemovePlateFromStack, the point at which the error occurs will be saved in ErrorCode(2), which can be retrieved using the GetErrorCode method. This value can be passed into RemovePlateFromStackResume after recovering from the error.

9.172 RemoveVisionOffset

(ByVal offsetJointPos() As Double, ByRef zeroJointPos() As Double)

Parameters:

offsetJointPos—An array containing robot joint positions that have already been modified based on the vision offset

zeroJointPos—Returns an array of robot joint positions that represents offsetJointPos with the vision offset unapplied to it

Description:

Removes the vision offset specified by SetVisionOffset from a robot joint position array. This function is called automatically by the DLL internal functions when EnableVisionOffset(True) has been called, specifically when teaching points while the vision offset is enabled.

9.173 RotateXYPointAboutOrigin

(X As Double, Y As Double, RotAngle As Double, NewX As Double, NewY As Double)

Parameters:

X—Specifies the horizontal coordinate of the point to be rotated.

Y—Specifies the vertical coordinate of the point to be rotated.

RotAngle—Specifies the angle, in degrees, by which the (X,Y) coordinate is to be rotated about the origin.

NewX—Returns the horizontal coordinate of the point after rotation.

NewY—Returns the vertical coordinate of the point after rotation.

Description:

This method receives a set of (X,Y) coordinates, and rotates them about the origin by the specified angle, generating a new set of (X,Y) coordinates.

9.174 SaveMoveDataToDrives

() As Short

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

During robot operation, motor cycle data is stored temporarily in the Windows Registry. This data should be saved to the motor drives occasionally so that it won't be lost if the motor is moved to a different PC. Normally, the ShutDown function saves the data to the motor drives, but this can be disabled using the SetSaveMoveDataOnShutDown function. Disabling automatic data saving may be desirable to avoid the potential data corruption that can occur if robot power is disconnected while data is being saved to the motor drives. Instead, the user has explicit control over saving the data by using this function.

9.175 ScanHotel

(TopTeachPoint As String, BottomTeachPoint As String, RetractTeachPoint As String, HotelCapacity As Short, PlateNumber As Short, LiftHeightMM As Double, Velocity As Double, PlateFound As Boolean, ApproachHeightMM As Double, ApproachVelocity As Double, WayPoint As String, FirstMoveJointPathMode As Boolean, PreloadMoves As Boolean) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top slot of the hotel.

BottomTeachPoint—Specifies the teach point that has been taught at the bottom slot of the hotel.

RetractTeachPoint—Specifies the teach point that has been taught at the retract position. The retract position should be taught at the same Z height as the TopTeachPoint, with the robot retracted out of the hotel.

HotelCapacity—Specifies the total number of slots in the hotel.

PlateNumber—Specifies the slot in the hotel to be scanned.

LiftHeightMM—Specifies the height in millimeters the robot must move above the current hotel slot when moving into and out of the slot.

Velocity— The desired velocity, expressed in percent of maximum.

PlateFound—Returns True if a plate was found in the slot, or False if no plate was found.

ApproachHeightMM—When extending into the hotel, the gripper will be at the height specified by LiftHeightMM plus the value specified by ApproachHeightMM.

ApproachVelocity—Specifies the velocity of the initial move to in front of the hotel slot.

WayPoint—Specifies the teach point that has been taught in between the TopTeachPoint and the RetractTeachPoint. The robot will move to this intermediate position before moving into or out of the hotel. This is useful when using hotels that have little lateral clearance.

FirstMoveJointPathMode—If set to True, then the first move to the RetractTeachPoint will be executed in Joint MovePath Mode even if the MovePath Mode is set to Linear.

PreloadMoves—If set to True, then the moves associated with this function will be batch-loaded into the robot. This approach reduces pause time between moves.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method is nearly identical to RemovePlateFromHotel, but it does not actually remove a plate from the hotel. Its purpose is to determine the presence of a plate in the specified shelf. An overload is available that excludes FirstMoveJointPathMode and PreloadMoves.

9.176 ScanStack

(TopTeachPoint As String, BottomTeachPoint As String, StackCapacity As Short, TopClearanceMM As Double, Velocity As Double, SensorOffsetMM As Double, MeasuredPlateNumber As Short, MeasuredStackHeightMM As Double, Optional SearchVelocity As Double, Optional YOffset As Double, Optional FlipWrist As Double) As Short

Parameters:

TopTeachPoint—Specifies the teach point that has been taught at the top position in the stack.

BottomTeachPoint—Specifies the teach point that has been taught at the bottom position in the stack.

StackCapacity—Specifies the maximum number of plates that can be placed in the stack.

TopClearanceMM—Specifies the height in millimeters that the robot must move above the top of the stack when approaching and departing from the stack.

Velocity— The desired velocity, expressed in percent of maximum.

SensorOffsetMM—Once the gripper sensor detects the plate and the robot stops, the robot will move the amount specified by this parameter. The robot may not stop at exactly the right grip height, so this parameter can be used to adjust the height of the robot before measuring the stack height.

MeasuredPlateNumber—This parameter returns the number of plates in the stack once the top plate has been found.

MeasuredStackHeightMM—This parameter returns the distance in millimeters from the Z value of BottomTeachPoint to the height at which the top of the stack was found.

SearchVelocity—Specifies the velocity at which the robot will search for the top of the stack. If this parameter is omitted, then 54mm/sec will be used.

YOffset—Specifies a distance, in millimeters, to offset the gripper in the Cartesian Y direction. This can be used in situations where the plate sensor needs to be farther from or closer to the stack in order to improve search results. This parameter must be used when scanning with Top-Grip fingers. The recommended value is -125mm.

FlipWrist—If set to True, then the wrist will be positioned 180° from its normal position. This allows the use of a rear-facing plate sensor, which can be advantageous when using dropped fingers that strike the bottom of the stack when performing a conventional search.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method is nearly identical to RemovePlateFromStack, but it does not actually remove a plate from the stack. Its purpose is to determine the number of plates currently in the stack.

9.177 ScriptCreate

(Filename As String, ScriptName As String, Operation() As String) As Short

Parameters:

Filename—Specifies the location of the file in which the script is to be saved. To use default sequence file, set equal to a blank string.

ScriptName—Specifies the name of the script to be saved. If the name already exists, the previous script will be overwritten. Scripts cannot be named “Variables”, as this name is reserved for storing script variables.

Operation()—This parameter is a string array containing all of the operations for the script being saved.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to save a new script to a file.

9.178 ScriptDelete

(Filename As String, ScriptName As String) As Short

Parameters:

Filename—Specifies the location of the file containing the script to be deleted. To use default sequence file, set equal to a blank string.

ScriptName—Specifies the name of the script to be deleted.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to delete an unwanted script from the specified script file.

9.179 ScriptEditorGetSize

(Height As Integer, Width As Integer)

Parameters:

Height—Returns the height of the Script Editor in units of pixels.

Width—Returns the width of the Script Editor in units of pixels.

Description:

Returns the height and width of the Script Editor window in units of pixels.

9.180 ScriptEditorShow

(Show As Boolean, Optional Mode As Short, Optional Top As Short, Optional Left As Short)

Parameters:

Show—If set to “true”, the window will open. Otherwise, the window will be closed.

Mode—If set to 1, the window will be modal. If set to 0, the window will be modeless.

Top—Specifies the position of the top of the window. The default is zero (SequenceEditorTop Registry setting is used).

Left—Specifies the position of the left side of the window. The default is zero (SequenceEditorLeft Registry setting is used).

Description:

A script is a series of robot and logical operations saved to a text file. This subroutine will open or close the script editor window.

9.181 ScriptGetOperations

(Filename As String, ScriptName As String, Operation() As String)

Parameters:

Filename—Specifies the location of the file containing the operations to be retrieved. To use default sequence file, set equal to a blank string.

ScriptName—Specifies the name of the script that contains the operations to be retrieved.

Operation()—Returns a string array containing the operations in the specified file and script.

Description:

Used to retrieve the operations of an existing script

9.182 ScriptOperationDelete

(Filename As String, ScriptName As String, Operation As Short) As Short

Parameters:

Filename—Specifies the location of the file containing the operation to be deleted. To use default sequence file, set equal to a blank string.

ScriptName—Specifies the name of the script that contains the operation to be deleted.

Operation—Specifies the operation number to be deleted (the first operation in a script is number zero).

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to delete an unwanted operation from the specified script in the specified script file.

9.183 ScriptOperationInsert

(Filename As String, ScriptName As String, Operation As String, OperationNumber As Short) As Short

Parameters:

Filename—Specifies the location of the file in which the operation is to be inserted. To use default sequence file, set equal to a blank string.

ScriptName—Specifies the name of the script in which the operation is to be inserted.

Operation—Specifies the operation to be inserted.

OperationNumber—Specifies the location in the script where the operation is to be inserted.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to insert a new operation at a specific location in the specified script in the specified script file.

9.184 ScriptOperationRun

(Operation As String, ReturnData As String) As Short

Parameters:

Operation—Specifies the operation to run.

ReturnData—Passes back data returned by a failed READINPUT operation.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to run a single script operation manually, not from a file. The following is a list of the available operations:

-BARCODEREAD,
WaitUntilDone, (TRUE or FALSE)
ReadMode, “SINGLE, MULTIPLE or CONTINUOUS)
ReadTime, (ONESECOND – NINESECONDS or INDEFINITE)
Barcode (use variable, reading passed back)
Example: “BARCODEREAD, TRUE,SINGLE,ONESECOND,$Barcode”
DLL Function: BarcodeRead
-BARCODEREADERSENDCOMMAND,
Command,
Timeout, (milliseconds)
Response (use variable, response passed back)
Example: “BARCODEREADERSENDCOMMAND, Z1,1000,$Response
DLL Function: BarcodeReaderSendCommand
-CALLSUBROUTINE,
ScriptName
Description: This is a script-specific command that allows another script to be called from the current script.
-COMMENT,
Text
Example: “COMMENT,Move robot to safe position”

Description: Provides means of adding descriptive text inside script for documentational purposes

-CONFIGUREINPUT
Axis,
InputNumber,
LogicHigh (TRUE or FALSE),
LogicType (FREEWHEEL, SOFTANDAUXSTOP, NONE, GENERALPURPOSE, ENABLEFORWARDONLY, ENABLEREVERSEONLY, BEGINMOTION, SOFTSTOP, ENABLEMAINHOMESEQUENCE, ENABLEAUXHOMESEQUENCE, STOPUNDERCONTROL, ABORTMOTION)
Example: “CONFIGUREINPUT,2,4,FALSE, ENABLEFORWARDONLY”
DLL Function: ConfigureInputLogic

-FLAG,
Name
Example: “FLAG,LoopStartFlag”
Description: This is a scrip-specific command that provides a destination for IFTHENGOTO and GOTO.

-GOTO,
DestinationType, (FLAG or SUBROUTINE)
Destination
Example: “GOTO,SUBROUTINE,ExitSequence”
Description: This is a script-specific command that allows the point of execution to jump to either a flag or a subroutine.

-HOMESINGLEAXIS,
Axis
Example: “HOMESINGLEAXIS,2”
DLL Function: HomeMotor

-IFTHENGOTO,
Variable1,
Comparator, (=, >, >=, <, <=, <>, Contains, DoesNotContain)
Variable2,
DestinationType, (FLAG or SUBROUTINE)
Destination
Example: “IFTHENGOTO,$Val1,=,5, FLAG,LoopStartFlag”
Description: This is a script-specific command that allows the point of execution to jump to either a flag or a subroutine based on the value of a variable. Use Contains and DoesNotContain comparators when working with non-numeric strings.

-INITIALIZE,
HomeGripper, (TRUE or FALSE)
SkipHomeIfAlreadyHome, (TRUE or FALSE)
HomeStatus (returned value)
Example: “INITIALIZE,FALSE,TRUE,$HomeStatus”
DLL Function: Initialize

-MESSAGEBOX,

Message,
Style, (OKCANCEL, ABORTRETRYIGNORE, YESNO, YESNOCANCEL, RETRYCANCEL)
Title,
Result (variable return value = Ok, Cancel, Abort, Retry, Ignore, Yes, No, or Cancel)
Example: “MESSAGEBOX,”Abort run?”,YESNO,”Operator Input Needed”,$Abort

-MOTORGETCURRENTPOSITION
Axis,
Position (returned value)
Example: “MOTORGETCURRENTPOSITION,1”
DLL Function: MotorGetCurrentPosition

-MOTORSENDCOMMAND,
Axis,
Command, (PX, MS, MO, etc.)
Index,
Value, (if changing a value, put the value here and don’t use an “=” sign)
ValFloat (TRUE or FALSE)
Response (if expecting a response, set equal to a variable name)
Example: “MOTORSENDCOMMAND,1,”PX”,0,”?”,FALSE,$Response”
DLL Function: MotorSendCommand

-MOVE,
TeachPointName,
Velocity,
Acceleration,
WaitUntilDone (TRUE or FALSE)
Example: “MOVE,TP1,50,50,TRUE”
DLL Function: TeachPointMoveTo

-MOVELINEARINCREMENTAL,
TeachPoint,
Velocity,
Acceleration,
Increments
Example: “MOVELINEARINCREMENTAL,TP1,25,25,10”
DLL Function: TeachPointMoveToLinearIncremental

-MOVERELATIVE,
TeachPointName,
Velocity,
Acceleration,
Axis1Offset,
Axis2Offset,
Axis3Offset,
Axis4Offset,
Axis5Offset, (rail, optional),
WaitUntilDone (TRUE or FALSE)
Example: “MOVERELATIVE,TP1,50,50,0,-20,0,0,TRUE
DLL Function: TeachPointMoveRelativeTo

-MOVERELATIVECARTESIAN,

Velocity,
Acceleration,
ReferenceFrame, (Tool or World)
ReferenceFrameAngle,
ToolOffset,
XOffset,
YOffset,
ZOffset,
ThetaOffset,
Rail Offset (optional),
WaitUntilDone (TRUE or FALSE)
Example: “MOVERELATIVECARTESIAN,50,50,
Tool,0,0,-125,0,0,TRUE”
DLL Function: MoveRelativeCartesian

-MOVERELATIVETPCARTESIAN,
TeachPointName,
Velocity,
Acceleration,
ReferenceFrame, (Tool or World)
ReferenceFrameAngle,
ToolOffset,
XOffset,
YOffset,
ZOffset,
ThetaOffset,
Rail Offset (optional),
WaitUntilDone (TRUE or FALSE)
Example: “MOVERELATIVETPCARTESIAN,TP1,50,50,
Tool,0,0,-125,0,0,TRUE”
DLL Function: TeachPointMoveRelativeToCartesian

-MOVESINGLEAXIS,
Axis,
Velocity,
Acceleration,
Position,
MoveRelative, (TRUE or FALSE)
WaitUntilDone (TRUE or FALSE)
Example: “MOVESINGLEAXIS,2,50,50,25,TRUE,TRUE”
DLL Functions: MoveAbsoluteSingleAxis,
MoveRelativeSingleAxis

-MOVESTOP
Example: “MOVESTOP”
DLL Function: JogStop

-MOVETOARRAYPOINT,
TeachPointsList, (semicolon-separated list)
NumberOfPointsList, (semicolon-separated list)
MovePointList, (semicolon-separated list)
CoordinateOffsetList, (semicolon-separated list)
Velocity,
Acceleration,
WaitUntilDone (TRUE or FALSE)
Example: “MOVETOARRAYPOINT, TP1;TP2;TP3, 5;10,
2;4, 0;0;20;0, 50,50,TRUE”

DLL Function: MoveToArrayPoint

-PLACEPLATEINHOTEL,
TopTeachPoint,
BottomTeachPoint,
RetractTeachPoint,
HotelCapacity,
PlateNumber,
LiftHeight,
Velocity,
Acceleration,
GripperTimeDelay,
IgnoreSensor, (TRUE or FALSE)
ExtraApproachHeight,
ApproachVelocity, (optional)
WayPoint (optional)
Example: “PLACEPLATEINHOTEL,TP1,TP2,TP3,
15,1,5,50,50,500,FALSE,0”
DLL Function: PlacePlateInHotel

-PLACEPLATEINSTACK,
TopTeachPoint,
BottomTeachPoint,
StackCapacity,
PlateNumber,
TopClearanceMM,
Velocity,
GripperTimeDelayMsec,
IgnoreSensor (TRUE or FALSE)
Example: “PLACEPLATEINSTACK,TP1,TP2,30,15,50,
50,500,FALSE”
DLL Function: PlacePlateInStack

-READINPUT,
Axis,
InputNumber,
DesiredState, (TRUE or FALSE)
IncorrectStateAction (STOP or CONTINUE)
WaitForTimeout (TRUE or FALSE)
Timeout (msec)
(i.e. “READINPUT,2,4,True,STOP,TRUE,1000”).
If “CONTINUE” is specified for the IncorrectStateAction, if an input is not in the desired state, the script will continue, and the sensor error will be reported at the end. If “STOP” is specified for the IncorrectStateAction, the script will be aborted.
Example: “READINPUT,2,4,TRUE,Stop,1000”
DLL Function: ReadInput

-READINPUTNOACTION
Axis,
InputNumber,
State (use a variable)
Example: “READINPUTNOACTION,2,4,$State”
DLL Function: ReadInput

-REMOVEPLATEFROMHOTEL,
TopTeachPoint,

BottomTeachPoint,
RetractTeachPoint,
HotelCapacity,
PlateNumber,
LiftHeight,
Velocity,
Acceleration,
GripperTimeDelay,
IgnoreSensor, (TRUE or FALSE)
ExtraApproachHeight,
Approach Velocity, (optional)
Waypoint (optional)
Example: “REMOVEPlATEFROMHOTEL,TP1,TP2,TP3,
15,1,5,50,50,500,FALSE,0”
DLL Function: RemovePlateFromHotel

-REMOVEPLATEFROMSTACK,
TopTeachPoint,
BottomTeachPoint,
StackCapacity,
PlateNumber, (if SearchForPlate=TRUE, set equal to a variable, and the current number of plates will be passed back)
TopClearanceMM,
Velocity,
SearchForPlate, (TRUE or FALSE)
SensorOffsetMM,
GripperTimeDelayMsec,
IgnoreSensor (TRUE or FALSE)
The measured plate number will be returned in the ReturnData parameter.
Example: REMOVEPLATEFROMSTACK,TP1,TP2,30,
15,50,50,TRUE,-5,500,FALSE”
DLL Function: RemovePlateFromStack

-SCANHOTEL,
TopTeachPoint,
BottomTeachPoint,
RetractTeachPoint,
HotelCapacity,
PlateNumber,
LiftHeight,
Velocity,
ExtraApproachHeight,
PlateFound
Example: “SCANHOTEL,TP1,TP2,TP3,
15,1,5,50,0,$PlateFound”
DLL Function: ScanHotel

-SCANSTACK,
TopTeachPoint,
BottomTeachPoint,
StackCapacity,
TopClearanceMM,
Velocity,
SensorOffsetMM,

MeasuredPlateNumber (set equal to a variable)
The measured plate number will be returned in the ReturnData parameter,
YOffset (optional),
FlipWrist (TRUE or FALSE)
Example: SCANSTACK,TP1,TP2,30,50,50,
-5,$PlateNum,-125,FALSE"
DLL Function: ScanStack

-SERIALPORTCLOSE,
PortNum

-SERIALPORTOPEN,
PortNum, (1, 2, etc.)
BaudRate, (9600,19200, etc.)
DataBits, (5, 6, 7, 8)
Parity, (EVEN, MARK, NONE, ODD, SPACE)
StopBits, (1, 1.5, 2)
HandShake (NONE, RTS, RTS XON/XOFF,XON/XOFF)

-SERIALPORTREAD,
PortNum,
ReadText

-SERIALPORTREADINPUT,
PortNum, (1, 2, etc.)
Input, (DCD, DSR, or CTS)
DesiredValue, (TRUE or FALSE)
IncorrectStateAction, (STOP or CONTINUE)
WaitForTimeout, (TRUE or FALSE)
Timeout (msec)

-SERIALPORTSETOUTPUT,
PortNum, (1, 2, etc.)
OutputPin, (DTR, or RTS)
OutputState, (TRUE, FALSE)
TimeDelay (msec)

-SERIALPORTWAIT,
PortNum,
WaitText,
TimeoutSec

-SERIALPORTWRITE,
PortNum,
Text,
AppendCarriageReturn (TRUE, FALSE)

-SERVOGRIPPERCLOSE,
Velocity,
ClosedPosition, (pass non-numeric to use default)
IgnoreCloseError (TRUE or FALSE)
Example: "SERVOGRIPPERCLOSE,100,Default,
FALSE"
DLL Function: ServoGripperClose

-SERVOGRIPPEROPEN,
Velocity,
OpenPosition, (pass non-numeric to use default)
WaitUntilDone ("TRUE" or "FALSE")
Example: "SERVOGRIPPEROPEN,100,Default, TRUE"
DLL Function: ServoGripperOpen

-SERVOGRIPPERSETCLOSEDPOSITION
DefaultGripperClosedPosition
Example: “SERVOGRIPPERSETCLOSEDPOSITION,0”
DLL Function: SetDefaultServoGripperClosedPosition
-SERVOGRIPPERSETGRIPFORCE,
GripForcePercent
Example: “SERVOGRIPPERSETGRIPFORCE,50”
DLL Function: ServoGripperSetDefaultGrippingForce
-SERVOGRIPPERSETOPENPOSITION
DefaultGripperOpenPosition
Example: “SERVOGRIPPERSETOPENPOSITION,26”
DLL Function: SetDefaultServoGripperOpenPosition
-SETJOINTMOVEDIRECTION
DirectionAxis1 (SHORTESTWAY, CCW or CW)
DirectionAxis2
DirectionAxis3
DirectionAxis4
DirectionAxis5 (rail, optional)
Example: “SETJOINTMOVEDIRECTION, CCW,
SHORTESTWAY,SHORTESTWAY,SHORTESTWAY”
DLL Function: SetJointMoveDirection
-SETMOVEPATHMODE
Mode (LINEAR or JOINT)
Example: “SETMOVEPATHMODE,LINEAR”
DLL Function: SetMovePathMode
-SETVARIABLE,
VariableName, ($VarName)
VariableValue (+ - * / math operations ok)
Example: “SETVARIABLE,$LoopCount,$LoopCount+1”
DLL Function: ScriptVariableSetValue
Description: Script variable names must start with a $
sign. Script variable names can be passed in for script
operation parameter values.
-SETOUTPUT,
Axis,
OutputNumber,
State, (TRUE or FALSE)
TimeDelay
Example: “SETOUTPUT,1,2,True,500”
DLL Function: SetOutput
-SLEEP,
TimeDelayMsec
Example: “SLEEP,1000”
DLL Function: DelayMsec
-VERIFYPOSITION,
TeachPointName,
Tolerance (mm or deg.)
Example: “VERIFYPOSITION,Safe,5”
DLL Function: DetermineTeachPoint
-VISIONALIGNROBOT1,
AprilTagName,
Velocity,
Acceleration,

VisionEnableOffset
Example:
"VISIONALIGNROBOT1,AprilTag5,100,100,True"
DLL Function: VisionAlignRobot1

-VISIONALIGNROBOT2,
AprilTagName1,
AprilTagName2,
Velocity,
Acceleration,
VisionEnableOffset
Example:
"VISIONALIGNROBOT2,AprilTag6,AprilTag7,100,100,
True"
DLL Function: VisionAlignRobot2

-VISIONENABLEOFFSET,
Enabled
Example: "VISIONENABLEOFFSET,True"
DLL Function: EnableVisionOffset

-VISIONEXPOSURETIME,
Time (ms)
Example: "VISIONEXPOSURETIME,200"
DLL Function: CameraSetExposureTime

9.185 ScriptResume

(Filename As String, ScriptName As String, ResumeLine As Short, Optional WaitUntilDone As Boolean) As Short

Parameters:

Filename—Specifies the location of the file in which the script is located. If a blank string is passed in, the default sequence file will be used.

ScriptName—Specifies the name of the script to be resumed.

ResumeLine—Specifies the line number at which execution of the script is to start.

WaitUntilDone—Specifies whether the function will return immediately, or after the script has run to completion. The ScriptDone event will be triggered upon completion of the script.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to run a script contained in a file, starting at any desired line within the script. If an error occurs during the middle of executing a script with ScripRun, the line number where that error occurred will be written to ErrorCode(2), which can be retrieved using GetErrorCode(2). Use ScriptResume to continue execution of the script.

9.186 ScriptResumeNested

(Filename As String, NestedScript() As String, NestedScriptLine() As Short, NestCount As Short, Optional WaitUntilDone As Boolean, Optional RunNextOperationOnly As Boolean) As Short

Parameters:

Filename—Specifies the location of the file in which the script is located. If a blank string is passed in, the default sequence file will be used.

NestedScript()—An array containing the names of the scripts in the current call stack.

NestedScriptLine()—An array containing the current line being executed in each script in the current call stack.

NestCount—The number of scripts deep that the point of execution is nested.

WaitUntilDone—Specifies whether the function will return immediately, or after the script has run to completion. The ScriptDone event will be triggered upon completion of the script.

RunNextOperationOnly—Will execute only the next operation.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to run a script contained in a file, starting at any desired point within the script, including nested locations. Script operations such as IFTHENGOTO and CALLSUBROUTINE can form a call stack that cannot be resumed purely by referring to a line number. Instead, the entire call stack must be known in order to resume execution. The ScriptRunning event must be monitored, and its parameter values stored, in order to use ScriptResumeNested.

9.187 ScriptRun

(Filename As String, ScriptName As String, Optional WaitUntilDone As Boolean) As Short

Parameters:

Filename—Specifies the location of the file in which the script is located. If a blank string is passed in, the default parameter file will be used.

ScriptName—Specifies the name of the script to be run.

WaitUntilDone—Specifies whether the function will return immediately, or after the script has run to completion. The ScriptDone event will be triggered upon completion of the script.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs. The line number where the error occurred will be saved to ErrorCode(1), which can be retrieved using GetErrorCode(1). Execution of the script can be restarted at the point where the error occurred by using ScriptResume.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to run a script contained in a file.

9.188 ScriptsGetNames

(Filename As String, ScriptName() As String) As Short

Parameters:

Filename—Specifies the location of the file from which the script names are to be retrieved. If a blank string is passed in for the Filename, the default Filename will be used.

ScriptName()—Returned containing the names of all scripts in the file specified.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

A script is a series of robot and logical operations saved to a text file. This function can be used to retrieve all of the script names contained in a script file.

9.189 ScriptStop

Description:

This function will stop the execution of any script that is being executed. The script will stop after the current operation is completed.

9.190 ScriptVariableDelete

(Filename As String, Name As String) As Short

Parameters:

Filename—Specifies the location of the file containing script variable. If a blank string is passed in for the Filename, the default Filename will be used.

Name—Specifies the name of the script variable to be deleted.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to delete an unwanted script variable.

9.191 ScriptVariableGetValue

(Filename As String, Name As String, Value As String) As Short

Parameters:

Filename—Specifies the location of the file containing script variables. If a blank string is passed in for the Filename, the default Filename will be used.

Name—Specifies the name of the script variable to be read.

Value—Returns the value of the variable being read.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to retrieve the value of a specific script variable.

9.192 ScriptVariableSetValue

(Filename As String, Name As String, Value As String) As Short

Parameters:

Filename—Specifies the location of the file containing script variables. If a blank string is passed in for the Filename, the default Filename will be used.

Name—Specifies the name of the script variable to be set.

Value—Specifies the value of the variable being set.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This function can be used to set the value of a specific script variable, or to create a new variable.

9.193 ScriptVariablesGetNames

(Filename As String, Names() As String) As Short

Parameters:
Filename—Specifies the location of the file containing script variables. If a blank string is passed in for the Filename, the default Filename will be used.

Names—Returns a list of the names of all existing script variables.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This function can be used to retrieve the names of all script variables.

9.194 SendEmailMessage

(EmailAddress As String, EmailSubject As String, EmailMessage As String, Optional FileAttachmentPath As String) As String

Parameters:
EmailAddress—Specifies the destination email message.

EmailSubject—Specifies the text of the subject line of the email message.

EmailMessage—Specifies the text of the email message.

FileAttachmentPath—Specifies the path and name of a file to be attached.

Return Value:
Returns the text of the error code produced by the email application.

Description:
This is a non-robot-specific function provided as an extra utility for sending email messages. Can be utilized for notifying maintenance

personnel of the occurrence of a robot error. The email account settings must be specified in the Teach Pendant/Options/Mail Settings window.

9.195 SerialPortClose

(PortNum As Short) As String

Parameters:

PortNum—Specifies COM port to be closed.

Return Value:

Returns “0” if the port closes successfully. Otherwise an error message is returned.

Description:

This function can be used to close a serial port that was opened with SerialPortOpen.

9.196 SerialPortGetInputState

(Portnum As Short, InputPin As eSerialInputPin, State As Boolean) As String

Parameters:

Portnum—Specifies COM port to be used.

InputPin (Enum type eSerialInputPin)—The following is a list of available values:

eSerialInputPin.DCD (Pin 1)
eSerialInputPin.DSR (Pin 6)
eSerialInputPin.CTS (Pin 8)
eSerialInputPin.RI (Pin 9)

State—Returns the current voltage level of the input pin (True=High, False=Low)

Return Value:

Returns “0” if the port input is read successfully. Otherwise an error message is returned.

Description:

This function can be used to read the states of the input pins on a serial port that was opened with SerialPortOpen.

9.197 SerialPortOpen

(PortNum As Short, BaudRate As Integer, DataBits As Short, Parity As eParity, StopBits As eStopBits, Handshake As eHandShake) As String

Parameters:

PortNum—Specifies COM port to be used.

BaudRate—Specifies the communication rate (bits per second) for the COM port.

DataBits—Specifies the number of data bits to use.

Parity (Enum type eParity)—The following is a list of the available values:
eParity.Even
eParity.Mark
eParity.None
eParity.Odd
eParity.Space

StopBits (Enum type eStopBits)—The following is a list of the available values:
eStopBits.One
eStopBits.OnePointFive
eStopBits.Two

Handshake (Enum type eHandShake)—The following is a list of the available values:
EHandShake.None
EHandShake.RequestToSend
EHandShake.RequestToSendXOnXOff
EHandShake.XonXoff

Return Value:
Returns “0” if the port opens successfully. Otherwise an error message is returned.

Description:
This function can be used to open a serial port that is not being used by the robot.

9.198 SerialPortRead

(PortNum As Short, Text As String) As String

Parameters:
PortNum—Specifies COM port to be used.

Text—Returns the ASCII string data stored in the serial port input buffer.

Return Value:
Returns “0” if the port is read successfully. Otherwise an error message is returned.

Description:
This function can be used to read the input buffer contents of the serial port that was opened using SerialPortOpen.

9.199 SerialPortSetOutputState

(Portnum As Short, OutputPin As eSerialOutputPin, State As Boolean) As String

Parameters:

Portnum—Specifies COM port to be used.

OutputPin (Enum type eSerialOutputPin)—The following is a list of available values:

eSerialOutputPin.DTR (Pin 4)
eSerialOutputPin.RTS (Pin 7)

State—Specifies the desired voltage level of the output pin (True=High, False=Low)

Return Value:

Returns “0” if the port output is set successfully. Otherwise an error message is returned.

Description:

This function can be used to set the states of the output pins on a serial port that was opened with SerialPortOpen

9.200 SerialPortWait

(PortNum As Short, WaitText As String, TimeoutSec As Double) As String

Parameters:

PortNum—Specifies COM port to be used.

WaitText—Specifies the desired ASCII string data for which the software will wait to receive from the serial port.

TimeoutSec—Specifies the amount of time in seconds that the software will wait for WaitText to be received. If WaitText is not received within the time specified by TimeoutSec, then “Timeout” will be returned.

Return Value:

Returns “0” if WaitText is received successfully. Otherwise an error message is returned.

Description:

This function can be used to wait for a specific message to be received by a serial port that was opened using SerialPortOpen.

9.201 SerialPortWaitForInputState

(Portnum As Integer, InputPin As eSerialInputPin, State As Boolean, TimeoutMsec As Long) As String

Parameters:

Portnum—Specifies COM port to be used.

InputPin (Enum type eSerialInputPin)—The following is a list of available values:

eSerialInputPin.DCD (Pin 1)
eSerialInputPin.DSR (Pin 6)
eSerialInputPin.CTS (Pin 8)
eSerialInputPin.RI (Pin 9)

State—Specifies the desired voltage level of the input pin (True=High, False=Low)

TimeoutMsec—Specifies the maximum time allowed for the InputPin to reach the desired State.

Return Value:

Returns “0” if the port reaches the desired State within the Timeout period. Otherwise an error message is returned.

Description:

This function can be used to halt program execution until the state of a serial port input pin reaches the desired state.The serial port must first be opened with SerialPortOpen.

9.202 SerialPortWrite

(Portnum As Short, Text As String) As String

Parameters:

Portnum—Specifies COM port to be used.

Text—Specifies the ASCII string data to be sent to the serial port.

Return Value:

Returns “0” if the data is written to the port successfully. Otherwise an error message is returned.

Description:

This function can be used to send communication data to a serial port that was opened using SerialPortOpen.

9.203 ServoGripperClose

(Position As Double, Velocity As Double) As Short

Parameters:

Position—Specifies the desired closed position.

Velocity—Specifies the velocity, expressed in percent of maximum. If a negative value is used, the gripper will grip in the open direction.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Moves the gripper toward the closed position until resistance is encountered, indicating a successful grip.

9.204 ServoGripperGetHomedStatus

(Status As Boolean) As Short

Parameters:
Status—Returns the Homed status of the servo gripper.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Reports the Homed status of the servo gripper.

9.205 ServoGripperInitializeNoWait

(Optional SkipHomeIfAlreadyHomed As Boolean, Optional HomeMotor As Boolean)

Parameters:
SkipHomeIfAlreadyHomed—This optional parameter specifies whether the gripper is to be homed again, even if it has already been homed. When the power to the robot is turned off, the gripper encoder loses its position, which makes homing a requirement. If the gripper has already been homed, and power has not been cycled since then, the gripper encoder will retain its position, and homing will not be required. If this parameter is set to “true”, the application will check if the encoder position has been reset due to power cycling. If the encoder position has not been reset, then the gripper will not be homed unnecessarily. If the encoder position has been reset, then the gripper will be homed.

HomeMotor—This optional parameter specifies whether the gripper is to be homed. The default value is "true". If the gripper was previously homed successfully and power has not been turned off since then, and

SkipHomeIfAlreadyHomed=True, then homing will be skipped even if HomeMotor=True.

HomeMotor	SkipHomeIfAlreadyHomed	Action
True	True	Home only if needed
True	False	Home always
False	True or False	Home never

Description:

Enables the gripper motor and moves it home, if specified to do so by the parameters. This function will work only if Initialize() has already been executed successfully. Initialize() initializes the gripper internally, so this function may not be necessary to use. This function returns prior to completion of gripper initialization, which executes in the background.

9.206 ServoGripperOpen

(Position As Double, Velocity As Double, Optional WaitUntilDone As Boolean, Optional ByRef CalculatedMoveTimeMsec As Integer, Optional SendEventWhenMoveDone As Boolean, Optional ByRef Index As Byte) As Short

Parameters:

Position—Specifies the desired open position

Velocity—Specifies the velocity, expressed in percent of maximum

WaitUntilDone—This optional parameter has a default value of “true”. If set to “true”, execution will pause until the gripper is open. If set to “false”, execution will return before motion has been completed.

CalculatedMoveTimeMsec—This optional parameter returns the estimated time to complete the move.

SendEventWhenMoveDone—If this parameter is set to “true”, and WaitUntilDone is set to “false”, then the MoveAbsoluteSingleAxisDone event will be raised upon completion of motion.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteSingleAxisDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Moves the gripper to the specified position. MoveAbsoluteSingleAxis and MoveRelativeSingleAxis can also be used to move the servo gripper (Axis 6). Setting WaitUntilDone to ‘false’ allows the other robot axes to be moved at the same time the gripper is opening.

9.207 ServoGripperQueryGrippingForce

(ByRef MaxForcePercent As Byte) As Short

Parameter:

MaxForcePercent—Returns the gripping force of the servo gripper as a percentage of maximum.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method is used for determining the percent of maximum force the servo gripper is using to grip an object. A value of zero indicates that the gripper is not gripping the object adequately, or that there is no object in the gripper.
gripper is not gripping the object adequately, or that there is no object in the gripper.

9.208 ServoGripperSetDefaultGrippingForce

(MaxForcePercent As Short) As Short

Parameter:

MaxForcePercent —Specifies the gripping force, expressed as a percentage of maximum.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method is used for adjusting the amount of force the servo gripper uses to grip an object.

9.209 SetBarcodeReaderSerialPort

(PortNum As Byte)

Parameter:

PortNum—sets the port number.

Description:

This method sets the PC COM port number to be used for the robot barcode reader located in the gripper.

9.210 SetCANDevice

(Description As String)

Parameter:

Description—Sets the desired CAN device.

Description:

This method sets the CAN/USB device (internal to the robot) to be used.

9.211 SetDefaultErrorLogFile

(ErrorLogFileName As String)

Parameters:

ErrorLogFileName—Specifies the complete file path for the error log file.

Description:

Robot errors are written to the error log file. The location of the error log file is stored in the registry. The default location is “C:\ProgramData\Peak Robotics, Inc\KX2 Robot Control DLL\ErrorLog.txt”.

9.212 SetDefaultGripperShutdownState

(GripperOpen As Boolean)

Parameter:

GripperOpen—Specifies whether the gripper will be open (True) or closed (False) after calling ShutDown().

Description:

Controls what state the gripper will be in after shutting down the robot. The setting is saved in the registry, so it is unnecessary to call this method repeatedly.

9.213 SetDefaultGripperState

(GripperOpen As Boolean)

Parameter:

GripperOpen—Specifies whether the gripper will be open (True) or closed (False) after homing when calling Initialize().

Description:

Controls what state the gripper will be in after homing when initializing the robot. The setting is saved in the registry, so it is unnecessary to call this method repeatedly. The Teach Pendant\Options\Communication window can also be used for changing the default gripper state.

9.214 SetDefaultSequenceFile

(SequenceFileName As String)

Parameters:

SequenceFileName —Specifies the complete file path for the sequence file.

Description:

The sequence file stores motion sequences created with the Sequence Editor. The location of the sequence file is stored in the registry.

9.215 SetDefaultServoGripperClosedPosition

(ClosedPosition As Double) As Short

Parameter:

ClosedPosition—Specifies the closed position of the servo gripper, in millimeters.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This value is used by the ServoGripperClose() method. This value can also be set in the Teach Pendant /Options window, and is stored in the parameter file. This value will be used by Stack and Hotel functions.

9.216 SetDefaultServoGripperOpenPosition

(OpenPosition As Double) As Short

Parameter:

OpenPosition—Specifies the open position of the servo gripper, in millimeters.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This value is used by the ServoGripperOpen() method. This value can also be set in the Teach Pendant/Options window, and is stored in the parameter file. This value will be used by Stack and Hotel functions.

9.217 SetDefaultTeachPointFile

(TeachPointFileName As String)

Parameters:

TeachPointFileName —Specifies the complete file path for the teach point file.

Description:

The teach point file stores teach point coordinates created with the Teach Pendant. The location of the teach point file is stored in the registry.

9.218 SetDefaultVisionCalibrationFile

(VisionCalibrationFileName As String)

Parameter:

VisionCalibrationFileName—Specifies the complete file path for the vision calibration file.

Description:

This method sets the default vision calibration file in the registry. The vision calibration file stores AprilTag calibration data used by the integrated camera.

9.219 SetEthernetPort

(ByVal Port As Integer)

Parameter:

Port – Specifies the port number for the Ethernet interface.

Description:

Specifies the port number of the CAN/Ethernet gateway. This function merely informs the software of the port number. If the port number needs to be changed, then the gateway configuration utility must be used for that purpose. The robot hardware must match the specified interface type.

9.220 SetInterfaceType

(ByVal Interface_Type As eInterfaceType)

Parameter:

The following values are accepted:

USB—A CAN/USB adapter is specified

Ethernet—An SSTcomm GT200-MT-CA CAN/Ethernet gateway is specified

Description:

Specifies the interface type being used by the PC to communicate with the robot. The robot hardware must match the interface type specified.

9.221 SetJointMoveDirection

(Axis As Short, Direction As eJointMoveDirection)

Parameters:

Axis—The robot axis that will move in the specified direction.

Direction—One of the following options:

Normal—The axis will move the direction that does not require passing through the 0°/360° rollover point

Clockwise—The axis will always move in the clockwise (negative) direction

Counterclockwise—The axis will always move in the counterclockwise (positive) direction

ShortestWay—The axis will always move the shortest direction, even if it must pass through the 0°/360° rollover point

Description:

Specifies the move direction for an axis that has unlimited travel, such as the Shoulder or Wrist.

9.222 SetMaintainGripAfterShutdown

(MaintainGrip As Boolean)

Parameter:

MaintainGrip—If set to True, the gripper motor will be left enabled after shutdown. If False, the motor will be disabled.

Description:

This method stores the setting for the post-shutdown gripper motor state in the registry. If set to True, the gripper motor will be left in an energized

state after the ShutDown function is called. Otherwise, the gripper motor will be de-energized.

9.223 SetMasterVelocityScale

(VelocityScale As Double)

Parameters:

VelocityScale —Specifies the master velocity scale from 0 to 100.

Description:

The master velocity scale value is used for scaling down the velocity of all motion commands sent to the robot. The value of the master velocity scale is stored in the registry.

9.224 SetMovePathMode

(Mode As eMovePathMode)

Parameters:

eMovePathMode —The following is a list of available values:

0 – Joint
1 – Linear

Description:

This method specifies whether the robot will execute joint or linear moves when commanded via the MoveAbsoluteAllAxes method, as well as other methods that use the MoveAbsoluteAllAxes, like the stack and hotel methods.

9.225 SetOutput

(Axis As Short, OutputNumber As Short, State As Short) As Short

Parameters:

Axis—Specifies the axis with the output to be controlled.

OutputNumber—Specifies the output to be controlled.

State—Specifies whether the output is on (State = 1) or off (State = 0).

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Provides a method for controlling digital outputs.

9.226 SetRegistryRootDirectory

(Directory As eRegistryRootDirectory)

Parameter:
Directory—Specifies the root directory.
HKEY_CURRENT_USER = 0
HKEY_LOCAL_MACHINE = 1

Description:
The default root directory is HKEY_CURRENT_USER. Set the root directory to HKEY_LOCAL_MACHINE to allow the same registry settings to be used by multiple Windows User accounts. All files used by the software must be placed in a folder that has full permissions for all users.

9.227 SetRobotIPAddress

(ByVal IPAddress As String)

Parameter:
IPAddress – Specifies the IP address for the Ethernet interface.

Description:
Specifies the IP address of the CAN/Ethernet gateway. This function merely informs the software of the IP address. If the IP address needs to be changed, then the gateway configuration utility must be used for that purpose. The robot hardware must match the specified interface type.

9.228 SetRobotSerialNumber

(SerialNumber As String)

Parameter:
SerialNumber —Specifies the serial number of the robot being controlled by this instance of the KX2RobotControl class. Using the serial number on the base of the robot is recommended, although any unique description can be used.

Description:
This method must be used when creating multiple instances of the KX2RobotControl class for operating multiple robots from one computer. Call SetRobotSerialNumber immediately after creating each instance of the KX2RobotControl class, and give each instance a unique value for

SerialNumber. This method will allow the DLL to save unique settings in the Windows Registry for each robot.

9.229 SetSaveMoveDataOnShutDown

(Enable As Boolean)

Parameter:

True enables the setting. False disables the setting.

Description:

Sets the value of the SaveMoveDataOnShutDown registry setting that controls whether move cycle data will be saved to the motor drives when the ShutDown function is called.

9.230 SetSaveMoveDataOnShutDownOverride

Description:

Call this function prior to calling ShutDown to force ShutDown not to save move data to the motor drives regardless of the value of the SaveMoveDataOnShutDown setting.

9.231 SetSubWindowMode

(SWMode As Integer)

Parameter:

SWMode:

0 = Modeless
1 = Modal

Description:

Sets the value of ModelessSubWindows in the Windows Registry. The value can also be changed by selecting “Modeless Sub-Windows” in the Teach Pendant/Tools/Miscellaneous menu.

9.232 SetTeachPendantAccessLevel

(AccessLevel As eTeachPendantAccessLevel)

Parameter:

AccessLevel—Specifies the allowed level of access.

Full = 0, Enables all functionality
Technician = 1, Allows reteaching, but teachpoints cannot be renamed

Limited = *2*, Cannot modify teachpoints

Description:

Provides the ability to limit the functionality of the Teach Pendant window. The access level can also be changed in the Teach Pendant/Tools menu.

9.233 SetTeachPendantAccessPassword

(AccessLevel As eTeachPendantAccessLevel, Password As String)

Parameters:

AccessLevel—Specifies the access level to which the password will be applied.

Full = 0
Technician = 1
Note: Limited access has no password.

Password—Specifies the desired password.

Description:

Use this method to set or change the password for a specific access level. The password is then encrypted and saved in the Windows Registry.

9.234 SetVisionOffset

(ByVal zeroRef3DCoord As t3DCoord, ByVal zeroRefQuaternion As tQuaternion, ByVal offsetRef3DCoord As t3DCoord, ByVal offsetRefQuaternion As tQuaternion, Optional ByVal name As String = “”)

Parameters:

zeroRef3DCoord – Specifies the (x,y,z) spatial position coordinate of the vision target, as recorded when positioned at the designated zero position during setup.

zeroRefQuaternion – Specifies the (X,Y,Z,W) rotational position coordinate of the vision target, as recorded when positioned at the designated zero position during setup.

offsetRef3DCoord – Specifies the (x,y,z) spatial position coordinate of the current location of the vision target.

offsetRefQuaternion – Specifies the (X,Y,Z,W) rotational position coordinate of the current location of the vision target.

name – Specifies a recognizable name for distinguishing the current vision offset from others. Will be displayed in the tooltip for the vision offset Enable/Disable button on the Teach Pendant.

Description:

Loads a set of vision system zero reference and current positions into the software. Call EnableVisionOffset to activate the offset. Once activated, all multi-axis moves will be adjusted based on the coordinates loaded through this function. The zero reference coordinates of the vision target should be recorded with the vision system and stored externally when the robot teach points are first taught. When the teach points are to be accessed by the robot later when the location of the robot has moved relative to the teach points, the vision system should then read the current position of the target. The zero reference and current positions are then passed through this function, and EnableVisionOffset(True) is called. Subsequent multi-axis move commands will be adjusted accordingly to account for the new location of the robot.

9.235 SetWarningAfterIdleTime

(Seconds As Long)

Parameter:

Seconds—Specifies the idle time setting, in seconds.

Description:

Sets the number of seconds of idle time required to activate a warning (buzzer beeps, indicator light flashes) at the beginning of the next move.

9.236 SetWarningBeforeMove

(Enable As Boolean)

Parameter:

Set True to activate or False to deactivate.

Description:

Sets the state of the WarningBeforeMove setting. If activated, the buzzer beeps and the indicator light flashes at the beginning of a move that is executed after the robot has sat idle for a time period equal to or greater than WarningAfterIdleTime.

9.237 SetWarningDuration

(Seconds As Short)

Parameter:

Sets the warning duration, in seconds.

Description:

Sets the duration of buzzer beeping and indicator light flashing at the beginning of a move that is executed after the robot has sat idle.

9.238 ShutDown

(Optional DisableMotors As Boolean)

Parameter:

DisableMotors—If True, then the motors will be disabled. If False, then the motors will be left enabled if they are already enabled, but motor cycle data will not be saved to the motor drives, as this requires the motors to be disabled. ShutDown should be called with DisableMotors=True at least periodically so the cycle data will be saved.

Description:

This method should be called immediately before the controlling application is terminated. The robot motors are turned off (DisableMotors = True), motor cycle data is saved to the motor drives, and the communication connection is closed. If the DLL is to be closed completely, PreClassTerminateCleanup should also be called. The ShutdownComplete event will be raised when ShutDown is finished.

9.239 StatusWindowShow

(Show As Boolean, Optional Title As String, Optional WindowTop As Short, Optional WindowLeft As Short)

Parameters:

Show—Specifies whether Status window should be shown or hidden.

Title—This optional parameter specifies the text of the title to be displayed on the Status window.

WindowTop—This optional parameter specifies the startup top position of the Status window. The default is zero (Windows controls the position).

WindowLeft—This optional parameter specifies the startup left position of the Status window. The default is zero (Windows controls the position).

Description:

Can be used to show or hide the Status window. If the Status window is shown prior to calling ShutDown, then the status of saving move data to the motor drives will be displayed on the Status window.

9.240 TeachButtonEnable

(State As Boolean, AutoTeachWindow As Boolean)

Parameters:

State—If True, then the teach button will be enabled. Otherwise, it is disabled.

AutoTeachWindow—Set to False—for internal use only.

Description:

Turns on the teach button indicator light and monitors the teach button. When the teach button is enabled and the operator presses it, the light will flash, the buzzer will sound and the TeachButtonPressed event will be raised. No teaching will occur if the AutoTeach window is not shown. It is the responsibility of the host software to perform any desired teaching operation or custom action.

9.241 TeachPendantGetSize

(Height As Integer, Width As Integer)

Parameters:

Height—Returns the height of the Teach Pendant window, in units of pixels.

Mode—Returns the width of the Teach Pendant window, in units of pixels.

Description:

Can be used to retrieve the size of the Teach Pendant Window.

9.242 TeachPendantShow

(Show As Boolean, Optional Mode As Short , Optional WindowTop As Short, Optional WindowLeft As Short)

Parameters:

Show—Specifies whether Teach Pendant window should be shown or hidden.

Mode—This optional parameter specifies whether the Teach Pendant is modal (1) or modeless (0).

WindowTop—This optional parameter specifies the startup top position of the Teach Pendant. The default is zero (TeachPendantTop Registry setting is used).

WindowLeft—This optional parameter specifies the startup left position of the Teach Pendant. The default is zero (TeachPendantLeft Registry setting is used).

Description:

Can be used to show or hide the Teach Pendant window. The user may not want the Teach Pendant to be shown at all times. The event "TeachPendantClosed" will fire when the Teach Pendant is closed.

9.243 TeachPointDelete

(Name As String)

Parameters:

Name—Specifies the name of the Teach Point to delete.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Use this method to delete the name and position data of the specified Teach Point from the "Teachpoints.ini" file.

9.244 TeachPointFindName

(Name As String, Position As Short) As Short

Parameters:

Name—Name of the Teach Point for which to be searched.

Position—Returned number associated with Teach Point.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function can be used to determine if a Teach Point exists with a given name. If the Teach Point exists, its number is returned. Each Teach Point has a name and a number by which it is defined.

9.245 TeachPointGetName

(Position As Short, Name As String) As Short

Parameters:

Position—Specifies the Teach Point number.

Name—Returns the text name of the Teach Point number specified.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Provides a method for looking up the text name of a Teach Point.

9.246 TeachPointGetValue

(Position As Short, Value() As String) As Short

Parameters:
Position—Specifies the Teach Point number.

Value()—Returns position data for the axes associated with the specified Teach Point number.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
Provides a method for looking up the positional data associated with a Teach Point.

9.247 TeachPointMoveRelativeTo

(PositionNameOrNumber As String, Offset() As Double, Velocity As Double, Acceleration As Double, WaitUntilDone As Boolean, TimeoutMsec As Integer, SendEventWhenMoveDone As Boolean = False, Optional ByRef Index As Byte) As Short

Parameters:
PositionNameOrNumber—Specifies the Teach Point to which the robot will move. The Teach Point can be specified by its name or by its number.

Offset()—Specifies joint offsets for the robot axes.

Velocity—Specified as percent of maximum.

Acceleration—Specified as percent of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the motors start to move. Either the function “MotorWaitForMoveDone” will have to be called once

for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped.

TimeoutMsec —Returns the time in milliseconds that the move will take to complete. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "TeachPointMoveToDone" event will be triggered when the move is done.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Moves the robot to the specified Teach Point, with each axis offset by the amount specified by the Offset() array.

9.248 TeachPointMoveRelativeToCartesian

(PositionNameOrNumber As String, Offset() As Double, CmdVel As Double, CmdAccel As Double, WaitUntilDone As Boolean, CalculatedMoveTimeMsec As Integer, SendEventWhenMoveDone As Boolean = False, Optional RefFrame As Short, Optional RefFrameRotate As Double, Optional ToolOffset As Double, Optional ByRef Index As Byte) As Short

Parameters:

PositionNameOrNumber—Specifies the Teach Point to which the robot will move, plus the specified offset. The Teach Point can be specified by its name or by its number.

Offset()—Specifies Cartesian offsets for the robot.

CmdVel—Specified as percent of maximum.

CmdAccel—Specified as percent of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the motors start to move. Either the function “MotorWaitForMoveDone” will have to be called once for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped.

CalculatedMoveTimeMsec —Returns the time in milliseconds that the move will take to complete. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "TeachPointMoveToDone" event will be triggered when the move is done.

RefFrame—This optional parameter can have the following values:

0 = World Reference Frame (relative to base of robot, +X = right, +Y = forward)

1 = Tool Reference Frame (relative to robot gripper, +X = right, +Y = forward)

2 = Rotated World Reference Frame (rotates the world reference frame by the number of degrees specified by RefFrameRotate parameter)

RefFrameRotate—Specifies a rotary offset, specified in degrees, for the gripper. This provides compensation for a gripper that is attached so that it does not face straight forward when the wrist is at zero.

ToolOffset—Specifies a linear offset from the center of the wrist to the center of the object being held by the gripper. The offset will be calculated from the center of the object in the gripper instead of the center of the wrist.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Moves the robot to the specified Teach Point, but offset by the amount specified by the Offset() array and applied based on the RefFrame, RefFrameRotate and ToolOffset parameters.

9.249 TeachPointMoveTo

(PositionNameOrNumber As String, Velocity As Double, Acceleration As Double, WaitUntilDone As Boolean, , Optional TimeoutMsec As Integer, Optional SendEventWhenMoveDone As Boolean, Optional ByRef Index As Byte) As Short

Parameters:

PositionNameOrNumber—Specifies the Teach Point to which the robot will move. The Teach Point can be specified by its name or by its number.

Velocity—Specified in percent of maximum.

Acceleration—Specified in percent of maximum.

WaitUntilDone—If set to “True”, execution will pause until motion is complete, or until an error has occurred. If WaitUntilDone is set to “False”, the function will return as soon as the motors start to move. Either the function “MotorWaitForMoveDone” will have to be called once for each motor, or the function “MotorCheckIfMotionDone” will have to be called repeatedly to verify each motor has stopped.

If this method is called multiple times before the robot has completed the previous moves, it will be loaded into the move buffer and will be executed as soon as the previous moves have finished. Preloading multiple moves into the move buffer in this manner will allow each subsequent move to be preloaded into the robot, which reduces the pause time between moves.

TimeoutMsec —Returns the time in milliseconds that the move will take. This value can be used by the calling application as a timeout period for the completion of the move.

SendEventWhenMoveDone—If set to “True”, the "TeachPointMoveToDone" event will be triggered when the move is done.

Index—Returns the move buffer index of this move. This can be used to correlate the move command with its associated MoveAbsoluteAllAxesDone event by comparing the move Index parameter to the event Index parameter.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Moves the robot to the specified Teach Point.

9.250 TeachPointMoveToLinearIncremental

(PositionNameOrNumber As String, Velocity As Double, Acceleration As Double, NumIncrements As Short) As Short

Parameters:

PositionNameOrNumber—Specifies the Teach Point to which the robot will move. The Teach Point can be specified by its name or by its number.

Velocity—Specified in percent of maximum.

Acceleration—Specified in percent of maximum.

NumIncrements—Specifies the number of incremental moves the robot will make on its way to the Teach Point.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

Moves the robot to the specified Teach Point. The move will be broken up into several smaller moves that cause the end of the robot arm to follow a linear path. The higher the value of NumIncrements, the closer the move will be to following the linear path, but the move will take more time.

9.251 TeachPointSetName

(Position As Short, Name As String) As Short

Parameters:
Position—Specifies the Teach Point number to be named.

Name—Specifies the name for the Teach Point. If the name is less than four characters long, it must contain at least one non-numeric character.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This function sets the name of a Teach Point. The name will only be in active memory, so “TeachPointsSave” must be called to save the new Teach Point name to file.

9.252 TeachPointSetValue

(Position As Short, Value() As Double) As Short

Parameters:
Position—Specifies the Teach Point number to be set.

Value()—A list of the new axis joint position values. Expressed in degrees or millimeters.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This function sets the positional values for a Teach Point. The values will only be in active memory, so “TeachPointsSave” must be called to save the new Teach Point values to file.

9.253 TeachPointsGetCount

(Count As Short, Optional Filename As String = "") As Short

Parameters:
Count—Returns the total number of teachpoints stored in the teachpoint file.

Filename—Specifies the name and path of the teachpoint file. If omitted, the default teachpoint file will be used.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This method retrieves the total number of teachpoints stored in the teachpoint file (see TeachPointsSetCount).

9.254 TeachPointsLoad

(Optional FileName As String) As Short

Parameters:
FileName—Specifies the path and name of the file containing the Teach Point data. If left blank, the file path specified in TeachPendant//Options/File Locations will be used.

Return Value:
Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:
This function reads a file containing Teach Point data. This must be done before the robot can be moved to Teach Points. This function is called by Initialize().

9.255 TeachPointsSave

(Optional FileName As String) As Short

Parameters:

FileName—Specifies the path and name of the file where the Teach Point data is to be saved. If left blank, the file path specified in TeachPendant//Options/File Locations will be used.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function saves the Teach Point data that is in active memory to a file. This must be done before the application is closed, or any new Teach Point data will be lost.

9.256 TeachPointsSetCount

(Count As Short, Optional Filename As String) As Short

Parameters:

Count—Specifies the total number of teachpoints stored in the teachpoint file. The default value is 100.

Filename—Optional parameter that specifies the name and location of the teachpoint file. If omitted, the default teachpoint file will be used.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method is used for adjusting the total number of teachpoints stored in the teachpoint file. If the number is reduced, some existing teachpoints may be lost.

9.257 UseDefaultGripperShutdownState

(UseDefault As Boolean)

Parameter:

UseDefault—If set to “True”, the gripper will be moved to the default gripper shutdown state during Shutdown. Otherwise, the gripper will be left in its current location.

Description:

This function specifies whether the default gripper shutdown state setting is to be used.

9.258 UseDefaultGripperState

(UseDefault As Boolean)

Parameter:

UseDefault—If set to “True”, the gripper will be moved to the default gripper state after homing during initialization. Otherwise, the gripper will be left in its current location.

Description:

This function specifies whether the default gripper state setting is to be used.

9.259 UseFileNotRegistry

(UseFile As Boolean, Filename As String) As Short

Parameters:

UseFile—Specifies whether to use a file.

Filename—Specifies the location of the file to use for storing settings.

Description:

This function directs the DLL to store settings in an INI file instead of in the Windows Registry. This allows multiple Windows User accounts to share settings, provided the file is placed in a public location.

9.260 VisionAlignAprilTag

(AprilTagID As Short, TeachPoint As String, ZOffset As Double, AlignRobot As Boolean, Velocity As Double, Acceleration As Double, AprilTagWidthMM As Double, ByRef AprilTagCoord As t2DCoord, ByRef mmPerPixel As Double, ByRef AprilTagAngle As Double, ByRef AprilTagNum As Byte, UseCurrentPosition As Boolean) As Short

Parameters:

AprilTagID—Specifies the ID encoded in the AprilTag being used for alignment

TeachPoint—Specifies the teach point that the robot will move to when searching for the AprilTag

ZOffset—Specifies the vertical offset from the teach point that the robot will move to for the initial AprilTag search

AlignRobot—Specifies whether the robot is move after locating the AprilTag so that the AprilTag is centered in the camera image

Velocity—Specifies the velocity for the move, expressed as a percentage of maximum

Acceleration—Specifies the acceleration for the move, expressed as a percentage of maximum

AprilTagWidthMM—Specifies the physical width, in millimeters, of the AprilTag being used for alignment

AprilTagCoord—Retrieves the X,Y coordinates of the AprilTag relative to the camera in units of pixels. This is the position read prior to aligning the robot to the AprilTag

mmPerPixel—Retrieves the factor for converting between millimeters and pixels

AprilTagAngle—Retrieves the angle of the AprilTag relative to the camera in units of degrees.

AprilTagNum—Retrieves the camera internal index number assigned to the AprilTag

UseCurrentPosition—If set to True, then the robot will not be commanded to move to the teach point, and the AprilTag search will be conducted from the current position of the robot instead

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function measures the position of an AprilTag using the integrated vision. The measurement is conducted in the following two steps: the robot moves to the Z offset from the teach point and measures the AprilTag position, then the robot moves down to the height of the teach point but with the camera centered on the AprilTag. This function is used by VisionLocateAprilTag.

9.261 VisionAlignRobot1

(AprilTagName As Short, Velocity As Double, Acceleration As Double, VisionOffsetEnable As Boolean) As Short

Parameters:

AprilTagName—Specifies the name of the AprilTag record to use from the vision calibration file

Velocity—Specifies the velocity for the move, expressed as a percentage of maximum

Acceleration—Specifies the acceleration for the move, expressed as a percentage of maximum

VisionOffsetEnable—Specifies whether to enable the vision offset after completing the alignment operation

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function uses the integrated vision to measure the current position of an AprilTag that has already been configured and stored in the vision calibration file. The current AprilTag position is then applied, along with its zero position from the calibration file, as the vision offset (see SetVisionOffset function). The vision offset is then enabled (see EnableVisionOffset). This is the highest-level vision function, and should be used in conjunction with EnableVisionOffset(False) for standard applications that need vision-assisted position correction. This function should be used in scenarios where the AprilTag is in the center of the nest of the instrument being accessed by the robot. If the robot will be accessing multiple locations based on the vision offset, then it is recommended to use VisionAlignRobot2 instead.

9.262 VisionAlignRobot2

(AprilTagName1 As Short, AprilTagName2 As Short, Velocity As Double, Acceleration As Double, VisionOffsetEnable As Boolean) As Short

Parameters:

AprilTagName1—Specifies the name of the first AprilTag record to use from the vision calibration file

AprilTagName2—Specifies the name of the second AprilTag record to use from the vision calibration file

Velocity—Specifies the velocity for the move, expressed as a percentage of maximum

Acceleration—Specifies the acceleration for the move, expressed as a percentage of maximum

VisionOffsetEnable—Specifies whether to enable the vision offset after completing the alignment operation

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function uses the integrated vision to measure the current position of a pair of AprilTags that have already been configured and stored in the vision calibration file. The current AprilTag positions are then applied, along with its zero positions from the calibration file, as the vision offset (see VisionCombineTwoAprilTagPositions and SetVisionOffset functions). The vision offset is then enabled (see EnableVisionOffset). This is the highest-level vision function, and should be used in conjunction with EnableVisionOffset(False) for standard applications that need vision-assisted position correction. Using two AprilTags that are as far apart from each other as possible improves the angular measurement of the system. It is recommended to use this function instead of VisionAlignRobot1 if the robot will be accessing multiple instrument nests based on the vision offset.

9.263 VisionCalculateAprilTagZDistance

(AprilTagWidth As Double, mmPerPixel, ByRef ZDistance As Double) As Short

Parameters:

AprilTagWidth—Specifies the physical width of the AprilTag, in millimeters

mmPerPixel—Specifies the factor for converting between millimeters and pixels. This value can be measured using CameraGetAprilTagPosition or VisionAlignAprilTag

ZDistance—Retrieves the calculated vertical distance from the AprilTag to the camera lens

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function calculates the vertical distance from an AprilTag to the lens of the integrated camera based on the pixel size of the AprilTag in the camera image and the actual size of the AprilTag in millimeters.

9.264 VisionCombineTwoAprilTagPositions

(AprilTag1 As tAprilTag, AprilTag2 As tAprilTag, ByRef zeroRef3DCoord As t3DCoord, ByRef zeroRefQuaternion As tQuaternion, ByRef offsetRef3DCoord As t3DCoord, ByRef offsetRefQuaternion As tQuaternion)

Parameters:

AprilTag1—Specifies the first AprilTag calibration parameters as a tAprilTag structure that contains the following values: ID, Width, ZOffset, TeachPoint, ZeroX, ZeroY, ZeroTheta, CurrX, CurrY, CurrTheta

AprilTag2—Specifies the second AprilTag calibration parameters as a tAprilTag structure that contains the following values: ID, Width, ZOffset, TeachPoint, ZeroX, ZeroY, ZeroTheta, CurrX, CurrY, CurrTheta

zeroRef3DCoord—Retrieves the X,Y,Z coordinates of the resulting combined zero position of the two AprilTags

zeroRefQuaternion—Retrieves the quaternion rotary coordinates of the resulting combined zero position of the two AprilTags

offsetRef3DCoord—Retrieves the X,Y,Z coordinates of the resulting combined offset (current) position of the two AprilTags

offsetRefQuaternion—Retrieves the quaternion rotary coordinates of the resulting combined offset (current) position of the two AprilTags

Description:

This function takes two AprilTag position records and combines their positions into a single position that can then be used in functions that accept only one AprilTag position.

9.265 VisionConvertCameraPositionToRobotWorldCoordinates

(RobotJointPos() As Double, AprilTagCoord As t2DCoord, AprilTagWidth As Double, mmPerPixel As Double, AprilTagAngle As Double, ByRef AprilTagWorldCoord() As Double) As Short

Parameters:

RobotJointPos()—Specifies the positions of the robot joints

AprilTagCoord—Specifies the X,Y position of the AprilTag relative to the center of the camera image, in units of pixels

AprilTagWidth—Specifies the physical width of the AprilTag in millimeters

mmPerPixel—Specifies the factor for converting between millimeters and pixels. This value can be measured using CameraGetAprilTagPosition or VisionAlignAprilTag

AprilTagAngle—Specifies the angle of the AprilTag relative to the camera image, in units of degrees

AprilTagWorldCoord()—Retrieves the physical position of the AprilTag within the world coordinate system of the robot

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function calculates the physical position of an AprilTag based on the position of the robot joints and the position of the AprilTag in the integrated vision camera

9.266 VisionLocateAprilTag

(AprilTagName As String, Velocity As Double, Acceleration As Double, ByRef AprilTagWorldCoord() As Double, ByRef RobotJointPos() As Double, UseCurrentPosition As Boolean) As Short

Parameters:

AprilTagName—Specifies the name of the AprilTag record to use from the vision calibration file

Velocity—Specifies the velocity for the move, expressed as a percentage of maximum

Acceleration—Specifies the acceleration for the move, expressed as a percentage of maximum

AprilTagWorldCoord()—Retrieves the physical position of the AprilTag within the robot world coordinate frame

RobotJointPos()—Retrieves the robot joint positions with the AprilTag centered in the camera

UseCurrentPosition—If set to True, then the search for the AprilTag starts at the current position instead of using the teach point from the AprilTag calibration record

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function calls VisionAlignAprilTag twice: once from the ZOffset position stored in the AprilTag calibration record, and a second time from the height of the teach point stored in the AprilTag calibration record. The location of the AprilTag within the robot world coordinate frame is then provided. This function is used by VisionAlignRobot1 and VisionAlignRobot2.

9.267 WaitForInput

(Axis As Short, InputNumber As Short, State As Short, TimeoutMsec As Integer) As Short

Parameters:

Axis—Specifies the motor.

InputNumber—Specifies the input on the motor specified.

State—Specifies the state (1 or 0) for which to wait

TimeoutMsec—Specifies the number of milliseconds to wait before timing out.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This function will pause execution until the specified input changes to the desired state, or until the timeout has occurred.

9.268 WarningIdleStartTimeUpdate

Description:

If WarningBeforeMove is enabled (Teach Pendant/Options/Miscellaneous or Windows Registry), the buzzer will sound, and the RGB indicator light will flash prior to a move command if the robot has been idle for longer than the time specified by WarningAfterIdleTime. If the host software provides the operator with the ability to move the robot manually, it may be desired to override the warning for moves that are initiated manually. Call this method to restart the idle time counter and avoid the warning for a following manual move command.

9.269 WriteStringToFile

(WriteString As String, FileName As String, CleanFile As Boolean)

Parameters:

WriteString—Specifies text string to be written to file.

FileName—Specifies the path and name of the text file.

CleanFile—Specifies whether the data in the file is to be deleted prior to writing data to the file.

Description:

This is a non-robot-specific function provided as an extra utility for writing data to a text file.

9.270 WriteStringToINIFile

(Filename As String, Heading As String, Item As String, Value As String) As Short

Parameters:

Filename—Specifies the path and name of the INI file.

Heading—Specifies the section under which the value is to be written.

Item—Specifies the name of the value to be written.

Value—Specifies the value of the item being written.

Return Value:

Will return zero if successful, or an error code listed above, based on type of error that occurs.

Description:

This method is used for reading the value of an item in an INI file.

10.0 Descriptions of Events

10.1 BarcodeReaderData

Description:

Triggered when the barcode reader located in the robot gripper sends a response to a command. For example, if BarcodeRead() is called, and a barcode is read successfully, then the barcode data will be reported via this event.

10.2 InitializationComplete

Description:

This event is raised when the Initialize() function is finished. The success of initialization can be determined by calling IsInitialized().

10.3 InitializationStarted

Description:

This event is raised at the beginning of the Initialize() function.

10.4 InputChangedState

Parameter:

InputData As tInputEventData—Reports the associated axis and the states of all inputs for that axis. The tInputEventData type has the following format:

tInputEventData.Axis = Input axis number (motor number), Short
tInputEventData.State() = States of inputs, Boolean

Description:

This event is triggered when an input changes states. The polling period is normally 500 milliseconds, but is decreased to 100 milliseconds when the Teach button is active.

10.5 MaintenanceRequired

Parameter:

Axis As Short—Specifies the axis that requires maintenance.

Description:

The total number of motion cycles for each axis is logged in the Windows Registry. When this value exceeds and even multiple of the maintenance interval, this event is triggered, and will continue to be triggered each time the robot is initialized. Once maintenance has been performed, call the MaintenancePerformed() method to reset the flag. Please refer to the User's Manual for instructions on performing robot maintenance.

10.6 MotorPositions

Parameter:

Position()—Reports the encoder positions of all axes.

Description:

This event is triggered every 500 milliseconds unless none of the motor positions have changed since the last event.

10.7 MoveAbsoluteAllAxesDone

Parameter:

Index As Byte—Specifies the move buffer array index of the move that finished

Description:

Triggered when the robot completes a move commanded by MoveAbsoluteAllAxes(), with the WaitUntilDone parameter set to False, and the SendEventWhenMoveDone parameter set to True.

10.8 MoveAbsoluteSingleAxisDone

Parameter:

Index As Byte—Specifies the move buffer array index of the move that finished

Description:

Triggered when the robot completes a move commanded by MoveAbsoluteSingleAxis(), with the WaitUntilDone parameter set to False, and the SendEventWhenMoveDone parameter set to True.

10.9 MoveRelativeSingleAxisDone

Parameter:

Index As Byte—Specifies the move buffer array index of the move that finished

Description:

Triggered when the robot completes a move commanded by MoveRelativeSingleAxis(), with the WaitUntilDone parameter set to False, and the SendEventWhenMoveDone parameter set to True.

10.10 RobotError

Parameter:

ErrorCode As Short—Specifies the error occurred.

Description:

Triggered when a robot error occurs.

10.11 ScriptDone

Parameter:

SensorError As Short—Returns an error code if an error occurred.

Description:

Triggered when the robot completes all actions commanded by ScriptRun() with the WaitUntilDone parameter set to False and the SendEventWhenMoveDone parameter set to True.

10.12 ScriptEditorClosed

Description:

Triggered when the Sequence Editor window is closed.

10.13 ScriptError

Parameters:

ErrorCode As String—Returns the error code associated with the script error.

Description:

This event is raised when a script terminates due to an error.

10.14 ScriptRunning

Parameters:

NestedScript() As String—An array containing the names of the scripts in the current call stack.

NestedScriptLine() As Short—An array containing the current line being executed in each script in the current call stack.

NestCount As Short—The number of scripts deep that the point of execution is nested.

Description:

This event is raised during the execution of a script, each time a new line is about to be executed. The parameter values can be passed into ScriptResumeNested in order to resume after an error.

10.15 SetOutputDone

Description:

Triggered when a call to SetOutput() finishes, and SendEventWhenMoveDone parameter is set to True.

10.16 ShutdownComplete

Description:

This event is raised when the ShutDown() function is finished.

10.17 TeachButtonPressed

Parameters:

Enabled—If True, then the teach button function is currently in an enabled state, which means the indicator light will flash and the buzzer will sound.

AutoTeachWindow—If True, then the AutoTeach window is displayed currently.

Description:

This event is triggered when the teach button is pressed by the operator.

10.18 TeachPendantClosed

Description:

Triggered when the Teach Pendant window is closed.

10.19 TeachPointMoveRelativeToDone

Description:

Triggered when the robot completes all moves commanded by TeachPointMoveRelativeTo(), with the WaitUntilDone parameter set to False, and the SendEventWhenMoveDone parameter set to True.

10.20 TeachPointMoveToDone

Description:

Triggered when the robot completes all moves commanded by TeachPointMoveTo(), with the WaitUntilDone parameter set to False, and the SendEventWhenMoveDone parameter set to True.

Revision History

Revision	Date	Line	Description	SW Version
1	6/24/2020	All	Initial Release	1.1.0
2	7/27/2020	9.129	Added MeasuredStackHeightMM parameter to	1.1.1
			RemovePlateFromPitchStack	
		9.129	Added to YOffset parameter description	1.1.2
		9.135	Added YOffset & FlipWrist parameters to ScanStack	
		9.143	Added YOffset & FlipWrist parameters to SCANSTACK	
3	8/13/2020		Added Optional Index parameter to MoveAbsoluteSingleAxis	1.1.3
			MoveRelativeSingleAxis, ServoGripperOpen, Jog,	
			MotorsMoveJoint, MoveRelativeCartesian,	
			TeachPointMoveRelativeToCartesian, MoveAbsoluteAllAxes,	
			TeachPointMoveTo, TeachPointMoveRelativeTo,	
			MoveToCartesian, MoveToArrayPoint and	
			MotorsMoveLinear (lines 9.83, 101, 102, 104, 106, 110, 111,	
			112, 113, 164, 191, 192 & 193)	
4	9/24/2020	6.6.1	Added LogCANData, CANDataFile, LogCANDataNodeID,	1.1.5
			LogMovePathQueueErrors & MovePathQueueErrorDataFile	
			to registry settings.	
		8.0	Added ErrorCode(60)	
		9.31	Added GetAvailableCANDevices function	
		9.32	Added GetBarcodeReaderSerialPort function	
		9.36	Added GetCANDevice function	
		9.75	Added GetWarningAfterIdleTime function	
		9.76	Added GetWarningBeforeMove function	
		9.77	Added GetWarningDuration function	
		9.173	Added SetBarcodeReaderSerialPort function	
		9.174	Added SetCANDevice function	
		9.191	Added SetWarningAfterIdleTime function	
		9.192	Added SetWarningBeforeMove function	

Revision	Date	Line	Description	SW Version
		9.193	Added SetWarningDuration function	
		9.194	Added DisableMotors parameter to ShutDown function	
5	10/22/2020	6.6.1	Added Settings/SaveMoveDataOnShutDown registry setting	1.1.6
		9.65	Added GetSaveMoveDataOnShutDown function	
		9.141	Added SaveMoveDataToDrives function	
		9.190	Added SetSaveMoveDataOnShutDown function	
6	3/24/2021	9.191	Added SetSaveMoveDataOnShutdownOverride function	1.1.11
		9.199	Added StatusWindowShow function	
7	3/26/2021	9.119	Added RefFrame, RefFrameRotate, ToolOffset parameters	1.1.12
			To MoveToArrayPoint function	
8	4/2/2021	8.0	Added error codes 69-74 for MoveToArrayPoint	1.1.16
9	6/29/2021	9.172	Added ServoGripperInitializeNoWait function	1.1.19
		9.151	Added COMMENT command to ScriptOperationRun	1.1.22
10	9/3/2021	6.6.1	Added MaintainGripAfterShutdown registry setting	1.1.29
		9.38	Added GetDefaultGripperShutdownState function	
		9.59	Added GetMaintainGripAfterShutdown function	
		9.188	Added SetMaintainGripAfterShutdown function	
11	9/7/2021	6.6.1	Added UseDefaultGripperShutdownState	1.1.30
		9.76	Added GetUsingDefaultGripperShutdownState function	
		9.222	Added UseDefaultGripperShutdownState function	
12	10/8/2021	9.124	Documented grip height Z offset for PlacePlateInHotel	
		9.137	Documented grip height Z offset for RemovePlateFromHotel	
13	11/12/2021	9.59	Added GetJointMoveDirection	1.1.33
		9.63	Added GetMovePathMode	
14	12/3/2021	6.6.1	Added SequenceEditorHeight, SequenceEditorWidth,	1.1.34
			SequenceEditorTop, SequenceEditorLeft,	
			TeachPendantTop and Teach PendantLeft Registry settings	
		9.5	AutoTeachWindowShow Windows controlled for Top & Left 0	
		9.152	ScriptEditorShow uses Registry settings for Top & Left 0	

Revision	Date	Line	Description	SW Version
		9.206	StatusWindowShow Windows controlled for Top & Left 0	
		9.209	TeachPendantShow uses Registry settings for Top & Left 0	
15	6/8/2022	2.7	Changed from .NET 4.0 to .NET 4.6.1	1.1.43
		6.2.3	Installation directory changed from Peak Analysis & Automation to Peak Robotics, Inc	
		6.6.1	EthernetPort, InterfaceType, RobotIPAddress and Vision Offset Registry settings	
		8.0	New ErrorCode(395)	
		9.2	Added ApplyVisionOffset	
		9.31	Added EnableVisionOffset	
		9.50	Added GetEthernetPort	
		9.62	Added GetInterfaceType	
		9.73	Added GetRobotIPAddress	
		9.86	Added GetVisionOffsetEnabled	
		9.119	Added IgnoreVisionOffset parameter to MotorsMoveJoint	
		9.120	Added IgnoreVisionOffset parameter to MotorsMoveLinear	
		9.128	No vision offset applied to MoveRelativeCartesian if MoveToLimit = True	
		9.151	Added RemoveVisionOffset	
		9.190	Installation directory changed from Peak Analysis & Automation to Peak Robotics, Inc	
		9.197	Added SetEthernetPort	
		9.198	Added SetInterfaceType	
		9.205	Added SetRobotIPAddress	
		9.212	Added SetVisionOffset	
16	8/31/2022	9.93	Added UseCachedDriveParams & CachedDriveParams parameters to Initialize()	1.1.51
		9.114	Added LEDIndicatorOn parameter to MotorEnable()	
17	9/7/2022	6.6.1.1	New Windows Registry section DriveParamCache	1.1.52
		8.0	Error Code 100	

Revision	Date	Line	Description	SW Version
		9.93	If Initialize() CachedDriveParams parameter is passed in with empty values, then cached values will be stored in the Windows Registry	
18	3/20/2023	9.93	Added Initialize() overload without UseCachedDriveParams, CachedDriveParams	1.1.55
		9.132	Added FirstMoveJointPathMode, PreloadMoves parameters to PlacePlateInHotel and PlacePlateInHotelResume. Created overloads without the new parameters	
		9.145	Added FirstMoveJointPathMode, PreloadMoves parameters to RemovePlateFromHotel and RemovePlateFromHotelResume. Created overloads without the new parameters	
		9.154	Added FirstMoveJointPathMode, PreloadMoves parameters to ScanHotel. Created overload without the new parameters	
19	7/21/2023	6.6.1	Added Files\VisionCalibrationFile setting to Registry	1.2.0
		8.0	Added error codes 120, 230, 235, 394, 396-399	
		8.0	Added CAN error codes 75-79	
		9.3	Added AprilTagDeleteCalibrationFromFile	
		9.4	Added AprilTagReadCalibrationFromFile	
		9.5	Added AprilTagsGetNamesFromFile	
		9.6	Added AprilTagWriteCalibrationToFile	
		9.19	Added CameraCaptureImage	
		9.20	AddedCameraGetAprilTag	
		9.21	Added CameraGetAprilTagCornerCoordinates	
		9.22	Added CameraGetError	
		9.23	Added CameraGetAprilTagID	
		9.24	Added CameraGetAprilTagNum	
		9.25	Added CameraGetAprilTagPosition	
		9.26	Added CameraGetImage (three overloads)	
		9.29	Added CameraGetMaximumResolution	
		9.30	Added CameraGetWindowing	
		9.31	Added CameraSetExposureTime	
		9.40	Added ConvertJointPositionToCartesian overload	

Revision	Date	Line	Description	SW Version
		9.56	Added GetCameraPresent	
		9.57	Added GetCameraWristOffset	
		9.66	Added GetDefaultVisionCalibrationFile	
		9.184	Added VISION script operations	
		9.218	Added SetDefaultVisionCalibrationFile	
		9.260	Added VisionAlignAprilTag	
		9.261	Added VisionAlignRobot1	
		9.262	Added VisionAlignRobot2	
		9.263	Added VisionCalculateAprilTagZDistance	
		9.264	Added VisionCombineTwoAprilTagPositions	
		9.265	Added VisionConvertCameraPositionToRobotWorldCoords	
		9.266	Added VisionLocateAprilTag	
20	2/27/2024	9.184	Added SCANSTACK to ScriptOperationRun	1.2.3