MLX200
SOFTWARE AND OPERATION
USERS MANUAL

VERSION 2.1

Upon receipt of the product and prior to initial operation, read these instructions thoroughly, and retain for future reference.

MOTOMAN INSTRUCTIONS

MOTOMAN-□□□ INSTRUCTIONS
MLX200 HARDWARE INSTALLATION AND SOFTWARE UPGRADE

Part Number: 168542-1CD
Revision: 7

YASKAWA
Innovation

MLX200 Software and
Operations

168542-1CD

> ⓘ **MANDATORY**
>
> This manual explains the various components of the MLX200 application system and general operations. Read this manual carefully and be sure to understand its contents before operating the MLX200 application.

> ⚠ **CAUTION**
>
> - Some drawings in this manual are shown with the protective covers or shields removed for clarity. Be sure all covers and shields are replaced before operating this product.
>
> - The drawings and photos in this manual are representative examples and differences may exist between them and the delivered product.
>
> - YASKAWA may modify this model without notice when necessary due to product improvements, modifications, or changes in specifications. If such modification is made, the manual number will also be revised.
>
> - If your copy of the manual is damaged or lost, contact a YASKAWA representative to order a new copy. The representatives are listed on the back cover. Be sure to tell the representative the manual number listed on the front cover.
>
> - YASKAWA is not responsible for incidents arising from unauthorized modification of its products. Unauthorized modification voids your product's warranty.

We suggest that you obtain and review a copy of the ANSI/RIA National Safety Standard for Industrial Robots and Robot Systems (ANSI/RIA R15.06-2012). You can obtain this document from the Robotic Industries Association (RIA) at the following address:

Robotic Industries Association
900 Victors Way
P.O. Box 3724
Ann Arbor, Michigan 48106
TEL: (734) 994-6088
FAX: (734) 994-3338
www.roboticsonline.com

Ultimately, well-trained personnel are the best safeguard against accidents and damage that can result from improper operation of the equipment. The customer is responsible for providing adequately trained personnel to operate, program, and maintain the equipment. NEVER ALLOW UNTRAINED PERSONNEL TO OPERATE, PROGRAM, OR REPAIR THE EQUIPMENT!

We recommend approved Yaskawa training courses for all personnel involved with the operation, programming, or repair of the equipment.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications.

Notes for Safe Operation

Read this manual carefully before installation, operation, maintenance, or inspection of the MLX200.

In this manual, the Notes for Safe Operation are classified as "DANGER" "WARNING", "CAUTION", "MANDATORY", or "PROHIBITED".

⚠️	**DANGER**	Indicates an imminent hazardous situation which, if not avoided, could result in death or serious injury to personnel.
⚠️	**WARNING**	Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury to personnel.
⚠️	**CAUTION**	Indicates a potentially hazardous situation which, if not avoided, could result in minor or moderate injury to personnel and damage to equipment. It may also be used to alert against unsafe practices.
🛈	**MANDATORY**	Always be sure to follow explicitly the items listed under this heading.
🚫	**PROHIBITED**	Must never be performed.

Even items described as "CAUTION" may result in a serious accident in some situations.

At any rate, be sure to follow these important items.

> **NOTE**
> To ensure safe and efficient operation at all times, be sure to follow all instructions, even if not designated as "DANGER", "CAUTION" and "WARNING".

168542-1CD

MLX200 Software and
Operations

⚠ WARNING

- Confirm that no person is present in the P-point maximum envelope of the manipulator and that you are in a safe location before:
  - Turning ON the MLX200 Control Module power
  - Moving the manipulator with programming pendant or MLX200 Control Module HMI
  - Running the system in the check mode
  - Performing automatic operations

Injury may result if anyone enters the P-point maximum envelope of the manipulator during operation. Always press the [EMERGENCY STOP] button immediately if there are problems. The [EMERGENCY STOP] button is located on the right of the programming pendant.

- Observe the following precautions when performing teaching operations within the P-point maximum envelope of the manipulator:
  - View the manipulator from the front whenever possible.
  - Always follow the predetermined operating procedure.
  - Ensure that you have a safe place to retreat in case of emergency.

Improper or unintended manipulator operation may result in injury.

- Before operating the manipulator, check that servo power is turned OFF when the [EMERGENCY STOP] button on programming pendant is pressed.
  When the servo power is turned OFF, the SERVO ON INDICATOR on the programming pendant or MLX200 Control Module HMI is turned OFF.

Injury or damage to machinery may result if the Emergency Stop circuit cannot stop the positioner during an emergency. The positioner should not be used if the [EMERGENCY STOP] buttons do not function.

Fig. 1: EMERGENCY STOP Button

- Release the [EMERGENCY STOP] button (refer to *Fig. 2*). Once this button is released, clear the cell of all items which could interfere with the operation of the positioner. Then turn servo power ON.

Injury may result from unintentional or unexpected positioner motion.

Fig. 2: Release of EMERGENCY STOP Button

TURN

> ## ⚠️ CAUTION
>
> • Perform the following inspection procedures prior to conducting manipulator teaching. If problems are found, repair them immediately, and be sure that all other necessary processing has been performed.
> -Check for problems in manipulator movement.
> -Check for damage to insulation and sheathing of external wires.
>
> The optional programming pendant can be damaged if it is left in the P-point maximum envelope of the manipulator's work area, on the floor, or near fixtures.
>
> • Read and understand the Explanation of the Warning Labels before operating the manipulator.

Definition of Terms Used Often in This Manual

The MOTOMAN is the YASKAWA industrial robot product.

The MOTOMAN usually consists of the manipulator, a control module, a programming pendant, and supply cables.

In this manual, the equipment is designated as follows.

Equipment	Manual Designation
MLX200 Control Module	MLX200
MLX200 Programming Pendant	Programming pendant

Descriptions of the programming pendant keys, buttons, and displays are shown as follows:

Equipment		Manual Designation
Programming Pendant	Character Keys/ Symbol Keys	The keys which have characters or its symbol printed on them are denoted with []. ex. [ENTER]
	Axis Keys / Numeric Keys	[Axis Key] and [Numeric Key] are generic names for the keys for axis operation and number input.
	Keys pressed simultaneously	When two keys are to be pressed simultaneously, the keys are shown with a "+" sign between them, ex. [SHIFT]+[COORD]
	Displays	The menu displayed in the programming pendant is denoted with {}. ex. {JOB}

Description of the Operation Procedure

In the explanation of the operation procedure, the expression "Select • • • "
means that the cursor is moved to the object item and the SELECT key is
pressed, or that the item is directly selected by touching the screen.

Registered Trademark

In this manual, names of companies, corporations, or products are
trademarks, registered trademarks, or brand names for each company or
corporation. The indications of (R) and ^{TM} are omitted.

Explanation of Warning Labels

The following warning labels are attached to the manipulator.

Fully comply with the precautions on the warning labels.

⚠ WARNING

- The label described below is attached to the manipulator. Observe
the precautions on the warning labels.

- Failure to observe this caution may result in injury or damage to
equipment.

⚠ WARNING

Moving parts
may cause
injury

⚠ WARNING

Do not enter
robot
work area.

Refer to the manipulator manual for the warning label location.

168542-1CD

Table of Contents

168542-1CD

168542-1CD

168542-1CD

1   Introduction

The MLX200 Control Module from Yaskawa provides an easy way to develop PLC-based robotic solutions. Programming of the MLX200 is performed entirely using RSLogix 5000 from Rockwell Automation through the use of a set of RSLogix 5000 Add-on Instructions (AOIs) for Robot Motion and Configuration. These instructions are compatible with the ControlLogix, CompactLogix and GuardLogix families of Programmable Automation Controllers (PAC) from Rockwell Automation. This guide will provide a step-by-step description of how to configure, develop, deploy, and maintain applications using MLX200 Control Module. *Table 1-1 "MLX200 Control Panel Components and Terminology"* shows the various components that are contained within the MLX200 package.

Table 1-1: MLX200 Control Panel Components and Terminology

MLX200 Control Module Development Software (MLX-D)	Add on instructions, data types, HMI screens (referred to as MLX-HMI), examples, etc. for RSLogix 5000 and FTView. An MLX200 user programs their application using the MLX-D functionality within the RSLogix 5000 environment.
MLX200 Control Module PLC Interface Software (MLX-R)	Robot control firmware that is resident on the MLX200.
MLX200 Drive Panel	Hardware Drive Panel containing Servo Drives and Safety Circuit for a given Robot Model.
MLX200 Control Module	Stand-alone module that contains the MLX-R firmware. This module will be mounted on the MLX200 Drive Panel.

1.1   Requirements

1.1.1   Rockwell Automation PAC/PLC requirements for the MLX200:

- Hardware Requirements

– 1756 ControlLogix Chassis and Power supply

– 1756 ControlLogix/GuardLogix safety controller with 1756-ENBT EtherNet/IP module

– 1769 CompactLogix controller with built-in Ethernet/IP.

– 2 MB of available memory on PLC

- Development PC Requirements

– Windows XP or 7 32/64 bit on a PC with 4 GB RAM

- Software Requirements

– RSLogix 5000 V20

– FTView ME Station version 6.1, 30 Display Activation (note: MLX-HMI has 30 screens - a larger activation may be necessary if integrating into a larger HMI)

– Fuji's V-SFT development environment (version 5.4.33.0) for optional MLX-HMI for Fuji Monitouch panels.

1.1.2   RSLogix 5000 files included with the MLX200 Control Module:

- MLX200_ControlLogix.ACD - preconfigured project for use with ControlLogix systems

- MLX200_CompactLogix.ACD - preconfigured project for use with CompactLogix systems

- MLX200_Import.L5X - skeleton application for importing MLX200 AOIs and User Defined Types (UDTs)

- MLX200Communications_[0-3].L5X - RSLogix task for MLX200 communications

- MLX200HMI.APA - MLX200 HMI Project File

- MLxConveyorSync_[0-3].L5X - conveyor update task (used only for conveyor tracking)

- HMIUpdates.L5X - a RSLogix task that is needed to communicate with the MLX200 HMI

1.2   System Layout for the MLX200 Control Module

⚠ CAUTION
• The Ethernet cable between the MLX Control Module and Drive Panel must be a Shielded Twisted Pair Cat 5E cable to prevent noise-related issues that can vary greatly due to location and system setup. All other cables can be standard Ethernet. • The Ethernet Switch should not be used to handle any other network communications besides those shown in the layout. Other network traffic could cause poor performance or loss of communications.

Fig. 1-1 "MLX200 Connection System Layout" on page 1-3 shows an overall diagram of how the system should be connected. Ethernet port 1 on the MLX200 Control Module should be connected directly to the servo drive panel. Ethernet port 2 should be connected through an Ethernet switch to the PC running RSLogix as well as the CompactLogix or ControlLogix system.

168542-1CD

Fig. 1-1: MLX200 Connection System Layout
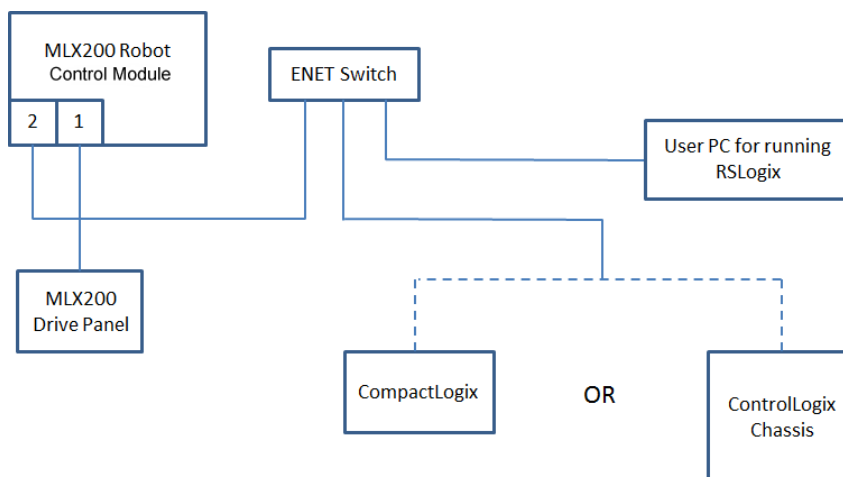
| NOTE | Multiple MLX200 Control Modules can be connected to a single Logix PLC. *Fig.1-1 "MLX200 Connection System Layout"* shows a single MLX200 Control Module layout. |

1.3   Customer Support Information

If you need assistance with any aspect of your MLX200 Software and Operations please contact YASKAWA Customer Support at the following 24-hour telephone number:

<div style="border:1px solid black; display:inline-block; padding:8px;">

(937) 847-3200

</div>

For **routine** technical inquiries, you can also contact YASKAWA Customer Support at the following e-mail address:

techsupport@motoman.com

When using e-mail to contact YASKAWA Customer Support, please provide a detailed description of your issue, along with complete contact information. Please allow approximately 24 to 36 hours for a response to your inquiry.

NOTE	Please use e-mail for **routine** inquiries only. If you have an urgent or emergency need for service, replacement parts, or information, you must contact Yaskawa Customer Support at the telephone number shown above.

Please have the following information ready before you call:

• System	MLX200
• Robots	
• Positioner	
• Primary Application	
• Software Version	Access this information on the Status Display screen of the MLX200 Control Module. Refer to *Fig. 7-1 "Status Display of MLX200 Control Module" on page 7-1* for information on connecting to the Status Display screen
• Robot Serial Number	Located on the robot data plate
• Robot Sales Order Number	Located on the Control Module data plate

2   System Configuration

This section of the guide will provide simple steps for connecting and configuring each component in the MLX200 system.

2.1   MLX200 Control Module

The MLX200 Control Module will come attached to the MLX200 Drive Panel. The module comes with two RJ-45 ports (*Fig.2-1 "Ethernet Ports"*) that are used for EtherCAT and Ethernet communications. The module can be powered on by turning on power to the Drive Panel (*Fig.2-2 "Power Indicators"*). *Table 2-1 "MLX200 Control Module LED Indicators" on page 2-6* shows the LED indicators on the MLX Control Module.

Fig. 2-1: Ethernet Ports

Fig. 2-2: Power Indicators

168542-1CD

2   System Configuration
2.1 MLX200 Control Module
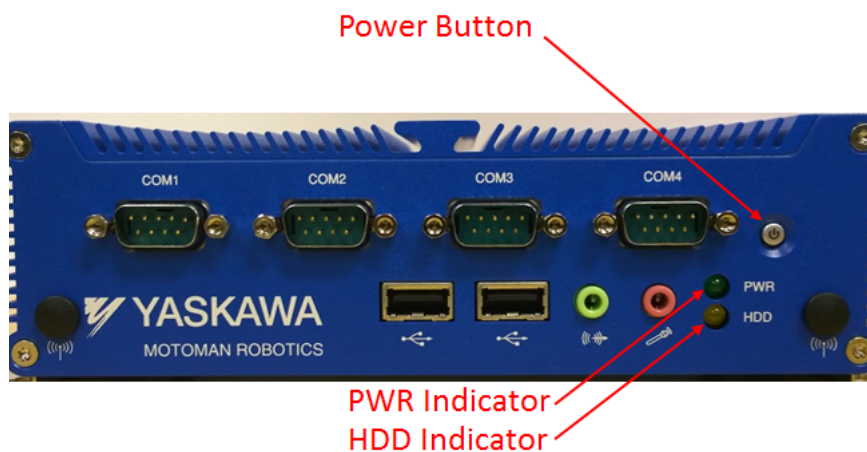
Table 2-1: MLX200 Control Module LED Indicators

LED	Color	State	Meaning
Power Button	None	Off	Power not connected to MLX200 Control Module
	Red	Solid	Power connected, but MLX200 Control Module is turned off.
	Blue	Solid	Power connected MLX200 Control Module running.
PWR	None	Off	Power not connected to MLX200 Control Module
	Green	Solid	Power connected and MLX200 Control Module running
HDD	Orange	Blinking	Indicates Hard Disk activity
Ethernet 1 (EtherCAT)	Green	Off	Link Inactive
		Solid	Link Active
	Orange	Off	Link Inactive
		Blinking	Link Active and data being transmitted.
Ethernet 2 (Ethernet/IP) (TCP/IP)	Green	Off	Link Inactive
		Solid	Link Active
	Orange	Off	Link Inactive
		Blinking	Link Active and data being transmitted.

NOTE   The Ethernet LEDs will show "Link Inactive" if the device they are connected to is not powered on.

168542-1CD

2.2   Configuring RSLogix Project

Once an MLX200 system has been connected, the next step is to start programming it from RSLogix 5000. This section describes how to setup an RSLogix project for MLX200 application development. There are two main methods for doing this:

- **Using pre-configured RSLogix projects.** This method is the simplest way to get started using MLX200 and should be used if a new application project is going to be developed.

- **Importing the MLX200 AOIs and UDTs.** This method involves importing the necessary MLX200 components into an existing project. This method should be used if MLX200 is going to be integrated into an existing application or project.

The following sections describe each method in detail.

2.2.1   Pre-Configured RSLogix Projects

The MLX200 package comes with two preconfigured RSLogix programs: one for ControlLogix (MLX200_ControlLogix.ACD) and one for CompactLogix (MLX200_CompactLogix.ACD). Follow these steps to configure these projects:

> **NOTE**
> The CompactLogix and ControlLogix projects are functionally equivalent. Both configurations are provided because changing the Control Module type between CompactLogix and ControlLogix will require re-entering the EtherNet/IP Module Configuration settings (Refer to *Section 2.2.2.3 "Adding an MLX200 Control Module to the I/O Configuration" on page 2-11*).

2.2.1.1   Configuring the Logix Controller

Open the proper .ACD file in RS Logix depending on which Control Module type you are configuring the program to use. Right-click on the Control Module in the Control Module Organizer and select Properties. Now, click on Change Control Module and input the proper ControlLogix or CompactLogix controller for the system (*Fig.2-3 "Control Module Selection"*).

Fig. 2-3: Control Module Selection

MLX200 Software and Operations	2 System Configuration 2.2 Configuring RSLogix Project
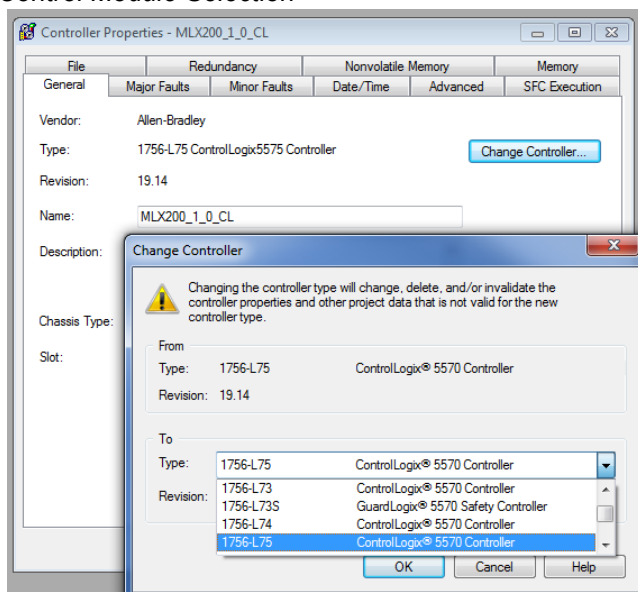
2.2.1.2 Configuring an MLX200 Control Module Communication

If using ControlLogix, open the I/O Configuration in the Control Module Organizer and verify that the correct chassis slot is selected for both the ControlLogix module and the EtherNet/IP module. Then, right-click on the EtherNet/IP Module, select Properties, and verify that the IP Address for the module matches what is displayed in the alpha numeric display on the front panel of the 1756-ENBT module for ControlLogix.

Fig. 2-4: EtherNet/IP Module Configuration
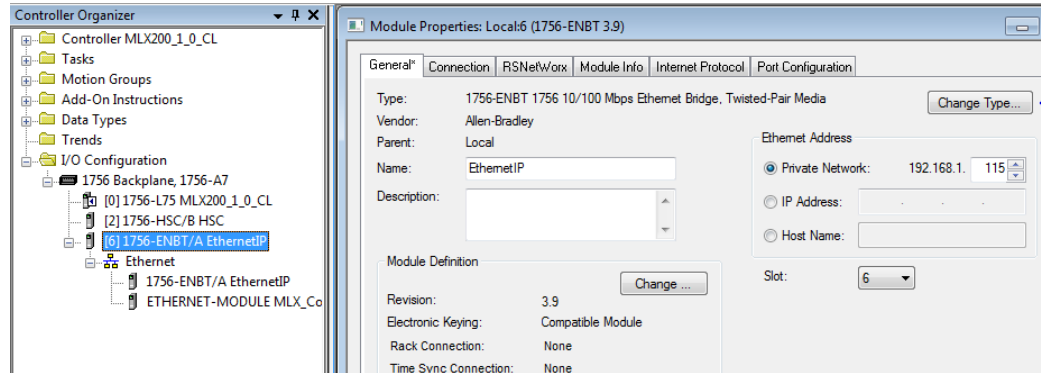
Finally, right-click on the MLX_Controller Ethernet Module, select Properties, and verify that the IP Address matches the IP Address for the MLX Control Module (default should be 192.168.1.200). All other settings on this module should be preconfigured correctly.

Fig. 2-5: EtherNet/IP Module Configuration

168542-1CD
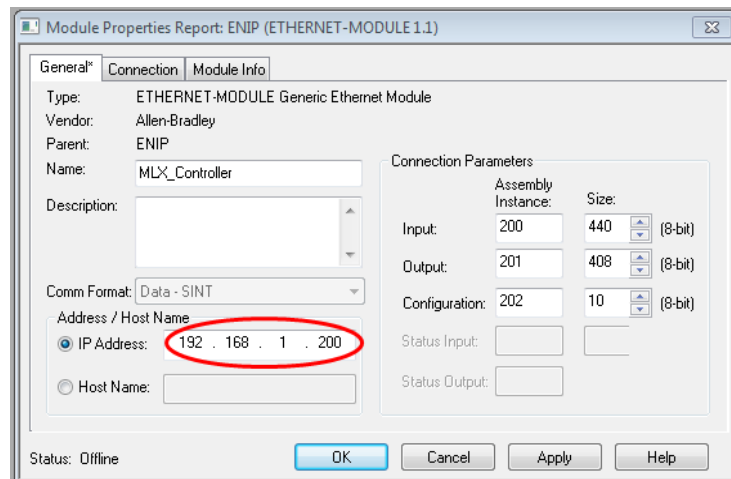
2.2.2   Importing MLX200 into Existing Project

This section will describe how to import a MLX200 into a new or existing RS Logix project. If using a pre-configured project as described in *Section 2.2.1 "Pre-Configured RSLogix Projects" on page 2-7*, this section can be skipped.

2.2.2.1   Import MLX200 AOIS AND UDTS

Right-click on Unscheduled Programs/Phases, select Import Program, and navigate to the MLX200_Import.L5X file included with MLX200 package. This will import all MLX200 AOIs and UDTs into the project.

Fig. 2-6: Importing LXADKImport.L5X
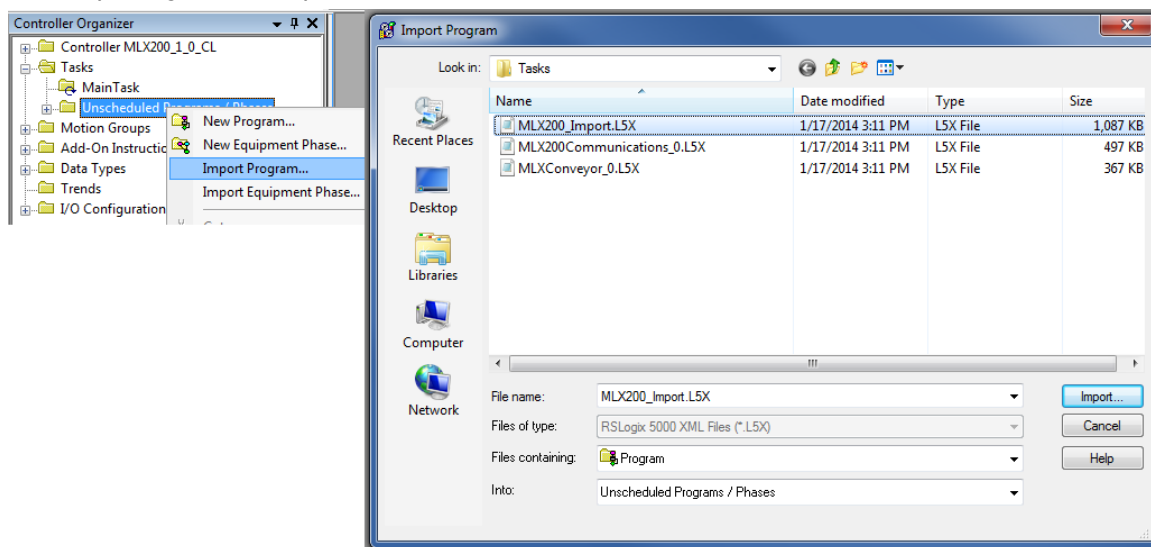
NOTE   The MLX200_Import program performs no function besides importing the AOIs and UDTs. It can be deleted from the RSLogix project after importing.

2.2.2.2   Creating the MLX200 Communications Task

Create a MLX200 communications task. Right-click on Tasks and select "New Task". Set the Type as Periodic, the Period to 2 ms, and the Priority to 1.

NOTE
- The Period/Priority settings are recommendations only. These can be changed if the settings lead to interference with other components in the system or too much CPU utilization; however, though this will not cause any motion control performance issues, lowering these values can cause synchronization delays between the Logix system and the MLX200 Robot Control Module. Appendix B. MLX200 Performance Results and Memory Usage contains some basic performance results for differing values of these parameters.

- The Task Period should be set equal to or higher than the MLX200 Robot Control Module RPI. Setting the Task Period lower than the RPI will not improve performance but will increase CPU load.

- When importing the communications task, you may receive a warning that says "Calls to the following instruction(s) exist in source that is not editable." This warning can be ignored.

Fig. 2-7: MLX200 Task Configuration
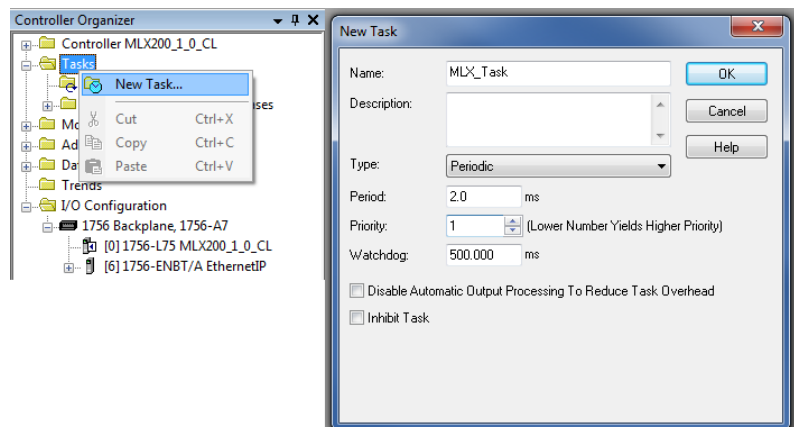
⚠ WARNING

All MLX200 application code should be placed inside the MLX_Task. Failure to do so could lead to unexpected behavior such as skipped motions or motions being out or order after a Hold/Restart scenario. See *Section 3.3.1 "Task Scheduling" on page 3-9*.

Now, right-click on the MLX200_Task, select Import Program, and navigate to the MLX200Communications_0.L5X file included with MLX200.

2.2.2.3   Adding an MLX200 Control Module to the I/O Configuration

Next, the EtherNet/IP communications must be configured. If using ControlLogix, first verify that the 1756 ENBT card is added to the I/O Configuration at the correct slot. Right-click on the Ethernet module and select "New Module". *Fig.2-8 "Adding a New Ethernet Module"* shows this for ControlLogix (left) and CompactLogix (right). Next, select Generic Ethernet Module from the list of devices.

Fig. 2-8: Adding a New Ethernet Module
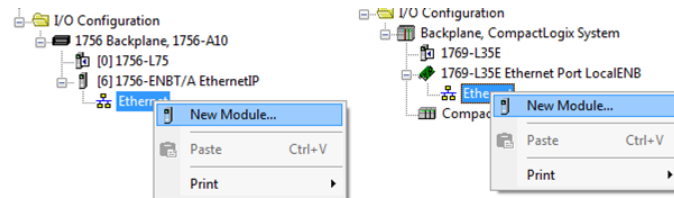
On the General tab, fill in the data as shown in *Fig.2-9 "General Ethernet Module Settings"*. The IP Address entered on this tab should be the IP Address of the MLX200 Control Module (default value is 192.168.1.200). On the Connection tab, set the RPI to 2 ms and make sure the "Use Unicast Connection" checkbox is *not* checked.

Fig. 2-9: General Ethernet Module Settings
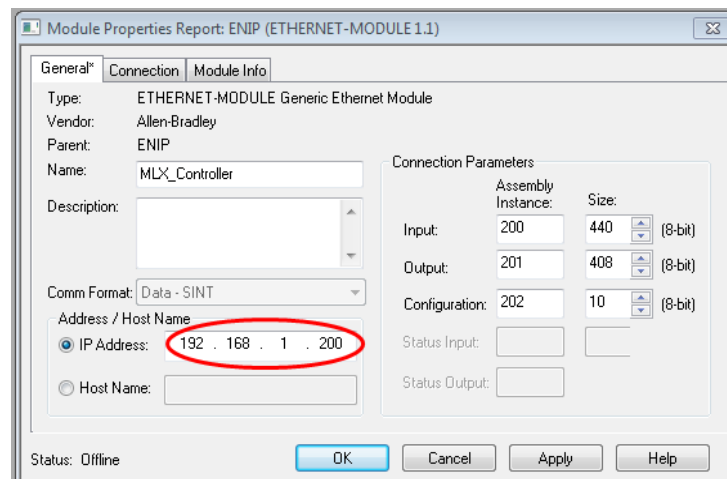
Fig. 2-10: Ethernet Module Connection Settings
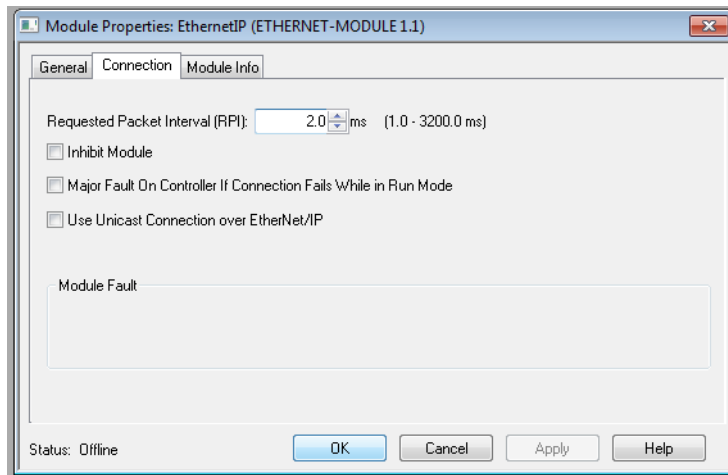
> **NOTE**
>
> If using the MLX-HMI, refer to *Section 3.4 "MLX-HMI" on page 3-19* for instructions on HMI configuration.

3   Developing with MLX200

This section explains how to use MLX200 to develop robotics applications. The first section provides an overview of the tag structure in MLX200. The next section provides an overview of the instructions available for configuring and controlling a robot in MLX200 as well as their basic behavior. Finally, the screens of the MLX-HMI are introduced.

> ⚠ **WARNING**
>
> UNEXPECTED MOVEMENT. Servo drives connected to the MLX200 Control Module may perform unexpected movements because of incorrect wiring, incorrect settings, incorrect data or other errors.
>
> • Interference (EMC) may cause unpredictable responses in the system.
>
> • Carefully install the wiring in accordance with the requirements provided by the servo drive vendor.
>
> • "Switch off the voltage at the inputs to the servo drives avoid an unexpected start of the motor before switching on and configuring the product.
>
> • Do NOT operate the product with unknown settings or data.
>
> • Perform a comprehensive commissioning test.
>
> • Failure to follow these instructions can result in death or serious injury.

3.1 MLX200 Tag Structures

The MLX200 tag structure exists as a Control Module-scope tag labeled MLx[]. This tag structure contains information on the overall state of the system, properties of the MLX200 Robot Control Module and configuration and feedback data for individual axes and robots. The following sections describe the tag structures in detail.

3.1.1 MLx-Level Tag Structure

The MLx-level tag structure contains configuration and feedback data for the system. It is defined as an array of size 1 by default. The size of the MLX-level data structure array will only increase if multiple MLX200 Control modules are being configured from the same system:

- **Root** - under the main MLx tag, there are variables describing the system state, error code, number of Robots, etc.

- **MLxControl Module Info** - describes properties of the MLX200 Control Module such as IP Address and firmware version.

- **Signals** - contains signals for informing the user when the system is connected and initialized.

- **Robot[]** - contains information on each initialized/used robot.

Fig. 3-1: MLx-Level Tag Structure.

168542-1CD
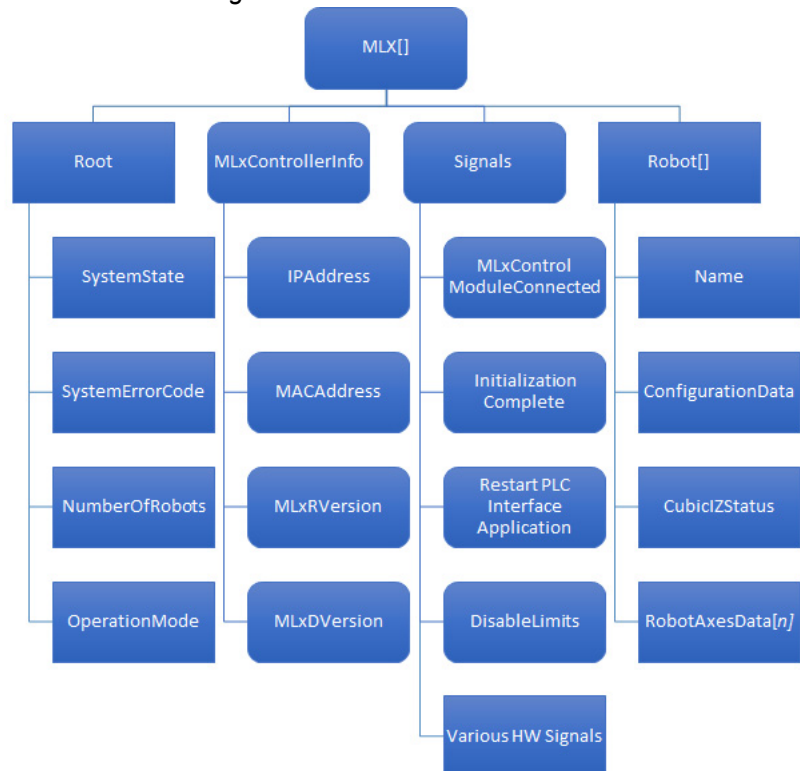
3.1.2 Robot-Level Tag Structure

Each individual robot data structure contains the following information:

- **Name** - name of the Robot defined inside the MLX200 Configuration File.

- **ConfigurationData** - contains properties of the robot kinematics such as TCP speed and acceleration. This data is configured is read from the MLX200 Configuration File and populated automatically during system initialization.

- **CubicIZStatus** - contains status for any Cubic Interference Zone (IZ) violations. The bit corresponding to the ZoneID will be high for any violated zones. See A.18 and A.19 for more information on Interference Zones.

- **RobotAxes[]** - contains information on each individual robot axis. The data for each axis is the same as described in *Section 3.1.3 "Axis-level Tag Structure" on page 3-4*.

Fig. 3-2: Robot-Level Tag Structure
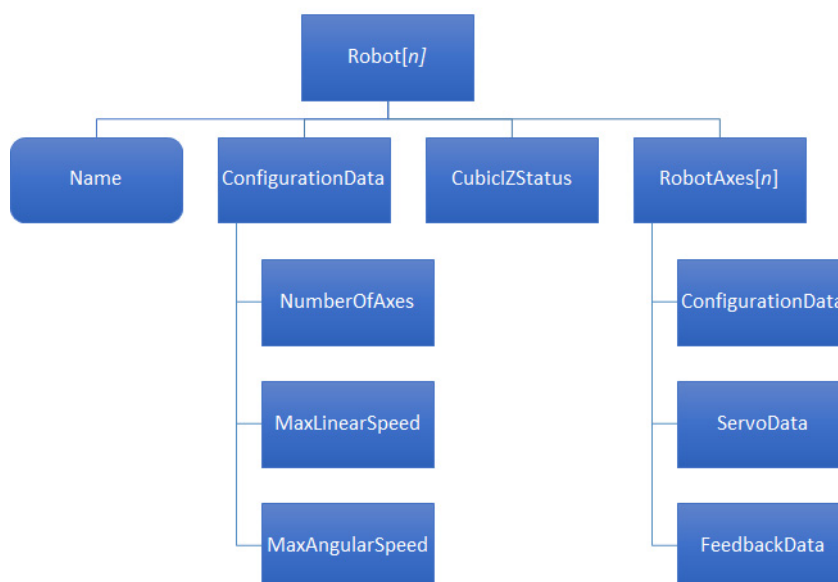
3.1.3 Axis-level Tag Structure

Each individual robot axis data contains the following information:

- **AxisName** - name of the Axis in the format **"[RobotName]-Axis[n]"**.

- **ConfigurationData** - contains properties of the axis kinematics such as position, velocity, and acceleration limits.

- **ServoData** - contains properties of the servo drive such as Pulse (Encoder Counts) Per Position Unit and maximum allowed following error.

- **Feedback** - contains real-time feedback of Commanded position as well as one user-defined feedback data type. By default, the user-defined feedback data will be Actual (HW) Position - use the MLxSetRobotProperties instruction to change this data type.

Fig. 3-3: Axis-level Tag Structure.

168542-1CD
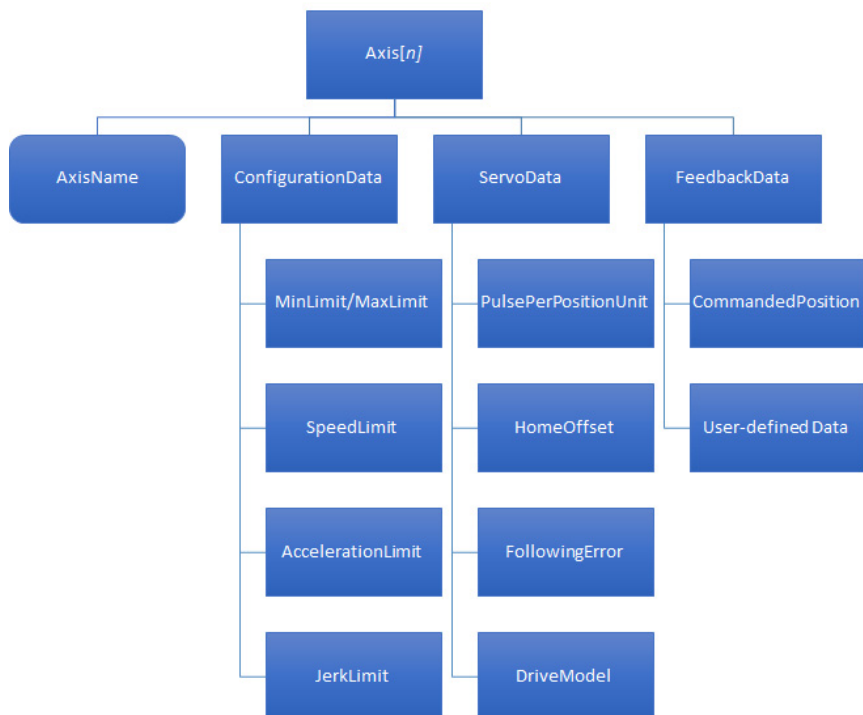
3.1.4   Application Data Tag Structure

The Application Data structure is a Control Module-Scope structure that contains the application data (Teach Points, Tools, IZs, etc…) for a given project. The MLX200 HMI (described in *Section 3-4 "MLX200 Tag Structures" on page 3-5*) will automatically link to this structure when you Teach Points or enter Tool/UF/IZ Data from the HMI. The overall data structure is shown in *Fig.3-4 "Application Data Tag Structure"*.

Fig. 3-4: Application Data Tag Structure
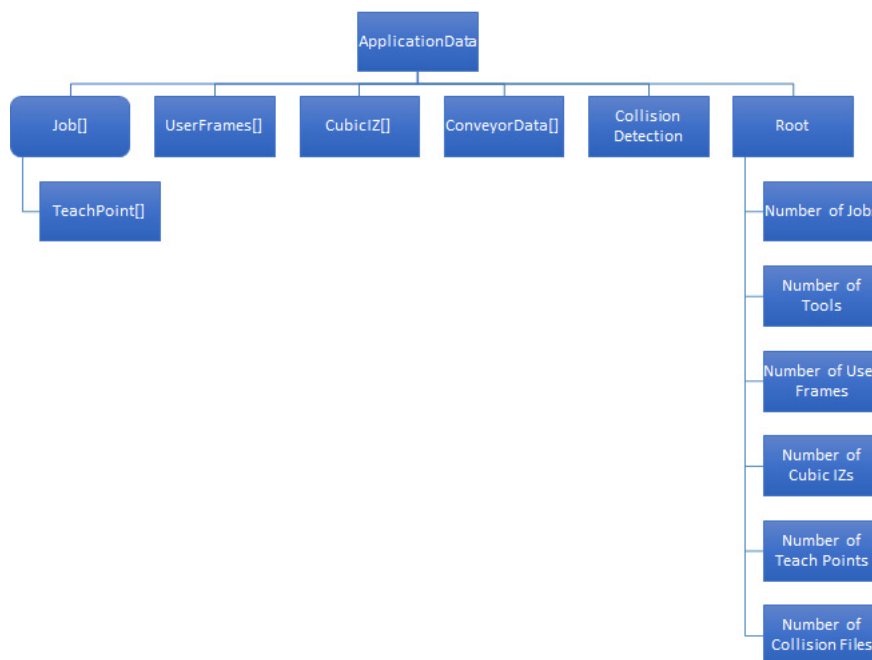
The Application Data structure is designed to be configured so that a user can extend or contract the amount of data stored in it based on the application needs and available Control Module Memory. To change the size of the Application Data requires two steps:

⚠ WARNING
Failure to match the length of the arrays with their corresponding size variable can allow an out-of-index array which will cause the Control Module to fault.

1. Change the variable corresponding to the type of data.

2. Change the array size of the UDT corresponding to the type of data

For example, *Fig. 3-5 "Changing the Size of the ApplicationData.NumberOfJobs Variable" on page 3-6* and *Fig. 3-6 "Changing Size of the MLxAppDataJob UDT" on page 3-6* show changing the number of Jobs from the default value of 10 to 20.

Fig. 3-5: Changing the Size of the ApplicationData.NumberOfJobs Variable

Fig. 3-6: Changing Size of the MLxAppDataJob UDT

> **NOTE**
> If the size of these arrays/variables is changed, the HMI must be shutdown and restarted to have the changes take affect.

3.2   Instruction Overview

This section provides an overview of the MLX200 instructions and a brief description of each one. The first section describes system level instructions for changing system state, retrieving error messages, etc. Then, the Robot-specific Configuration and Motion instructions are described. More detailed descriptions of these instructions can be found in Appendix A.

3.2.1   System Commands

Table 3-1 lists the system commands available in MLX200. The interaction between the state instructions (MLxAbort, MLxReset, etc) and the state of the system is shown in *Fig. 3-7 "State Management with MLX200 Instructions" on page 3-7*.

Table 3-1: MLX200 System Commands

AOI Name	AOI Description
MLxEnable	Enable all configured drives
MLxAbort	Abort all configured drives (controlled stop followed by servo off)
MLxStop	Stop all configured drives (controlled stop with servos staying on) and clear the motion queue
MLxHold	Stop all configured drives (Control Module stop with servos staying on) while maintaining the motion queue
MLxRestart	Restart the system along current path and motion queue from the Held state
MLxReset	Reset faults on all configured drives and clears motion queue
MLxResetAndHold	Reset faults on all configured drives while maintaining motion queue
MLxGetErrorDetail	Retrieve detailed error message from system
MLxGetModuleInfo	Retrieve information on the MLX200 Control Module configuration
MLxReadDigitalInput	Read digital input state on single axis
MLxWriteDigitalOutput	Write digital output state on single axis
MLxSetGlobalParameter	Set a global parameter for MLX. Valid values: 0 - Global Speed Scale.

Fig. 3-7: State Management with MLX200 Instructions

168542-1CD
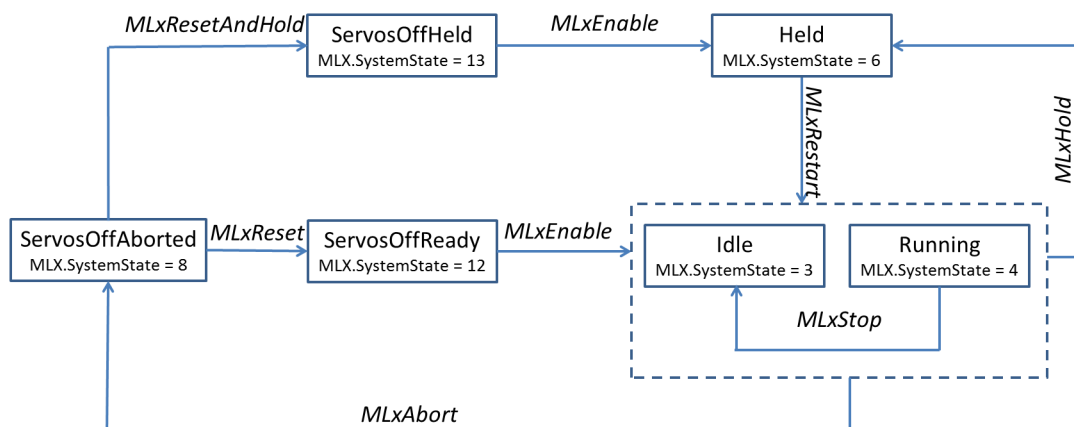
3.2.2 Robot Commands

The MLX200 Robot commands are used to move and configure robots. All of these commands start with the prefix "MLxRobot" and are divided here into Configuration Commands (*Table 3-2 "MLX200 Robot Configuration Instructions"*) and Motion Commands (*Table 3-3 "MLX200 Robot Motion Instructions"*).

Table 3-2: MLX200 Robot Configuration Instructions

AOI Name	AOI Description
MLxRobotCoordinateTransform	Transform Robot Axis Positions to TCP Positions (and vice versa) and convert TCP position between World and User
MLxRobotGetProperties	Read the configured parameters (limits, speeds, accel/decel) for robot
MLxRobotSetProperties	Set configured parameters (limits, speeds, accel/decel) for robots.
MLxRobotSetBasePose	Set the base position of the robot relative to the World Frame
MLxRobotSetToolPose	Set the Tool Offset for the TCP position of the robot
MLxRobotSetUserFrame	Set the active user frame for the robot
MLxRobotSetUserFrameByPoints	Set the active user frame for the robot by supplying Origin, XX, and XY positions
MLxRobotSetFrameShift	Set a constant offset to be used on target position
MLxRobotSetHomeToCurrent	Set the current robot position to the home position
MLxRobotSetCubicIZByCenterPoint	Define a Cubic Interference Zone with a center point and dimensions
MLxRobotSetCubicIZByTwoCorners	Define a Cubic Interference Zone by providing two cube corners
MLxRobotCollisionDetection	Turn on/off and configure Collision Detection

Table 3-3: MLX200 Robot Motion Instructions

AOI Name	AOI Description
MLxRobotMoveAxisAbsolute	Move robot to target absolute position using PTP motion
MLxRobotMoveAxisRelative	Move robot to relative position using PTP motion
MLxRobotMoveLinearAbsolute	Move robot to target absolute TCP position using linear motion
MLxRobotMoveLinearRelative	Move robot to relative TCP position using linear motion
MLxRobotMoveCircular	Move robot through two target positions using a circular motion
MLxRobotJogAxes	Jog the individual axes of the robot
MLxRobotJogTCP	Jog the TCP position of the robot
MLxRobotJogAxesToPoint	Jog the individual axes of the robot to a target position
MLxRobotJogTCPToPoint	Jog the TCP of the robot to the target position.

3.3   Programming Introduction

This section goes into more detail about the method of programming using MLX200. First, the status bits and execution life cycle of the MLX200 instructions are detailed. Then, a simple programming structure that can be used to build applications is described. The basic motion parameters common to the motion instructions are described here as well.

3.3.1   Task Scheduling

⚠️ **WARNING**

Failure to place MLX200 application code inside the MLX_Task could lead to unexpected behavior such as skipped motions or motions being out or order after a Hold/Restart scenario.

Fig. 3-8: Task Scheduling

Any MLX200 Application code should be placed inside the MLX_Task function and not inside the MainTask (*Fig.3-8 "Task Scheduling"*). This will allow for synchronization between the MLX Communications Task and the Application Code. Failure to place the application code inside the MLX_Task could lead to unexpected behavior such as skipped motions or motion being out or order after a Hold/Restart scenario.
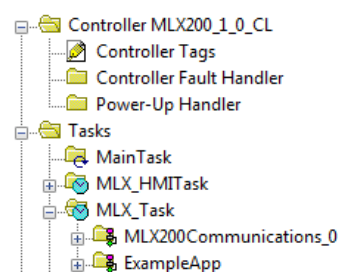
3.3.2   Instruction Execution and Status Bits

Each MLX200 instruction has several status bits that will update while the instruction is executing and provide information that can be used to control the application logic. The state management (e.g. MLxAbort, MLxEnable) instructions and configuration (e.g. MLxAxisReadProperties, MLxRobotSetBasePose) instructions have three status bits:

- **Sts_EN** - Turns on when the instruction rung is enabled
- **Sts_DN** - Turns on when the instruction has finished executing
- **Sts_ER** - Turns on if the instruction causes an error

The motion instructions have additional bits that will report the status of a motion throughout its processing and execution (i.e. movement).

> **NOTE**   For these status bits to report correctly the motion AOI rung should be enabled throughout the entire motion execution.

- **Sts_EN** - Turns on when the instruction rung is enabled
- **Sts_DN** - Turns on when the MLX200 Robot Control Module has acknowledged the motion command and motion has been queued
- **Sts_IP** - Turns on and stays on during motion processing and execution

168542-1CD

• **Sts_AC** - Turns on when motion begins executing and off when motion completes

• **Sts_PC** - Turns on when motion execution is complete

• **Sts_ER** - Turns on if the instruction causes an error

Fig.3-9 "MLX200 Motion Instruction Execution" shows a detailed timeline of the status bit behavior throughout the lifecycle of a motion instruction.

Fig. 3-9: MLX200 Motion Instruction Execution[1)]
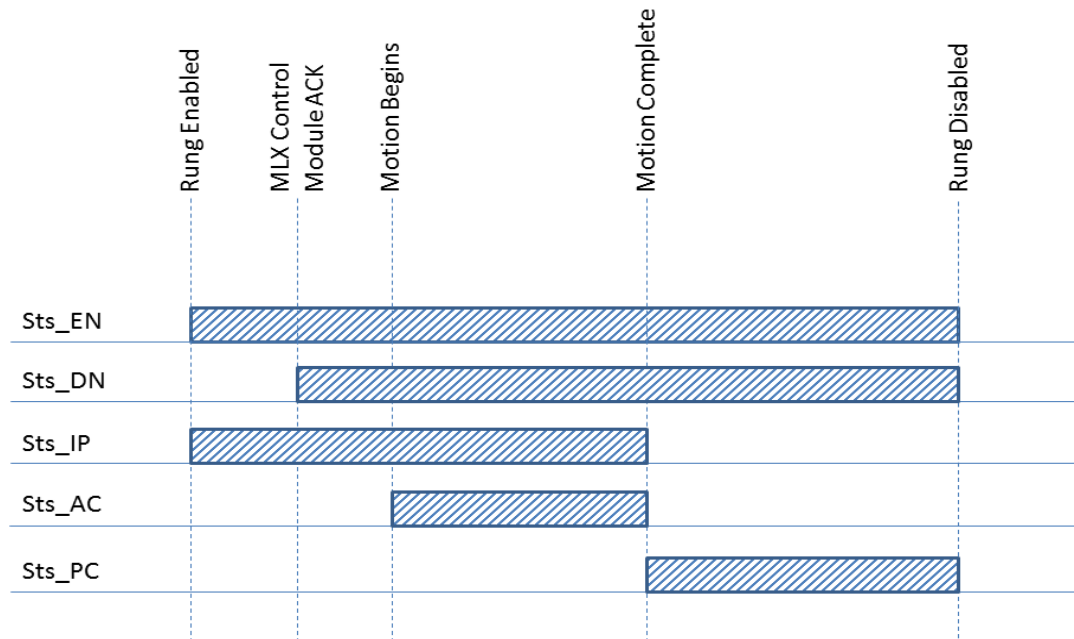
3.3.3 State Management and Configuration Instructions

State management and configuration instructions can be executed on a rung of ladder with the ".Sts_DN" bit used to determine when the instruction has been completed. For example, consider the rung of ladder shown in *Fig.3-10 "Example State Management Instruction"*. A single bit called "triggerEnable" is used to execute the instruction and a control variable of name "enableServos" is passed into the MLxEnable instruction. Then, the enableServos.Sts_DN bit is checked to determine that the execution has completed.

Fig. 3-10: Example State Management Instruction

---
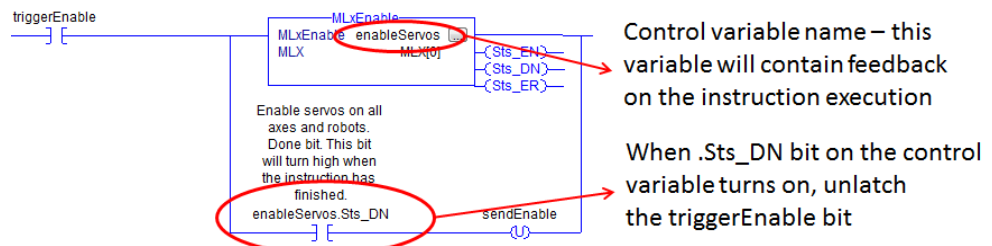
1 Note: *Fig.3-9 "MLX200 Motion Instruction Execution"* describes the behavior of the status bits when the rung is enabled until it is disabled. The Sts_PC will remain on after the rung is disabled until the next time the rung is enabled.

The same structure can also be used to execute configuration instructions such as setting base poses or interference zones.

> **NOTE**
> • Configuration instructions that affect program behavior (such as base pose, tool pose, interference zone, etc) should be executed during application initialization as these values will not hold over when the MLX200 Robot Control Module is restarted.
>
> • Every MLX200 instruction takes an MLxData parameter. By default, this parameter should always be MLX[0]. If using multiple MLX200 Control Modules, this parameter is used to send the command to the proper MLX200 Control Module.

3.3.4 Motion Instructions

The MLX200 motion instructions will make up the bulk of most applications. One simple way to use the motion instructions is to treat every rung of ladder as a "step" and then move to the next step when the motions have finished executing (i.e. the Sts_PC bit turns on). An example of this is shown in *Fig.3-11 "Example Ladder Rung of Simple Motion"*. The rung-in condition here checks whether the variable "step" is 10 and then calls an MLxRobotMoveAbsolute with the control variable "moveHome[0]" when this condition is true. Then, it waits for the moveHome [0].Sts_PC to turn on and moves to the next step of the ladder.

> **NOTE**
> It is often useful to define the control variables as arrays for motion instructions that will be used often in the program. This will cut down on the number of program variables and make the program easier to read.

Fig. 3-11: Example Ladder Rung of Simple Motion
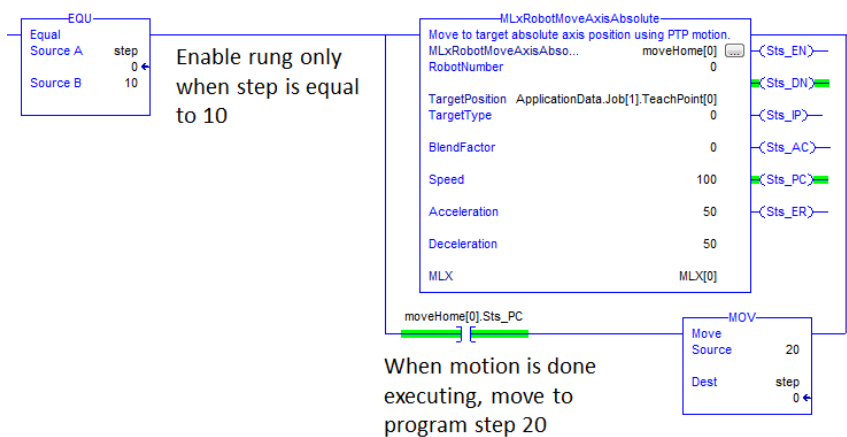
Multiple instructions can be placed on the same rung as shown in *Fig.3-12 "Multiple Motion Instruction On"*. If commanding multiple motions for the same Robot, these motions will be internally queued. Up to 25 motions can be queued (all Axis and Robots combined) in the system at one time. See *Chapter 4 "MLX200 Programming Guide"* for more detailed guide for programming the MLX200.

Fig. 3-12: Multiple Motion Instruction On
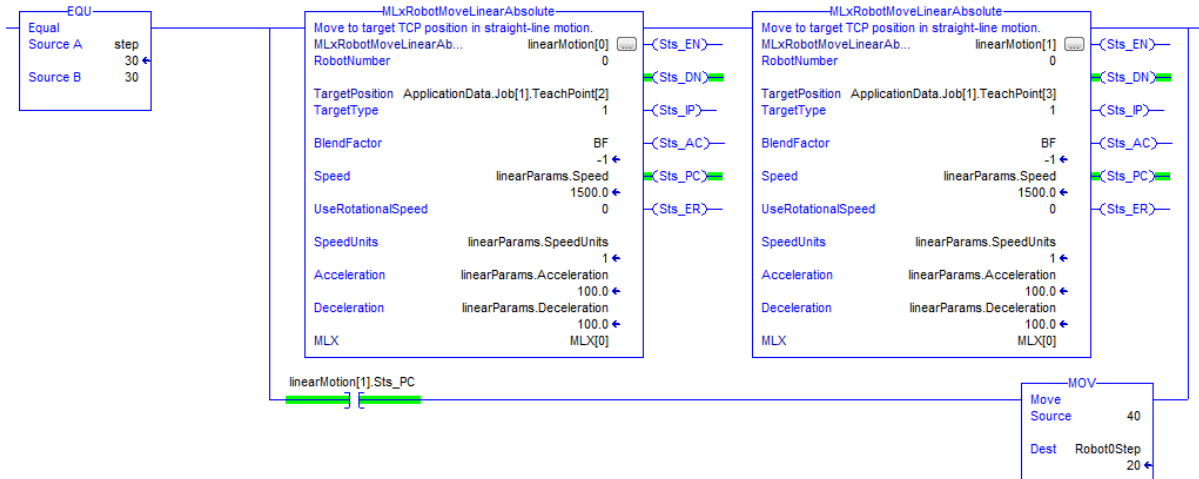
3.3.4.1 Speed, Acceleration, Parameters

Every MLX200 Motion Instruction will take parameters defining the speed and acceleration parameters for the motion. For all axis motions (e.g. MLxRobotMoveAxisAbsolute) parameters are defined in % of maximum. For linear motions, the acceleration parameters are always defined in % of maximum while the Speed can be defined by % of maximum or in absolute values (mm/sec) depending on the SpeedUnits parameter (0 = % maximum, 1 = absolute). For a full list of parameters for each instruction, refer to *Appendix A ""*.

3.3.4.2 Blend Factor

The Blend Factor parameter is used to define at what point a motion should begin to blend into the next queued motion. The valid values for this are (-1)-8 with 0 defining a motion that stops at the trajectory segment end point and 1-8 using distances defined inside the MLX200 Robot Configuration files. By specifying a Blend Factor of (-1), the system will automatically determine a blend distance to match the acceleration/deceleration portions of the motion.*Fig. 3-13 "Example of Blend Factor Effect on Motion" on page 3-13* shows a graphical representation of how increasing Blend Factor affects the shape of the trajectory.

> **NOTE** Blend Factor only applies to Robot Motions and the blend distances are defined as the Cartesian Distance from TCP to trajectory end point.

Fig. 3-13: Example of Blend Factor Effect on Motion
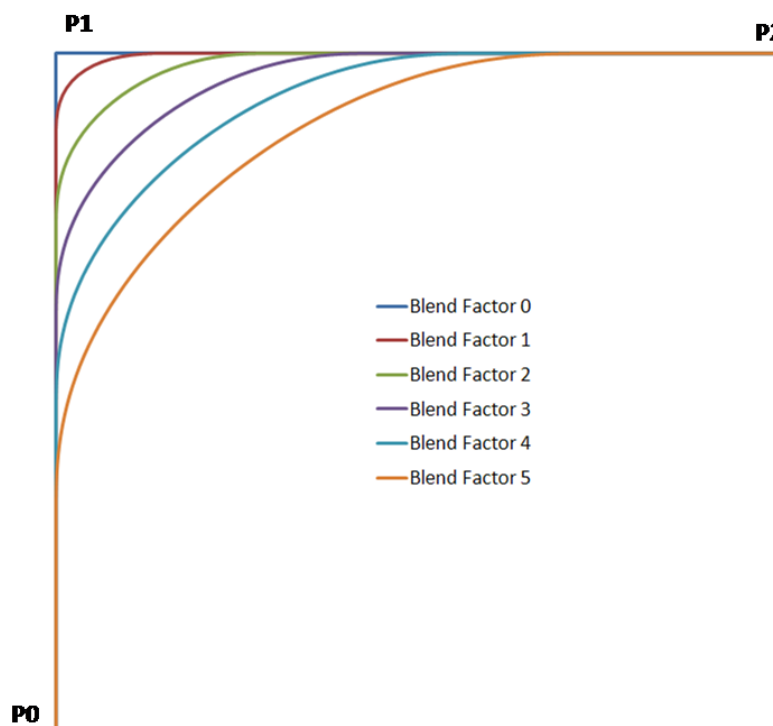
3.3.5   Coordinate Frames Relevant to Robotics

There are several coordinate frames (i.e. "frames of reference") that are important to understand when programming a robot in a workcell.

- **World Frame** - is the base coordinate frame (defined in Cartesian space (X,Y,Z)) of the overall workcell. The position of any object in a workcell (robots, conveyors, pallets, etc) is defined in this frame.

- **Robot Frame** - is the location of the base of the robot defined in World Frame coordinates. Positions defined in the Robot Frame will be relative to the base of the robot instead of the World Frame. This can be seen in *Fig.3-15 "Robot Frame vs. World Frame"* where the TCP reported in the Robot Frame will be offset from the TCP reported in the World Frame. By default, the Robot Frame and World Frame are setup to coincide but can be changed with the MLxRobotSetBasePose instruction.

Fig. 3-14: Robot Cell with Coordinate Frames
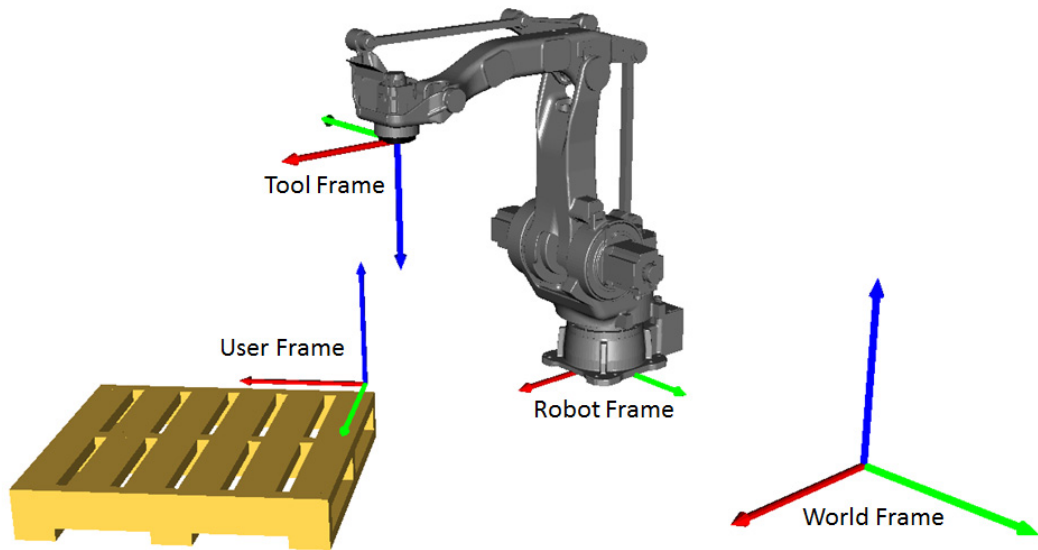
Fig. 3-15: Robot Frame vs. World Frame
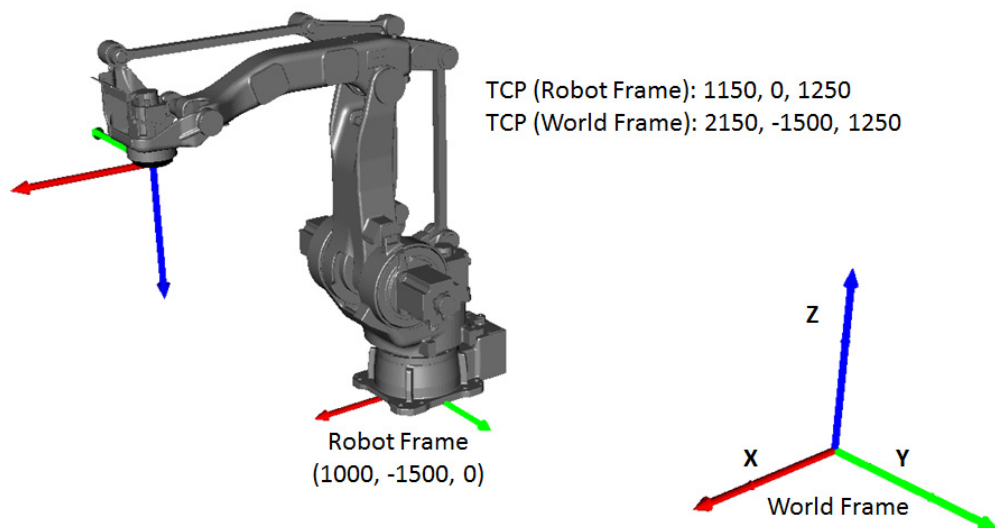
- **Tool Frame** - is a frame attached to a given tool (defaulting to the tool plate). This is useful for defining motions (jogging, incremental motions) relative to the tool axes. For example, a user may wish to move a drill relative to the drilling axis regardless of its location/orientation in the world frame. Tool frame is defined relative to the tool plate of the robot arm. For example, *Fig.3-16 "Changing TCP after Set Tool Command"* shows setting a Tool Pose of [0,0,250,45,0,0]. The Tool Pose can be changed either through the {Tool} Screen on the MLX-HMI or through the MLxSetToolProperties instruction.

168542-1CD

Fig. 3-16: Changing TCP after Set Tool Command
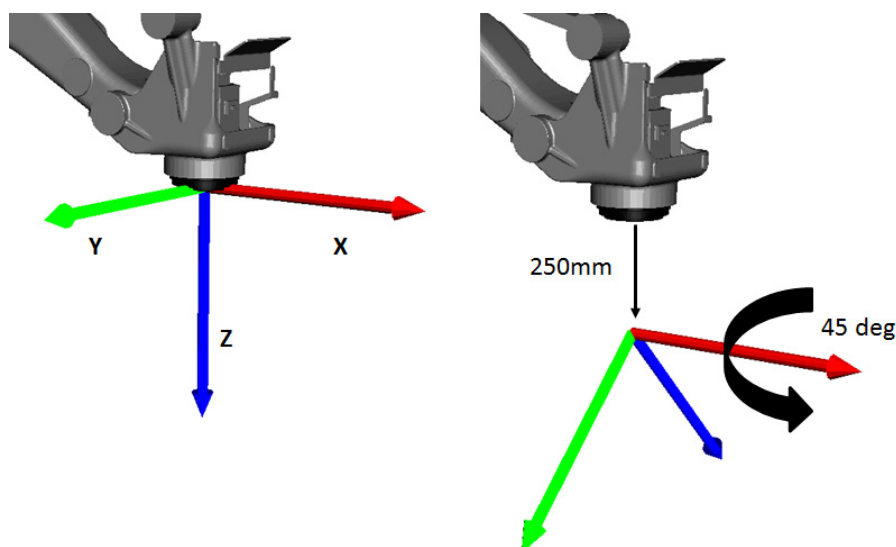
- **User Frame** - is a user defined frame usually attached to some object in the workcell (for example, a pallet in *Fig.3-14 "Robot Cell with Coordinate Frames"*). This allows a user to program motions relative to this location. User Frames are an absolute offset from the Base Pose (Robot Frame) of the robot. For example, if your Base Pose offset is (100,0,0) from the world frame and you define a User frame at (0,100,0) the origin of your user frame is at (100,100,0) from the world frame. A User Frame can be set either through the {User Frame} screen on the MLX-HMI or the MLxRobotSetUserFrame instruction.

3.3.6 Jogging Motions

The MLX200 Jog instructions work slightly differently from the planned motion instructions. First, they require the MLX200 Drive Panel to be set to Manual (Teach) Mode before these instructions will execute. This can be checked by looking at the MLX[].Signals.ManualMode tag in RSLogix.

Second, jogging instructions must be continually called for the motion to continue. If a jog instruction were called and then disabled when the Sts_DN bit turned on, the Robot would only move a small amount. However, if the rung remains active, the directions and speeds can be changed dynamically as from an HMI or other device and continuous motion can be achieved. One caveat in this approach is that other commands cannot be called while the jog instruction has the focus. Thus, to run other commands, the jogging instruction must be temporarily disabled.

| NOTE | • The {Teach} Screen on the MLX-HMI provides all forms of jogging and it is recommended that a user use this HMI rather than calling the Jog Commands directly from application logic. For more information on the, see *Section 3.4.6 "Teach Screen"*. |
| | • The MLxAbort command does not have this limitation and will work even when a jogging instruction has focus. |

168542-1CD

3.3.7   Error Messages

In the case of an error, the system will abort and then report an error code to the MLx[].SystemErrorCode variable. This error code will map to the errors listed in Appendix C. If using the MLX-HMI, there should also be a detailed error message displayed on the {HMI} screen. If not, the MLxGetErrorDetail instruction can be used to populate a variable of type MLxErrorDetail. This object will contain detailed error messages as well as information on how to recover from and avoid the error. An example for a position limit error is shown in *Fig. 3-17*

Fig. 3-17: Example Detailed Error Message

errorDetail	{...}
errorDetail.errorNumber	301
errorDetail.OEMerrorNumber	301
errorDetail.Origin	0
errorDetail.Type	0
errorDetail.Recovery	1
errorDetail.Message	'Axis Position Travel Limit Reached'
errorDetail.ExtendedDescription1	'Axis 0: Axis Position Value: 360.002 Degrees. Limits: -360 Degrees to 360 '
errorDetail.ExtendedDescription2	'Degrees.'
errorDetail.Remedy	'Change the commanded position to avoid violation of position travel limits.'

Table 3-4: MLxErrorDetail Parameters

errorNumber	The MLX error number
OEMerrorNumber	OEM Error number if error code comes from 3rd party device (Servo Drive, I/O Module, etc)
Origin	0 = MLX200 Control Module, 1 = Servo Drive
Type	0 = Alarm/Fault, 1 = Warning
Recovery	0 = No Action Required, 1 = Software Reset Required, 2 = Hardware Reset (System Restart) Required
Message	Basic Error Message
ExtendedDescription1	Extended error description
ExtendedDescription2	Extended error description (continued)
Remedy	Description of how to troubleshoot and prevent error.

3.3.8   Stopping and Recovering Robot Motion

There are multiple ways to stop a robot's motion either from the MLX-HMI, the application logic using the MLX state commands (e.g. MLxAbort) or the MLX200 Drive Panel. The motion can be "aborted" by pressing the [ABORT] button the HMI, calling MLxAbort, by pressing in the [EMERGENCY STOP] button on the control panel, or by opening the Guard Circuit on the control panel. The robot motion can be "held" by pressing the [HOLD] button on the HMI or calling the MLxHold instruction. Finally, the robot motion can be "stopped" by pressing the [STOP] button on the HMI or using the MLxStop instruction. The following sections describe the behavior of each of these and the different recovery methods for each.

> ⚠️ **CAUTION**
>
> The ABORT functionality provided in the MLX-HMI and the MLxAbort instruction should not be used for emergency stopping of the robot. The [EMERGENCY STOP] button that is hardwired in to the control panel should be used for this purpose and for any safety related interlocks.

3.3.8.1 Aborted Motions

A motion is aborted by pressing the [ABORT] button, calling the MLxAbort instruction, pressing the [EMERGENCY STOP] on the control panel, or opening the Guard Circuit on the control panel. When aborting, the robot will stop immediately and servos will be disengaged. The stop is a controlled category 1 stop (i.e. the robot is decelerated to a stop along its path before servo power is removed).

After stopping, the system will be in the ServosOffAborted state. There are two methods to recover from this state: pressing Reset (MLxReset) or pressing ResetAndHold (MLxResetAndHold). If pressing Reset, any motions left in the queue will be flushed. In this case, the program step should be reset, so that the program can restart from the beginning. If pressing ResetAndHold, the motions in the queue are held so that the program can start from where it left off. In this case, the program step should not be reset so that the same instructions are active after pressing START again.

> **NOTE**
> It is up to the application developer to handle the program step correctly when restarting the application depending on the use of Reset or ResetAndHold.

3.3.8.2 Stopped Motions

By pressing the [Stop] button on the HMI or by calling the MLxStop instruction, the robot will come to a Control Module stop but stay in the Idle state (i.e. servos will still be enabled). This can be used to stop the robot's current action and command it to do some other task. It is again up to the application developer to make sure the program step is correct during this operation.

> **NOTE**
> In this case that a motion is stopped while the robot is currently blending between two motions, the robot will stop along its current tangent direction. After restarting, it will move to its next commanded point but will not follow the exact path it would have if it had not been stopped.

3.3.9 Using Global Speed Scale

The MLxSetGlobalParameter instruction can be used to set a global speed scale for the system by passing ParameterType=0 and ParameterValue= (% maximum, 5-100). After this instruction is executed, all subsequently queued motions will be slowed by the defined percentage (e.g. if ParameterValue = 50, all motions will be performed at half speed). This is useful to lower speeds for debugging an application or to reduce speeds at a running application based on some input such as a light curtain. Note that this instruction will only affect motions that are queued after the instruction has completed. We recommend calling MLxHold->MLxSetGlobalParameter->MLxRestart to slow down current motions. This will stop the current motions and then restart them at lower speeds. You can also use the Robot Info HMI screen to change the Speed Scale. When pressing the "Update Speed Scale" button, an MLxHold->MLxSetGlobalParameter->MLxRestart sequence executes automatically.

Fig. 3-18: MLxSetGlobal Parameter Instruction
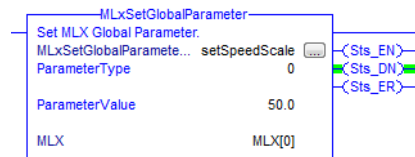
3.4   **MLX-HMI**

> ⚠ **CAUTION**
>
> When integrating the {MLX-HMI} screens into another HMI, DO NOT change any of the tags, {HMI} screens, or object layouts that are used in the MLX200 Robot {HMI} screens. Use of these screens is on an AS-IS basis.

The MLX-HMI includes {HMI} screens that provide a graphical mean for interacting with Robots and the data tables associated with MLX200. The {HMI} screens provide a resolution of 640x480 pixels. The screens are provided as an FTView Studio archive file (*.apa). This file contains the complete MLX HMI development environment and available to integrate the MLX-HMI into a larger HMI.

The following sections will describe how to set up the HMI, and the layout of the various screens.

3.4.1   Setting Up the HMI

The MLX-HMI consists of three parts: an RSLogix task which handles communications with the HMI, an Application Data structure that is used to store data, and the FactoryTalk HMI project.

> **NOTE**
>
> If using one of the pre-configuring RSLogix project files (as described in *Section 2.2.1 "Pre-Configured RSLogix Projects" on page 2-7*), the HMI Task and Application Data structure will already be present in the project. In this case, skip to *Section 3.4.1.3 "Running the FTVIEW HMI Application"*.

3.4.1.1   Importing the MLx-HMI Task

The first step to set up the HMI is to create the MLx_HMITask inside the
RSLogix Project. To do this, right-click on Tasks in the Control Module
Organizer and select "New Task". Then, enter the task properties as
shown in *Fig.3-19 "MLx_HMITask Properties"*.

Fig. 3-19: MLx_HMITask Properties
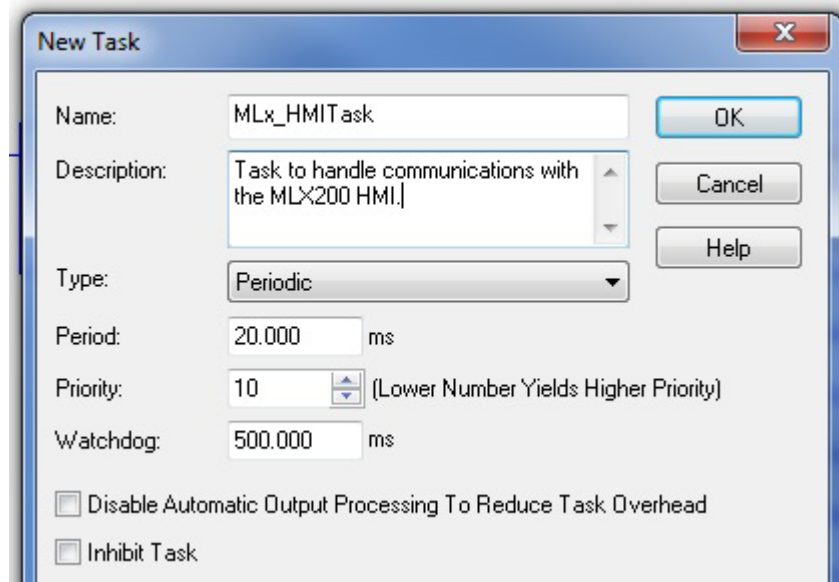
> The Period of 20 ms and Priority of 10 are recommended
> default settings. The Period and Priority could be made
> lower if the responsiveness of the HMI (particularly while
> NOTE jogging) seems slow, but this will increase the CPU Load.
> Similarly, the Period and Priority could be made higher if
> CPU resources are scarce, but this may lead to some lag or
> poor performance (particularly while jogging).

Next, right-click on the task, choose "Import Program", and select the
provided HMIUpdates.L5X file. After importing the task, it should appear in
the Control Module Organizer as shown in *Fig.3-20 "HMI Task in Control
Module Organizer"*.

Fig. 3-20: HMI Task in Control Module Organizer
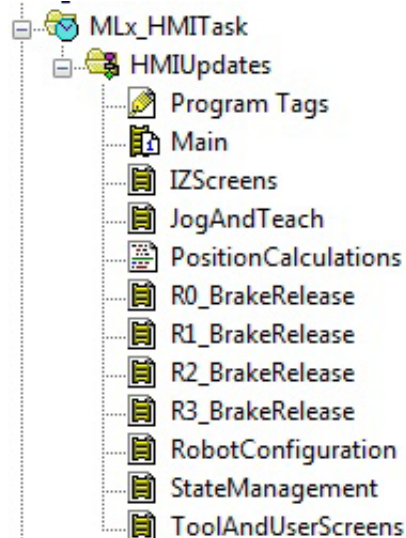
3.4.1.2   Importing MLxApplicationData

⚠ **CAUTION**

Failure to match the length of the arrays with their corresponding size variable can allow an out-of-index array which will cause the Control Module to fault. See *Section 3.1.4 "Application Data Tag Structure"* for more information on configuring the size of the Application Data.

The next step is to add a Control Module scope tag of type MLxApplicationData to the project. Right-click on Control Module Tags in the Control Module Organizer, and select "New Tag". Enter "MLxApplicationData" as the Data Type and "ApplicationData" as the Name. This variable should now appear in the Control Module Tags as seen in *Fig.3-21 "ApplicationData Tag Structure"*. The integer values (NumberOfJobs, NumberofTools, etc) are used by the HMI to determine the range of allowed indices of this value. These integer values must correspond to the actual array lengths (e.g. NumberOfJobs is 10 here and the length of the AxAppDataJob[] array is also 10). These array lengths can be modified if more or less data is required; however, the integer values must also be changed to match. Failure to do this could allow an out-of-index array access from the HMI which will cause the Control Module to fault.

> **NOTE**
> If this size of these arrays/variables is changed, the HMI must be shutdown and restarted to have the changes take affect. See *Section 3.1.4 "Application Data Tag Structure"* for more information on configuring the size of the ApplicationData.

Fig. 3-21: ApplicationData Tag Structure

ApplicationData	{...}	MLxApplicationData
ApplicationData.Job	{...}	MLxAppDataJob[10]
ApplicationData.Tools	{...}	MLxAppDataTool[24]
ApplicationData.UserFrames	{...}	MLxAppDataUserFrame[24]
ApplicationData.CubiclZ	{...}	MLxAppDataCubiclZ[32]
ApplicationData.ConveyorData	{...}	MLxConveyorData[4]
ApplicationData.CollisionDetect	{...}	MLxAppDataCollisionDetect[10]
ApplicationData.NumberOfJobs	20	DINT
ApplicationData.NumberOfTools	24	DINT
ApplicationData.NumberOfUserFrames	24	DINT
ApplicationData.NumberOfCubiclZs	32	DINT
ApplicationData.NumberOfTeachPoints	20	DINT
ApplicationData.NumberOfCollisionFiles	10	DINT

3.4.1.3   Running the FTVIEW HMI Application

After the Task and the ApplicationData have been added, the HMI project file can be opened in FactoryTalk View Studio and executed. The following sections will discuss the various screens on the HMI.

3.4.2 Main Screen

The {Main} screen (*Fig.3-22 "MLX200 Robot HMI Main Screen"*) is the
{Start-up} screen for the HMI. This screen contains the following
components:

- Indicators for when the MLX200 Control Module is initialized and
  connected

- Buttons to control the state of the system (Enable, Reset, Hold,
  Abort, etc…) along with an indicator for the current state. Only the
  buttons relevant to the current state will be displayed on the HMI. For
  example, *Fig.3-22 "MLX200 Robot HMI Main Screen"* shows the
  buttons available in the Idle state.
  Fig.3-24 "Full Display of State Management Buttons" shows a list of
  all of the various state buttons.

- A [Menu] button that overlays the Navigation Menu (*Fig.3-25
  "MLX200 Robot HMI Menu Selection"*) that can be used to navigate
  to the other {HMI} screens.

- A Key Switch indicator displaying whether the system is in Automatic
  or Manual Mode

- A [Release Limits] button that allows temporarily bypassing the
  software limits configured for the Robot.

- A [Brake Release] button that can be used to release the brakes on
  the Robot

- Login/Logout buttons for HMI security

Fig. 3-22: MLX200 Robot HMI Main Screen
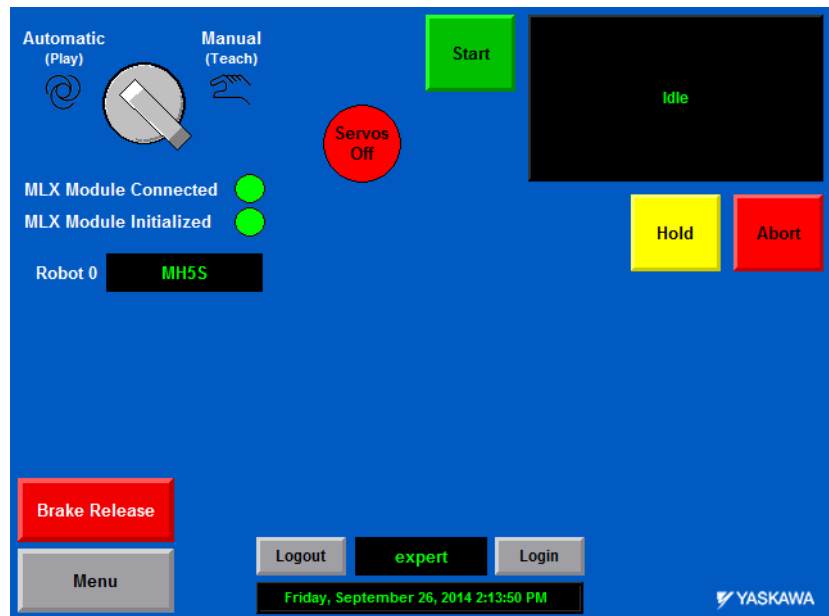
The {Main} screen also displays alarm messages when alarms occur during the normal operation of the system. *Fig.3-23 "Main Screen with Alarm Message display"* shows an example of the {Main} screen with alarm displayed. Clicking on the alarm message will take you to the {Alarm} screen which displays further information about the alarm. See *Section 3.4.5 "Alarm Screen"* for more information about the {Alarm} screen.

Fig. 3-23: Main Screen with Alarm Message display
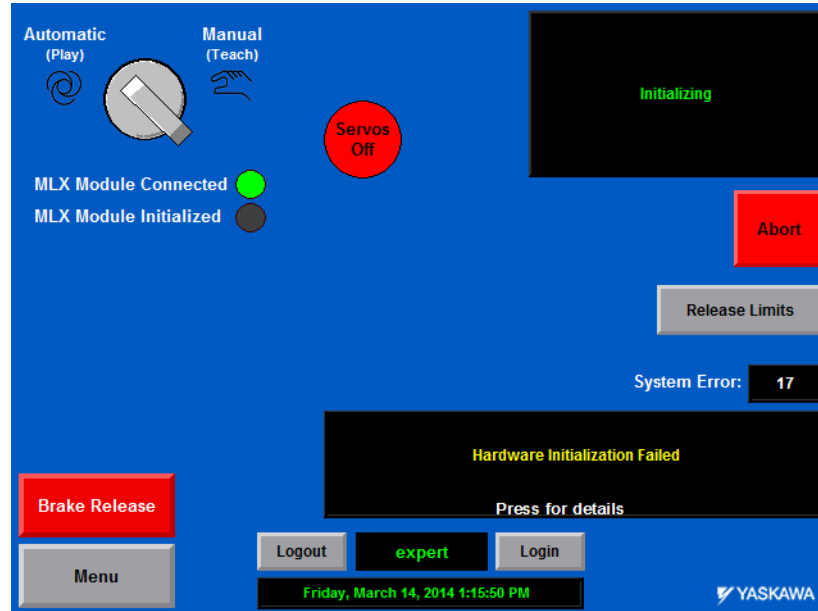
Fig. 3-24: Full Display of State Management Buttons
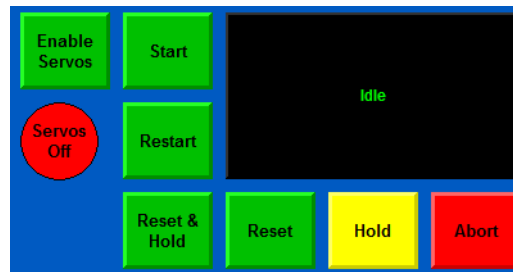
3.4.3 HMI Menu Selection

The [Menu] button on the bottom left of the MLX-HMI is used to access all the screens in the HMI. The visibility of the buttons in the {Menu} screen (*Fig.3-25 "MLX200 Robot HMI Menu Selection"*) is based on the user credentials. When a user does not have access to a particular area of the HMI that button is grayed out. See *Section 3.4.4 "Login and Security Settings"* for more information on login and security settings.

Fig. 3-25: MLX200 Robot HMI Menu Selection
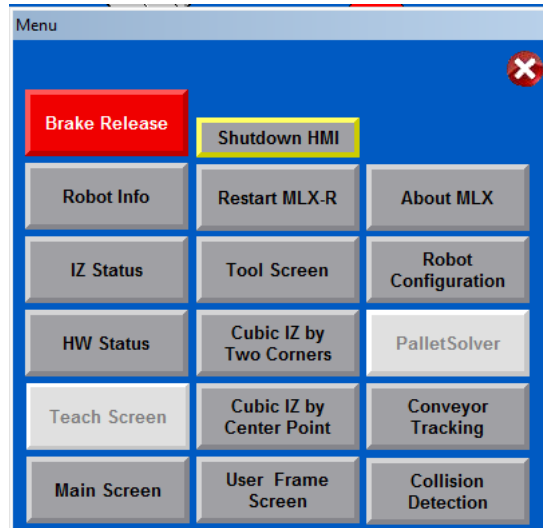
- **Restart MLX-R** - when pressed, a confirmation box will pop-up confirming that the user wants to restart MLX PLC Interface Software (MLX-R). This functionality is not part of "normal" robot operation and should only be used if the continued operation requires a restart of the MLX PLC Interface. Setting the Home Offsets and changing the operating mode are examples of operations that require the MLX200 process to restart. Prior to using this functionality, the user should press [ABORT] first.

Fig. 3-26: Restart MLX-R Confirmation Screen
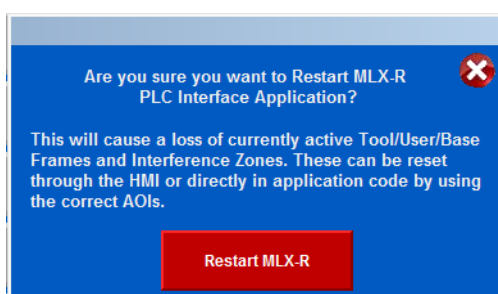
- **Brake Release** - if the logged in user has the correct permission, the brakes of a given axis (robot or individual) can be released. See *Section 3.4.11 "Brake Release Screen"*.

- **Robot Configuration** -the {Robot Configuration} Screen allows the user to setup data for the robot, see *Section 3.4.9 "Robot Configuration"*.

- **Robot Info** - the {Robot Info} Screen allows the see position and limits information for the current robot, see *Section 3.4.10 "Robot Info"*.

- **IZ Status** - this allows you to see the status of the interference zones. See *Section 3.4.12 "Interference Zone Status Screen"* for more information.

- **Cubic IZ Setup** - the two [Cubic IZ] buttons displays the respective screens for setting up Cubic Interference Zones using two methods - Two Corners and Center Point.

- **HW Status** - shows indicators for current HW signals' state (*Fig.3-27 "HW Status Screen"*)

Fig. 3-27: HW Status Screen
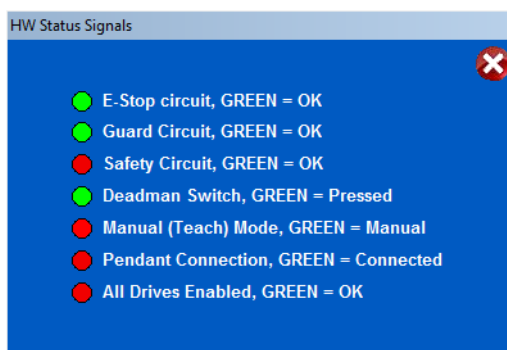
• **User Frame Screen** - There are two methods for setting up user frames, by coordinate or by taught points. See *Section 3.4.7 "Tool and User Frame Screens"* for more information.

• **Tool Screen** - This allows the user to enter the properties of the tools the robot will be using. See *Section 3.4.7 "Tool and User Frame Screens"* for more information.

• **Teaching Screen** - This button links to the Teach Screen, see *Section 3.4.6 "Teach Screen"* for more information

3.4.4   Login and Security Settings

The MLX-HMI comes with pre-defined user names and passwords. Every screen of the HMI has a minimum required security level to access that screen. In addition to per screen access privileges there are certain HMI functions that are hidden to users without proper security clearance. HMI security is based on the letters A (lowest security level) thru P (Highest security level). For example the "operator" login can access the error log, interact with the state manager, etc. but cannot access the {Teach} screen. *Table 3-5* lists all HMI user names and passwords that are predefined. *Table 3-6* lists all the {HMI} screens and the security level required to access that screen.

> **NOTE**
>
> When importing the MLX-HMI into a Factory Talk development environment the user names listed in *Table 3-5 "Changing the Size of the ApplicationData.NumberOfJobs Variable"* should be created in the system/user area () before importing the .apa file. If this is not done, the PLC Interface security will get corrupted and have to be recreated. If the user logins get corrupted or you want to change the user names then follow the security letter access laid out in *Section 3.4.4 "Login and Security Settings"*.

Table 3-5: MLX200 Robot HMI Security-User Logins

User Name	Password	Security Letter(s)
Default	\<no password\>	A
Operator	OPERATOR	A and B
Expert	EXPERT	A thru P

Table 3-6: HMI Security-by Screen

HMI Screen \ FTView Security Level	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
About Yaskawa	✔															
Extended Event Description	✔															
Hardware Status	✔															
IZ Status	✔															
Main Screen	✔															
Menu	✔															
Main Screen Abort Button	✔															
Main Screen Reset, Enable Button		✔														
Main Screen Jogging Mode, Release Limit Button				✔												
Release Brakes		✔														
MLX-R Restart Confirmation		✔														
Robot Configuration				✔												
Setup Cubic by Center Point				✔												
Setup Cubic Interference Zone				✔												
Setup Tool Properties				✔												
Setup User Frame Point				✔												
Teaching and Jogging				✔												

The appearance of {Main} screen and the Menu selection for each logged in user is shown in *Fig.3-28 "Main Screen and Menu Selection for the "Default" Login"*, *Fig.3-29 "Main Screen and Menu Selection for the "Operator" Login"* and *Fig.3-30 "Main Screen and Menu Selection for the "Expert" Login"*.

Fig. 3-28: Main Screen and Menu Selection for the "Default" Login
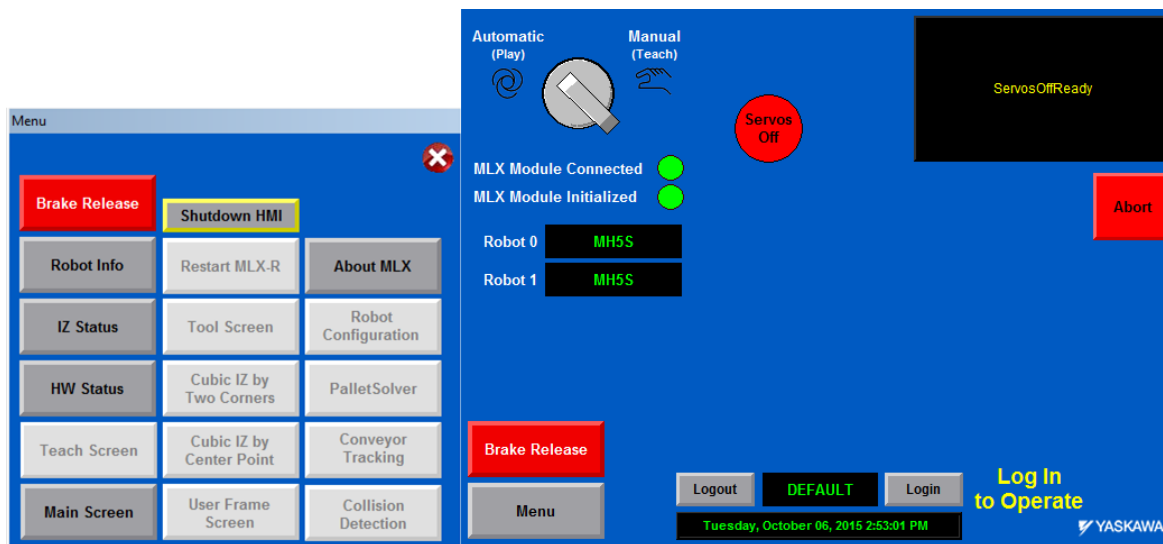
Fig. 3-29: Main Screen and Menu Selection for the "Operator" Login
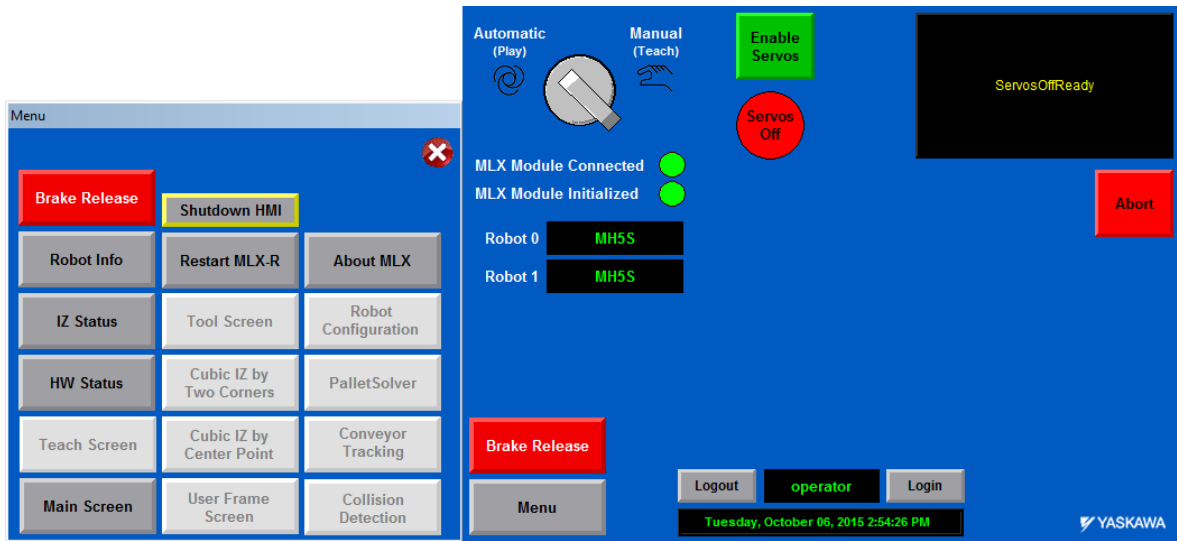
Fig. 3-30: Main Screen and Menu Selection for the "Expert" Login
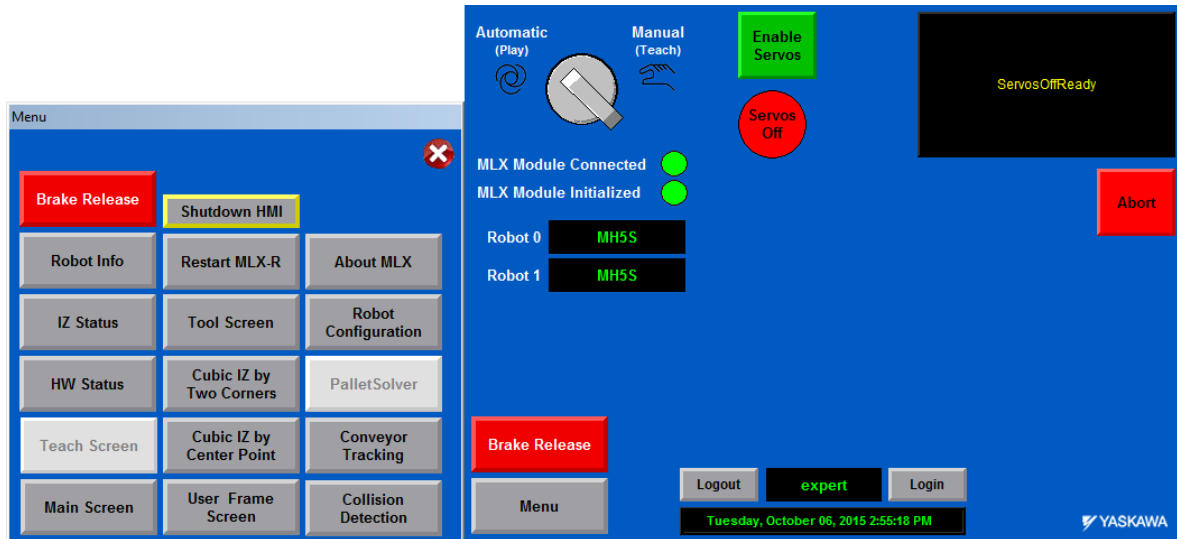
3.4.5 Alarm Screen

The {Alarm} screen display more information about the current alarm (error) in the system when the error is generated.

Fig. 3-31: Alarm Details Screen
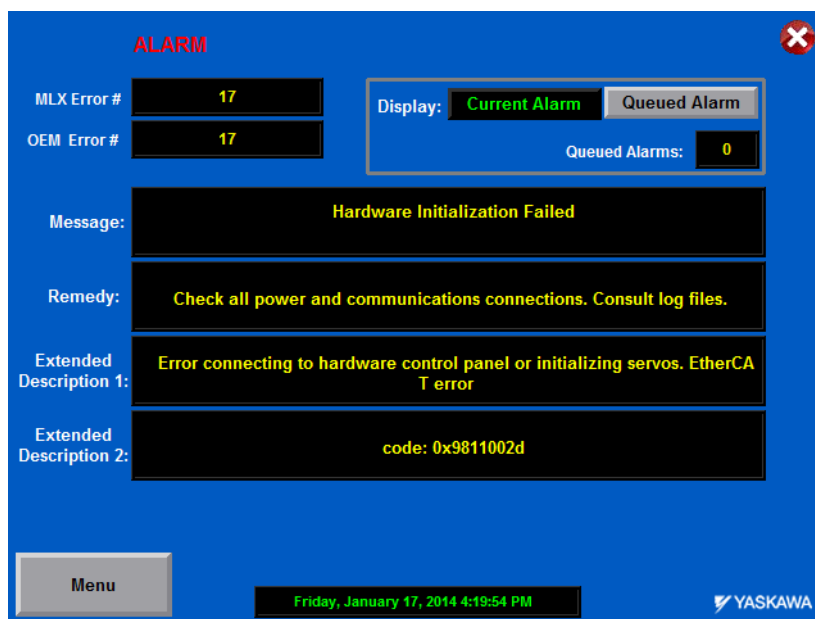
The Alarm details screen also allows the user to retrieve and view queued alarms.

Fig. 3-32: Queued Alarm Screen
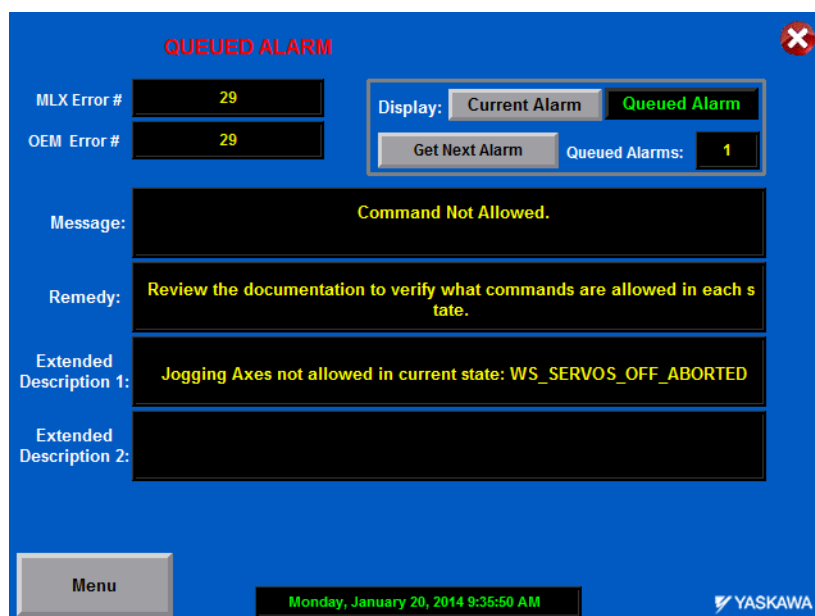
3.4.6 Teach Screen

The {Teach} Screen (*Fig.3-33 "MLX200 Robot HMI Teach Screen"*) allows the user to jog the robot in a variety of methods as well as teach positions. A brief overview of the functionality on this screen:

- The Jogging Mode can be switched between Axis and TCP depending on the desired mode of operation.

- In TCP mode, the Coordinate System can be chosen from World, Tool or User.

- The Jog Speed allows four different default values for: Inch, Slow, Medium, and Fast.

- The Job Number and Teach Point Number can be entered via the numerical inputs, and then when the [Teach Position] button is pressed this data will be copied and stored into the ApplicationData.Job[x].TeachPoint[y] tag where it can later be accessed from an application.

- The [Jog To Point] button can be used to jog to the current selected Teach Point for touching up or replacing a previously taught position. This button will highlight when the Taught Point is reached.

- The [Jog To Home] button can be used to jog the robot to a position of all zeros. This button will highlight when the Home Position is reached.

- [Release Limits] button that can be used to job the robot out of a limit violation.

- The IZ Indicator in the bottom right can be used to track the status of a single Interference Zone that is entered in the numeric input.

Fig. 3-33: MLX200 Robot HMI Teach Screen

168542-1CD
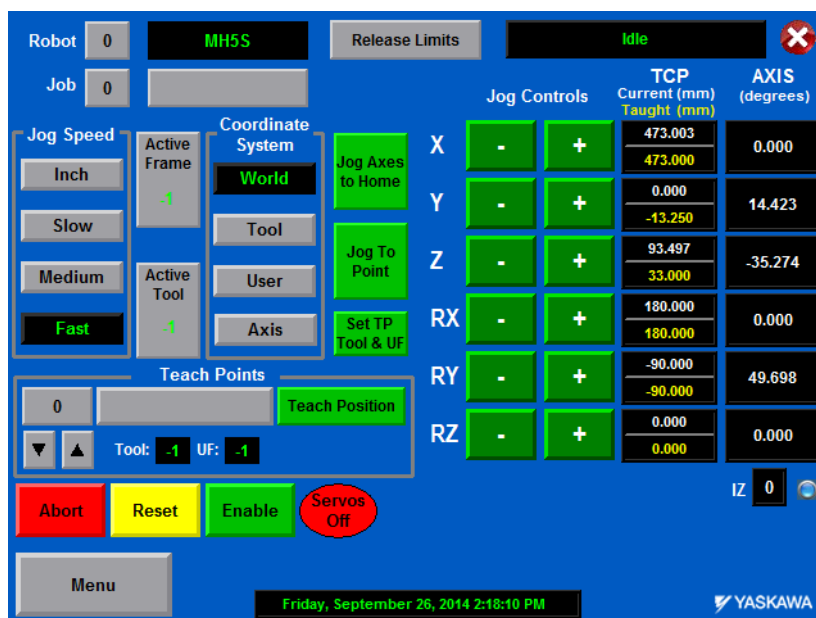
3.4.7 Tool and User Frame Screens

The {Tool and User Frame} screens are shown in *Fig.3-34 "Tool Properties Setup Screen"* and *Fig.3-35 "User Frame Setup Screen"*. The functionality of these screens is similar: both allow the properties of the Tool and User Frame to be inputted using numerical inputs (note: the User Frame also allows the user to copy the current X,Y, and Z position of the TCP for teaching User Frames). When the [Save Tool Data] or [Save User Frame] button is pressed, the data entered on the HMI is copied into the correct Tool or User Frame number inside the ApplicationData Control Tag. Then, when the [Execute Tool Change] or [Set User Frame] button is pressed, the Tool or User Frame is actually set on the robot.

Fig. 3-34: Tool Properties Setup Screen
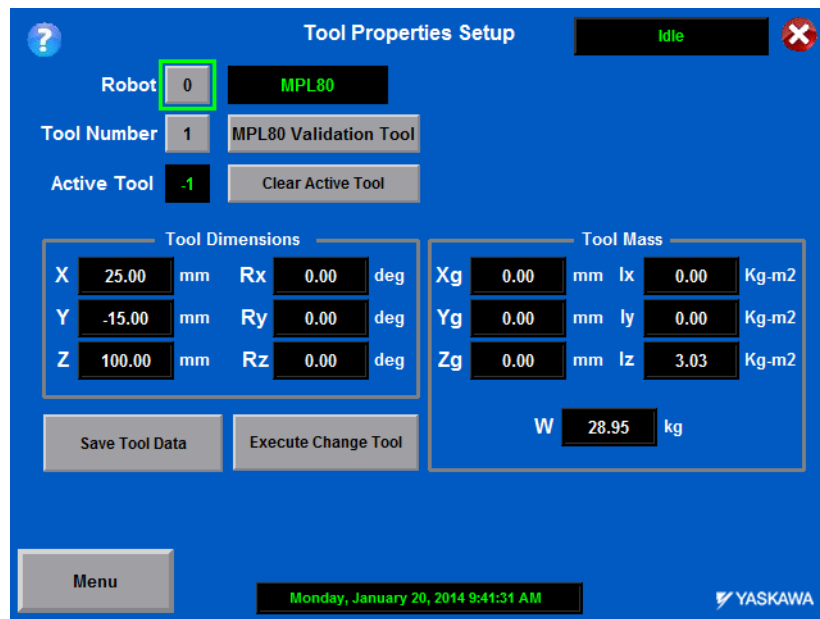
Fig. 3-35: User Frame Setup Screen
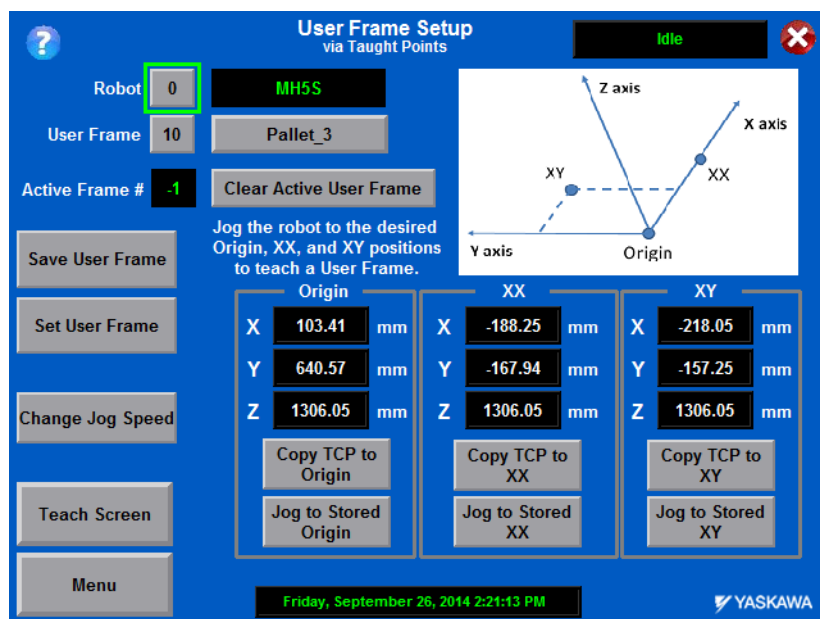
3.4.8   Cubic Interference Zones

The HMI offers two screens for setting Cubic Interference Zones: one for setting a Cubic IZ by defining Two Corners (*Fig.3-36 "Cubic Interference Zone Setup using 2 Corners"*) and one for setting a Cubic IZ using a Center Point and Dimensions (*Fig.3-37 "Cubic Interference Zone Setup using Center Point"*). Similar to the Teach and User screens, these screens allow data to be entered through numerical inputs, saved to the ApplicationData tag, and executed to activate the desired Interference Zone.

Fig. 3-36: Cubic Interference Zone Setup using 2 Corners
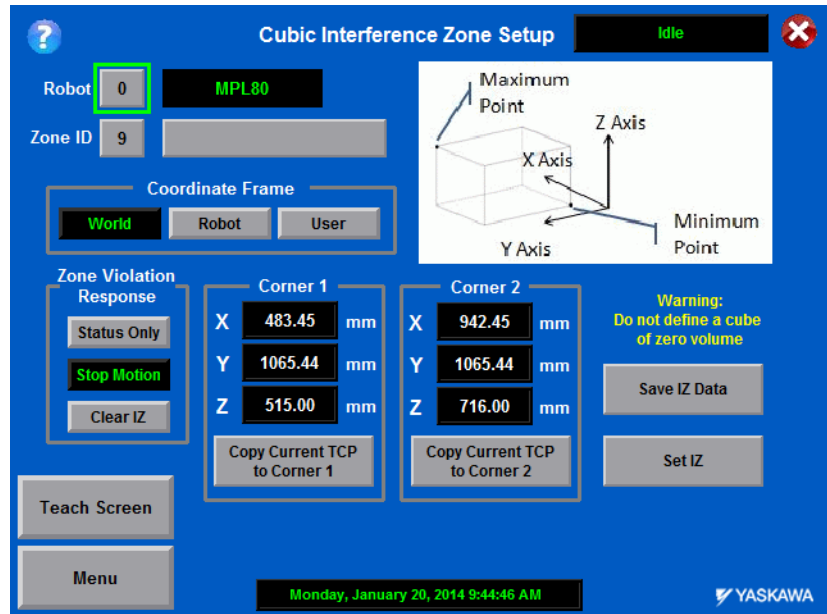
Fig. 3-37: Cubic Interference Zone Setup using Center Point

168542-1CD
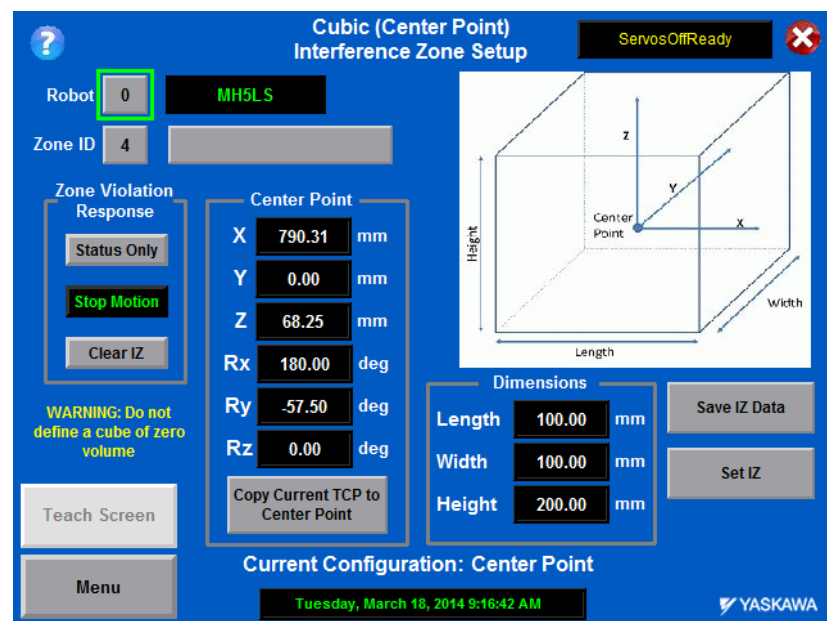
3.4.9 Robot Configuration

The {Robot Configuration} screen (*Fig.3-38 "Robot Configuration Screen"*) is used to update some properties of the robot system. This screen interacts with the RobotConfiguration subroutine in the HMI task. The properties that can be updated are:

- Robot Base Pose

- Robot Soft Position Limits

- Robot Home Offsets

Fig. 3-38: Robot Configuration Screen
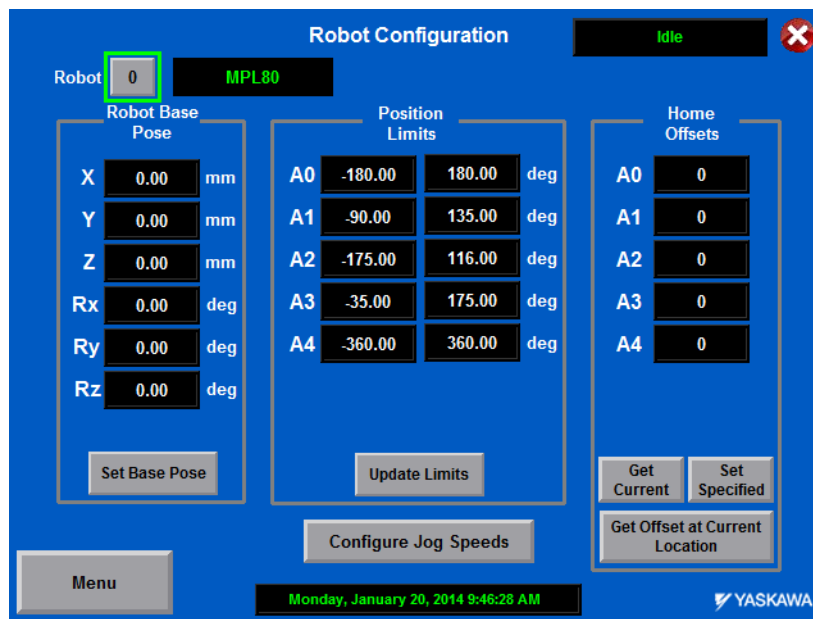
If updating the home offsets, the warning shown in *Fig.3-39 "Robot Configuration - Confirmation screen for setting home offsets."* will appear for additional confirmation.

Fig. 3-39: Robot Configuration - Confirmation screen for setting home offsets.

168542-1CD
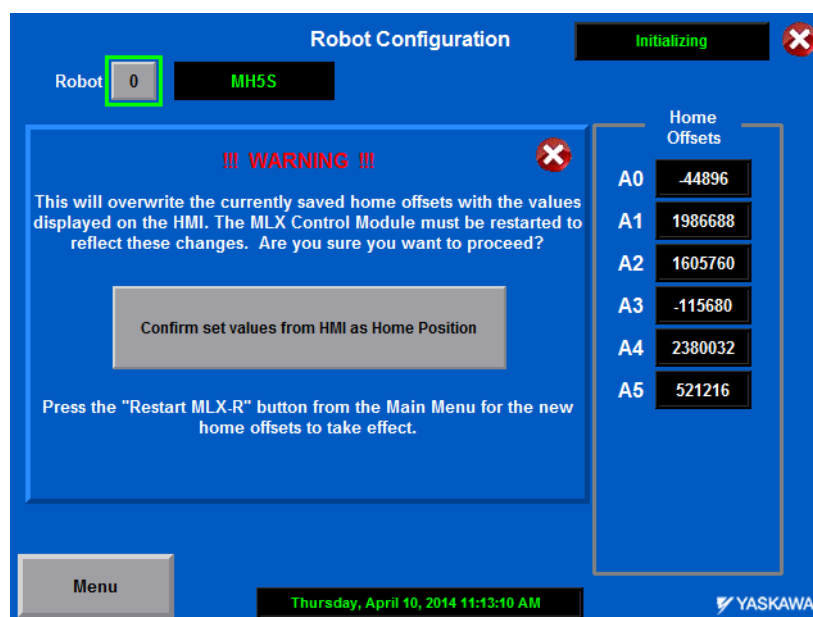
3.4.10 Robot Info

The {Robot Info} Screen is just a placeholder for displaying various properties of the robot. Currently, it displays the Robot Axis and TCP position as well as the Linear and Angular TCP limits data. It also displays the Optional Features that have been enabled on this robot.

There is also a control to change the Global Speed Scale for the system. Enter a % (5-100) into the numerical input and then press the Update Speed Scale button. Pressing this button automatically hold the system, change the speed scale, and then restart the system.

Fig. 3-40: Robot Info Screen
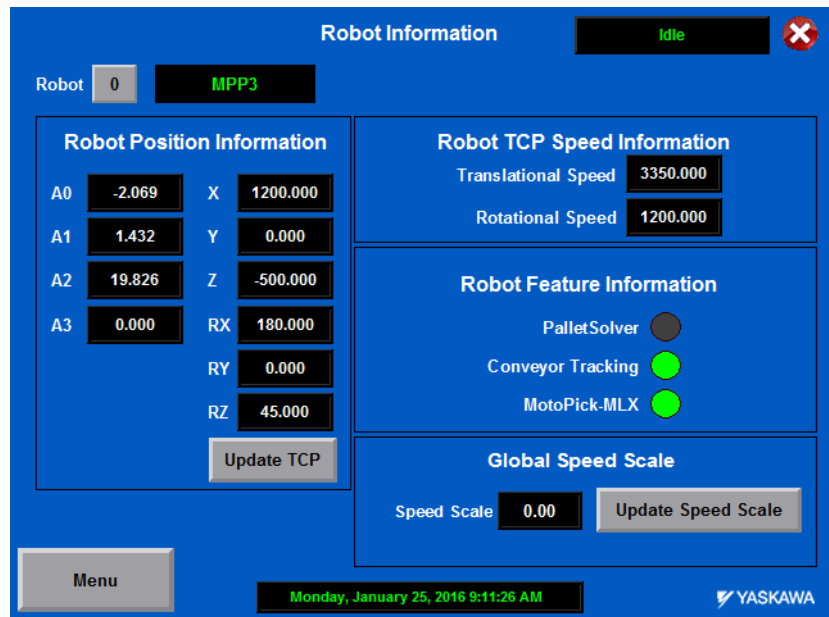
3.4.11 Brake Release Screen

The {Brake Release} screen can be used to release the brakes on any robot axis. The screen (*Fig.3-41 "Brake Release Screen"*) lists the available axes for each robot. To release a brake, all the signals listed under "Conditions Required to Release Brakes" should be green. Then, press and hold the brake release button to release the brake.

Fig. 3-41: Brake Release Screen
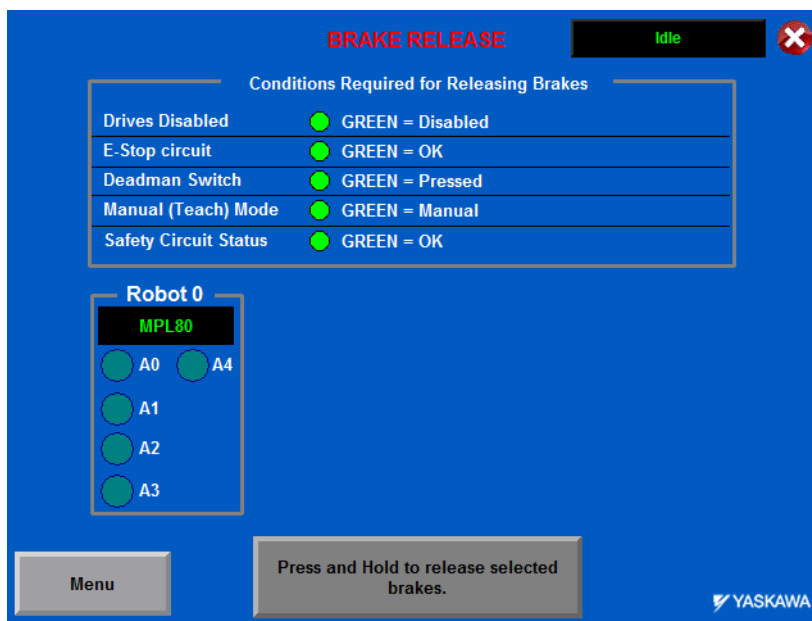
3.4.12 Interference Zone Status Screen

The Interference {Zone Status} Screen (*Fig.3-42 "IZ Status Screen"*) shows the current violation status of each of the 32 available Cubic Interference Zones as well as the current status of each zone: Not Active, Two Corners, or Center Pt. When a zone is violated, its status will turn red to indicate that the robot TCP is currently inside that zone. If no Interference Zones are configuring, all indicators will remain neutral.

Fig. 3-42: IZ Status Screen
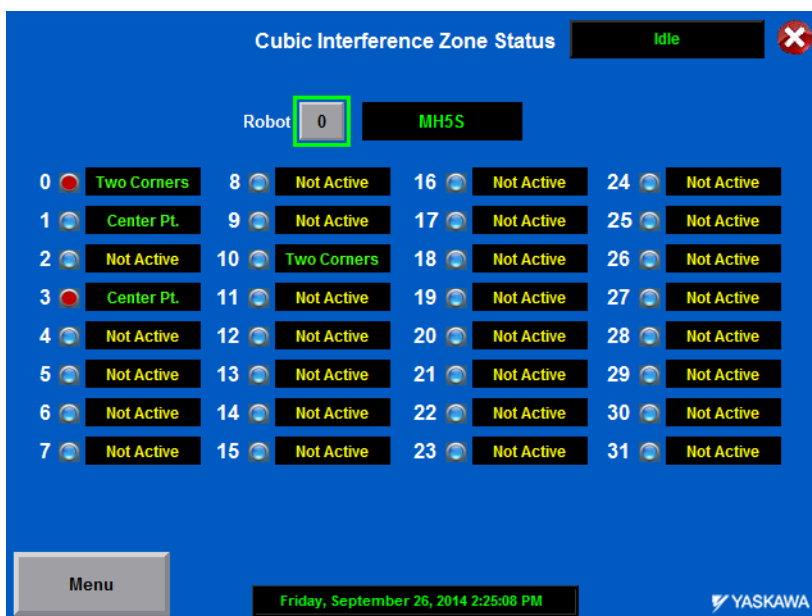
3.4.13 Information Screen

The Yaskawa {Information} Screen (*Fig.3-43 "Yaskawa Information Screen"*) can be accessed by clicking the Yaskawa logo on the {Main} Screen. This screen contains information on the MLX Control Module IP Address and Model as well as the firmware versions.

168542-1CD

Fig. 3-43: Yaskawa Information Screen
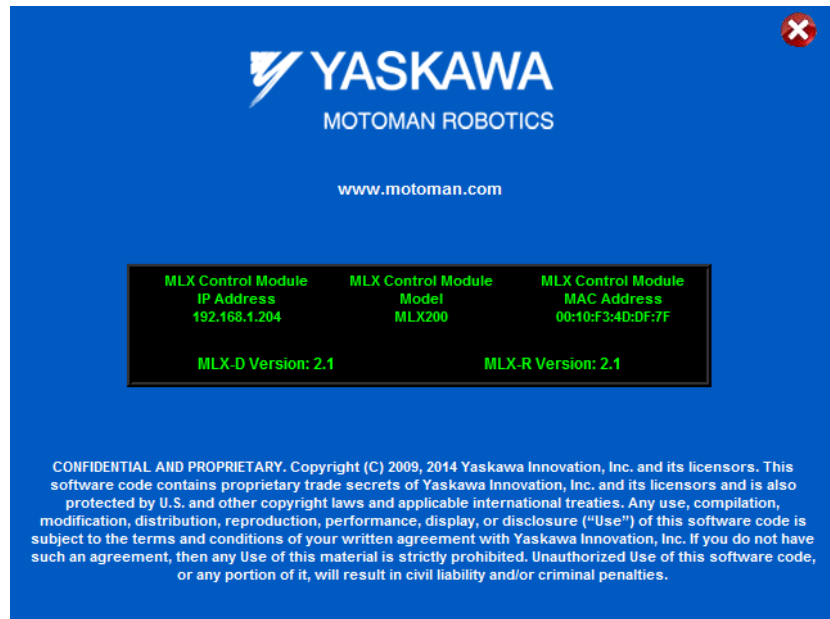
4 MLX200 Programming Guide

The previous sections have introduced the MLX200 data structures, instructions, and basic HMI operation. This section will demonstrate how to use these tools to teach positions and write MLX200 applications. The method to do this is slightly different than on traditional Robot Control Modules as the program itself is not developed from the teach pendant. Instead, the MLX-HMI is used to jog the robot and teach positions, and then these positions are accessed through a ladder program that is developed using the MLX200 instructions in RSLogix 5000. After describing a simple process for teaching and executing motions, the next section will describe how to incorporate Blend Factors into your program to reduce cycle times. Finally, some of the common best practices and potential pitfalls from programming MLX200 are discussed.

4.1 Developing a Simple Application

⚠️ **WARNING**

All MLX200 application code should be placed inside the MLX_Task. Failure to do so could lead to unexpected behavior such as skipped motions or motions being out or order after a Hold/Restart scenario. See *Section 3.3.1 "Task Scheduling" on page 3-9*.

This section will describe how to teach points with the MLX-HMI and then access these points from a ladder program to perform simple point to point motions.

4.1.1 Teaching Points with MLX-MHI

Section 3.4.1 "Setting Up the HMI" on page 3-21 introduced the Application data structure. This structure contains an array of type MLxAppDataJob that each contains an array of type MLxAppDataTeachPoint. Each Teach Point contains the following information:

- TeachPointName - optional user-description of the Teach Point

- TCPPosition - X, Y, Z, Rx, Ry, Rz position data as well as closure information

- AxisPosition - axis position data

- UserFrameNumber - The active User Frame when the point is taught (if not -1, the position will be converted to active user frame - see Section xxx)

- ToolNumber - The active Tool Number when the position was taught (used for information only)

All of this data will be populated when using the MLX-HMI Teach Screen for teaching points. First, the desired Job number should be entered on the top-left of the HMI. This will tell the system where in the ApplicationData structure to store the points. The jog controls can be used to move the robot into its desired position, and then the [Teach Position] button will store the current TCP/Joint positions into the tag structure based on which Position is highlighted. For example, *Fig.4-1 "Teaching a Point"* shows teaching Job 2 TeachPoint 12 (i.e.ApplicationData.Job[2].TeachPoint[12]). After the point has been taught, it can be accessed in the application logic as shown on the right side.

Fig. 4-1: Teaching a Point
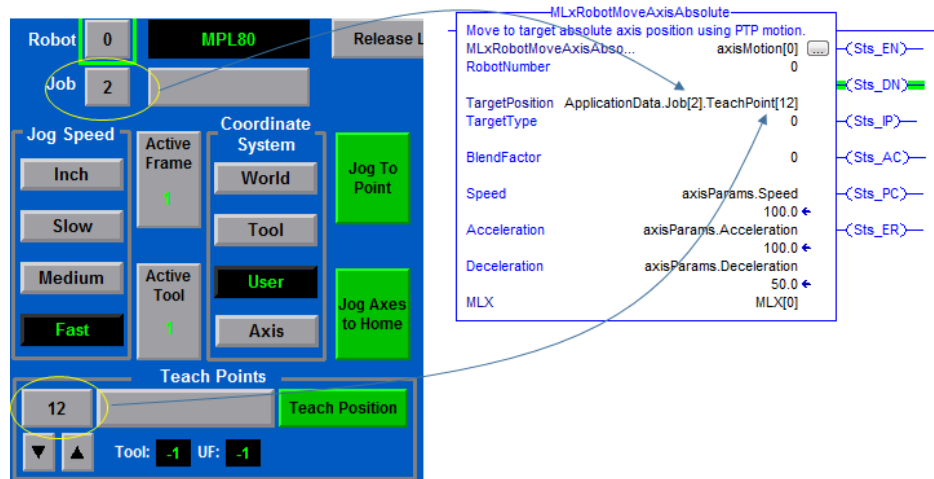
4.1.2   Accessing Taught Points From a Program

Once the desired points have been taught, they can be accessed through the MLX200 Motion Instructions. Each motion instruction takes a single Teach Point as input. It is often useful to create Aliases for points to provide a more descriptive/readable application program (*Fig.4-2 "Aliasing a Taught Point"*).

Fig. 4-2: Aliasing a Taught Point

168542-1CD
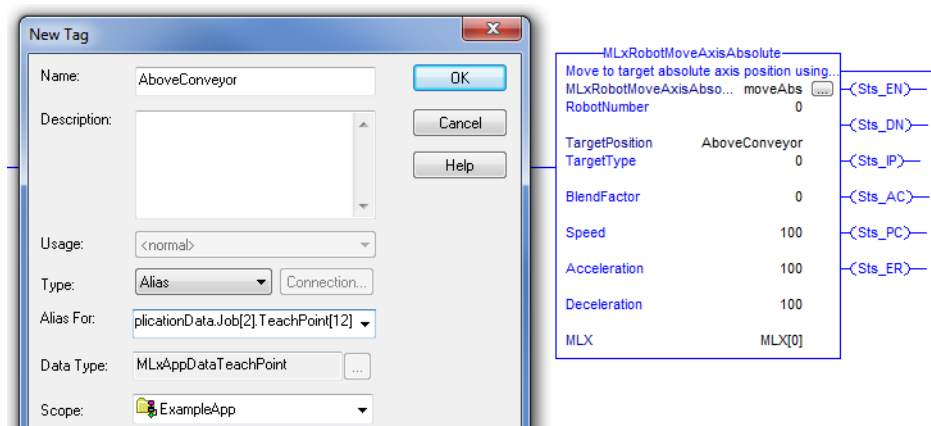
A recommended method of structuring a program is shown in *Fig.4-3 "Example Program Structure"*. In this structure, each rung of the program is given a step number. When this step is active, the motion instruction will be sent. Then, the step value is incremented when the instruction has completed (done by checking the Sts_PC bit). In the example shown, the program will loop between steps 10 and 20. The step numbers are defined to increment in 10s to allow for the insertion of additional rungs/points if the program is expanded.

Fig. 4-3: Example Program Structure
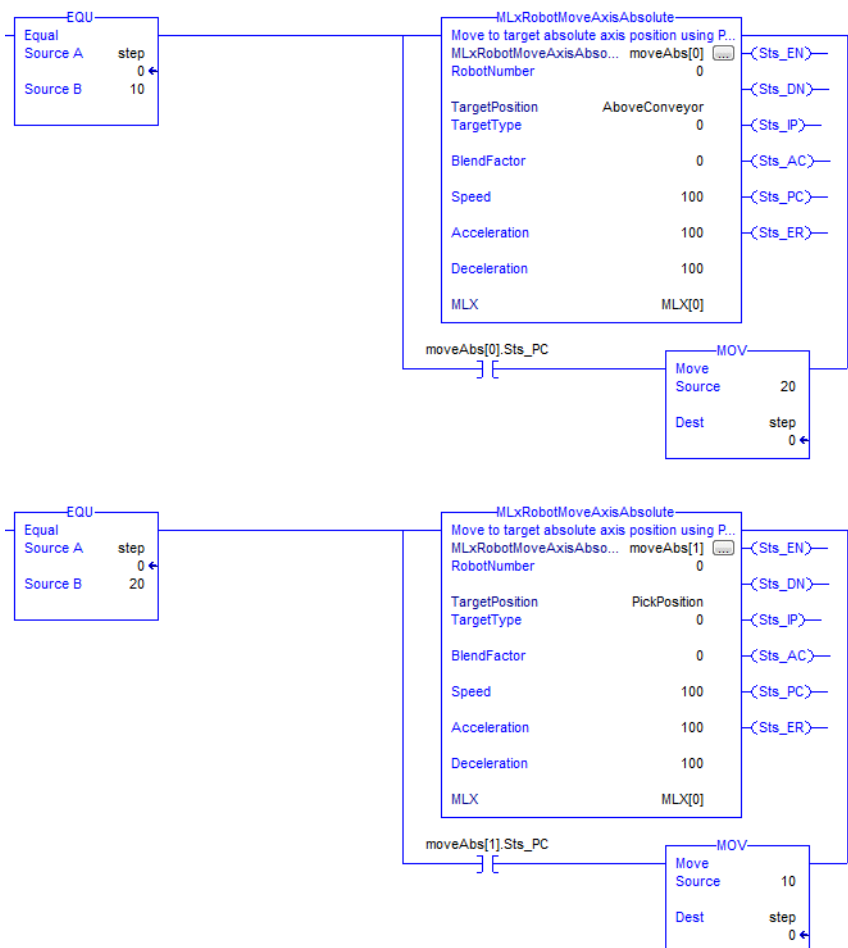
4.1.3 OPERATING A USER APPLICATION FROM HMI

After writing an application, it is often desirable to start and stop the application from the provided MLX HMI. To do this, the Main Screen has a "Start" button that is tied to the MLX[].UserStartProgram variable which can be used to initialize the application. The "Start" button will only appear on the screen in Idle mode, but it is generally good practice to double-check that the state is correct before starting as shown in *Fig.4-4 "Application Initialization from the MLX HMI"*.

Fig. 4-4: Application Initialization from the MLX HMI
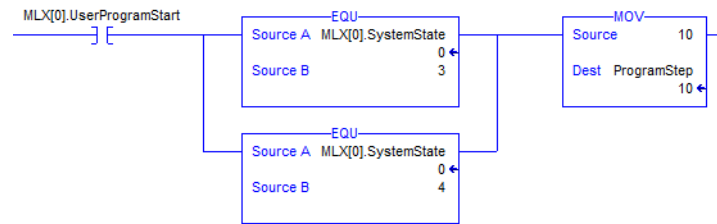
> **NOTE**
>
> It is also useful to have some logic to reset the program step; however, this logic will vary from application to application. For example, the program step could be reset to 0 whenever the system was aborted (MLX[].SystemState == 8) or whenever the system was reset to ServosOffReady (MLX[].SystemState == 12). Alternatively, the program step could be reset whenever the MLX200 Control Module was not connected, so that the program would only reset when the MLX200 Control Module was restarted.

After pressing the Start button from the HMI, this button will be replaced by the "Step Pause" button (left on *Fig.4-5 "Step Pause and Step Continue Buttons"*) and the value of MLX[0].UserStepAndContinue will be set to 1. When the "Step Pause" button is pressed, the "Step Continue" button will take its place and the value of MLX[0].UserStepAndContinue will be set to 0 (right on *Fig.4-5 "Step Pause and Step Continue Buttons"*).

Fig. 4-5: Step Pause and Step Continue Buttons
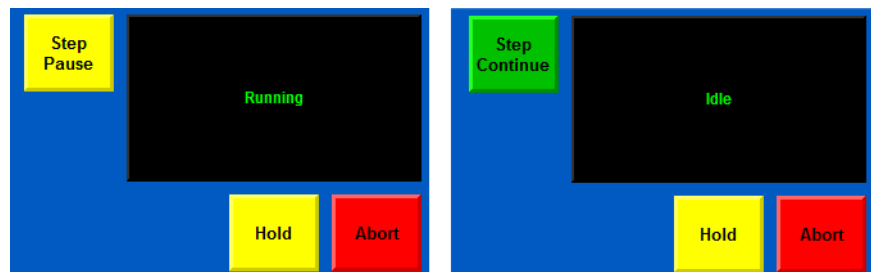
These buttons along with the UserStepAndContinue variable can be used to step through an application program by writing the application as shown in *Fig.4-6 "UserProgramStepContinue Functionality in Application Logic"*. Here, the application will only move to the next step if the UserStepAndContinue variable is turned on. Thus, the application will run normally when the Start button is pressed, but will stop at the end of the current rung when the Step Pause button is pressed. By pressing Step Continue, the program will start to execute normally again. Thus, toggling this button will allow a user to step through their application code.

Fig. 4-6: UserProgramStepContinue Functionality in Application Logic
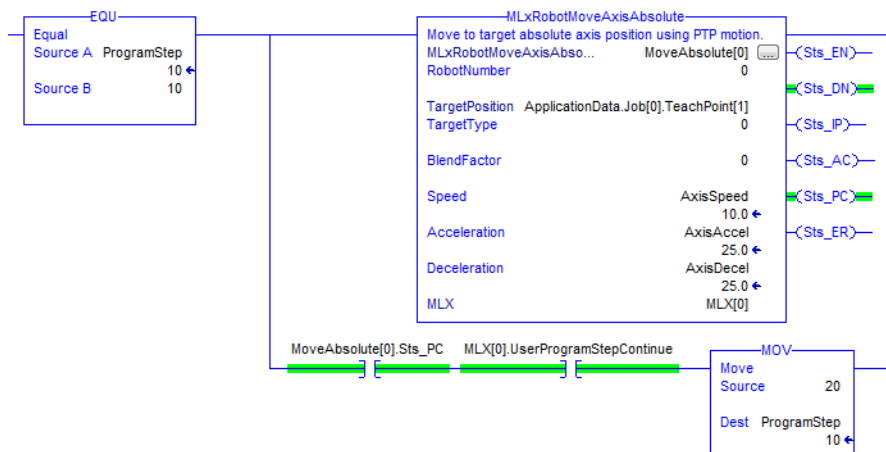
4.1.4 Teaching Points in User Frames

The previous sections described how to teach points in World Coordinates. However, it is often useful to teach points relative to a defined User Frame. For example, a User Frame could be attached to the corner of a pallet and then a sequence of motions could be taught relative to that frame. Then, the same program could be executed on another pallet by just changing the active User Frame.

To teach points relative to a User Frame, you must first define and activate a User Frame. A User Frame can be defined through the HMI screens described in *Section 3.4.7 "Tool and User Frame Screens" on page 3-31*. A frame can be activated either through the [Set User Frame] button on the HMI or by a call to the MLxRobotSetUserFrame instruction. After being activated, the User Frame number should appear in the MLX[].Robot[].ActiveUserFrameNumber [] variable which can also be seen from the HMI.

Fig. 4-7: User Coordinate System

Next, jog the robot to the desired position and switch the Coordinate System to "User". Once in User Mode, the TCP position reported on the HMI will switch to User Coordinates. Now, when the [Teach Position] button is pressed, these values relative to the User Frame will be stored in the Teach Point data structure along with the Active User Frame Number. When this Teach Point is passed into motion instruction with TargetType = 1, the system will move to the position relative to the active User Frame. Thus, the target position will change if the active User Frame is changed.

> **NOTE**
> • The User Frame number stored in the Teach Point is used only as a reference to the active User Frame when the point was taught. If the User Frame number is anything other than -1, the taught point will be converted into the active User Frame.
>
> • A Teach Point can be converted between World and User coordinates using the MLxRobotCoordinateTransform instruction.

4.1.5   USING REFERENCE POSITION VALUES

In some robots, a single TCP position (i.e. position and orientation of End-Effector) can be met by multiple axis positions. These multiple axis positions are often referred to as "closures" but can take other forms such as an axis with +/- 360 degree rotations. For example, *Fig. 4-8* shows a 6 axis robot with two different axis configurations that lead to the same TCP position.

Fig. 4-8: 6-Axis Robot with Different Closures

TCP Position: 660.7, 0.0, 190.1, 180.0, -90.0, 0.0
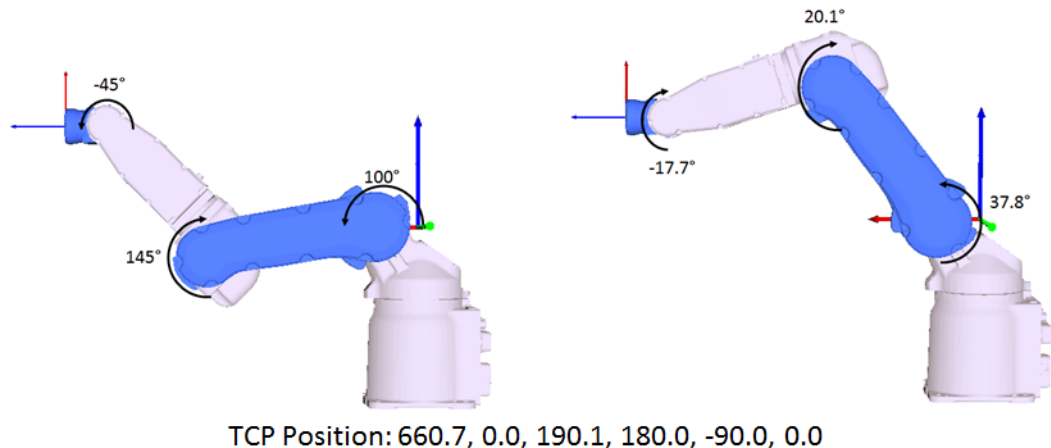
In these cases, the MLX system must be told what the expected axis position of the robot should be. This is done using the values inside "Closure" variable defined for each Teach Point (e.g. ApplicationData.Job[].TeachPoint[].TCPPosition.Closure.ReferencePosition). When a position is taught from the MLX HMI, these values will be automatically filled in with the current axis position and in most circumstances will never need to be changed. Then, when performing a linear motion to a target TCP position (TargetType = 1), the robot will attempt to move linearly to the target position and display an alarm if the target axis position cannot be reached through linear motion (e.g. because of a closure switch). This check is done to prevent unexpected motions

where the robot will end up at an axis configuration that is different from the position when the user taught the point.

For systems where the TCP Positions are calculated from outside sources such as camera systems, an all-zero Reference Position may be passed to allow MLX to determine the best solution. This will provide a good solution when one is available, but in certain applications it may be desirable to force a Reference Axis position check to prevent undesired behavior (e.g. interference with objects or cabling). The following sections will detail several example of using this variable in different scenarios.

4.1.5.1   Example 1: 6-Axis Robot

Consider a 6-Axis robot in one of the positions introduced before (*Fig. 4-9*). When the robot is jogged to this position and the Teach Position button is pressed from the MLX HMI, the TCP Position and Reference Position of the robot is automatically stored as shown below. Thus, by default, these positions should match.

Fig. 4-9: Closure Example

TCP Position: 660.7, 0.0, 190.1, 180.0, -90.0, 0.0

− ApplicationData.Job[6].TeachPoint[2]	{...}
+ ApplicationData.Job[6].TeachPoint[2].TeachPointName	'Closure Example'
− ApplicationData.Job[6].TeachPoint[2].TCPPosition	{...}
− ApplicationData.Job[6].TeachPoint[2].TCPPosition.TCPPosition	{...}
ApplicationData.Job[6].TeachPoint[2].TCPPosition.TCPPosition[0]	660.6703
ApplicationData.Job[6].TeachPoint[2].TCPPosition.TCPPosition[1]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.TCPPosition[2]	190.12463
ApplicationData.Job[6].TeachPoint[2].TCPPosition.TCPPosition[3]	-180.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.TCPPosition[4]	-90.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.TCPPosition[5]	0.0
− ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure	{...}
− ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition	{...}
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[0]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[1]	100.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[2]	145.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[3]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[4]	-45.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[5]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[6]	0.0

Next, suppose you want to command a linear motion to this position (either from the MLxRobotMoveLinear commands or by using JogToPoint from the HMI) from the initial positions shown in *Fig. 4-10*. Because there is a change in the elbow closure for this robot, this motion is not possible and you will see an error message similar to the one shown in *Fig. 4-11*. This prevents the robot from ending up at a position other than the point it was taught at.

Fig. 4-10: Motion with Elbow Closure Switch
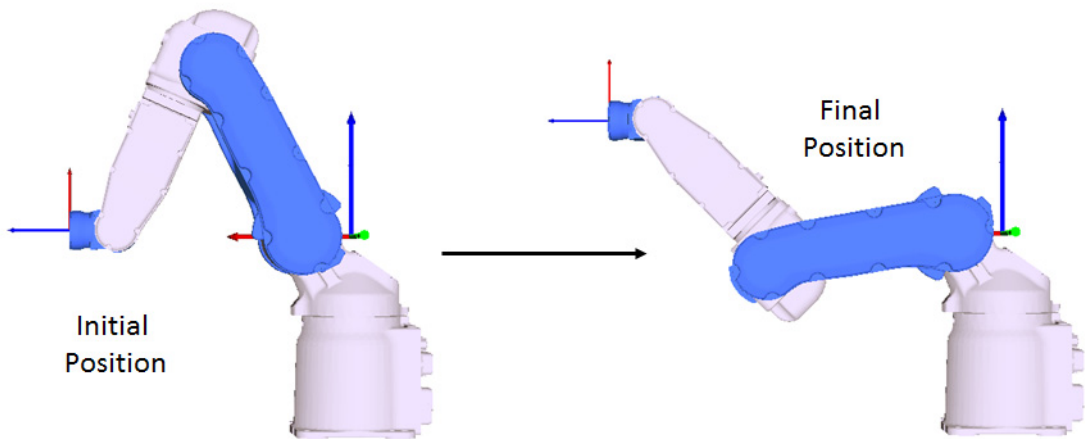
Fig. 4-11: Closure Error Message
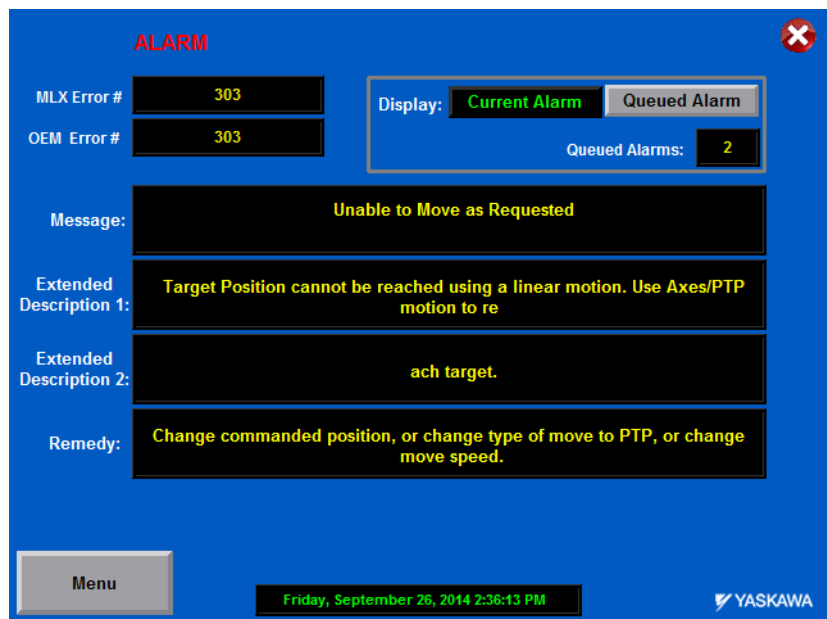
However, if the final axis position is not required to be met, a user may put an all-zero value into the Reference Position variable to tell the MLX system to ignore final axis position mismatch. In this case, the motion will be executed and the robot will reach the final position shown in *Fig. 4-10*. Note that the TCP position of the robot matches the TCP position of the target position in *Fig. 4-12*; however, the resulting axis position is different.

Fig. 4-12: Final Position with Zero Reference Position

⊟ ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure	{...}
⊟ ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition	{...}
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[0]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[1]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[2]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[3]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[4]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[5]	0.0
ApplicationData.Job[6].TeachPoint[2].TCPPosition.Closure.ReferencePosition[6]	0.0

20.1°

-17.7°

37.8°

TCP Position: 660.7, 0.0, 190.1, 180.0, -90.0, 0.0

NOTE
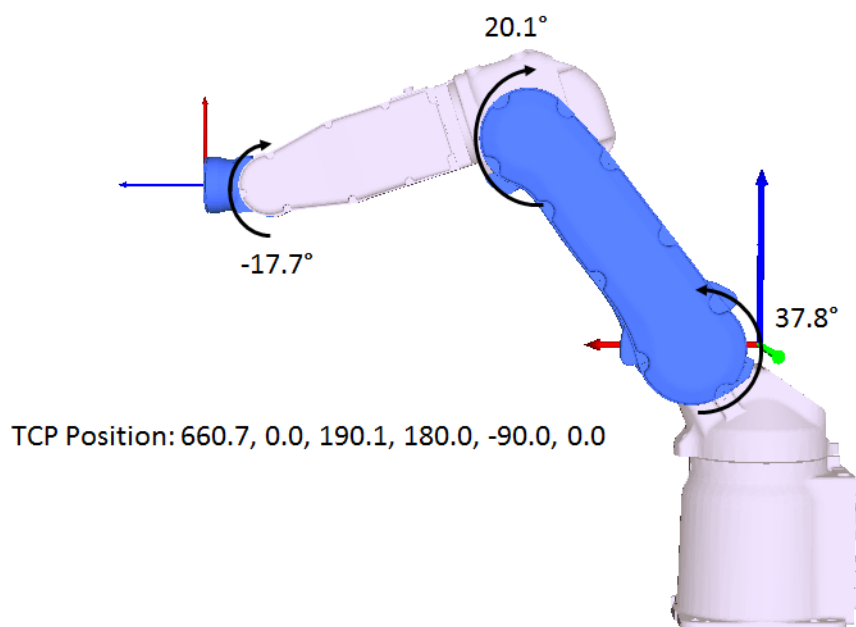
- If it is necessary to reach the Reference Position, an MLxRobotMoveAxis command or a JogToPoint in Axis mode can be used.

- Some closure switches can be reached through a linear motion. MLX will only display an alarm if the closure switch cannot be reached.

- If the X,Y,Z,Rx,Ry,Rz values of a teach point are manually changed/tweaked, it is possible (though unlikely) for the closure to switch. In this case, it is a good practice to "reteach" the position after tweaking the value to get the TCP and Reference Axis Position to match.

4.1.5.2   Example 2: 4-Axis Palletizing Robot

While the 4-Axis Palletizing robots do not have different closures within their work-spaces, they do have a +/- 360 degree range on their T-Axis which can lead to multiple axis solutions. In this case, the Reference Position can be used to control how the T-Axis will move during MLxRobotMoveLinear commands. Consider the 90 degree rotation between two points shown in Figure 6. Here the Reference Position acts to determine the final T-Axis solution for the robot. In this case, it will move 90 degrees to the target position.

Fig. 4-13: 90 Degree T-Axis Rotation

ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition	{...}
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[0]	0.0
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[1]	0.0
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[2]	0.0
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[3]	-90.0

However, the same TCP position can be reached by rotating the T-Axis in the opposite direction 270 degrees (+/- 360 degrees will yield the same Rz position). Thus, this opposite rotation can be achieved by supplying the reference position shown below.

Fig. 4-14: 270 Degree T-Axis Rotation

ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition	{...}
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[0]	0.0
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[1]	0.0
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[2]	0.0
ApplicationData.Job[6].TeachPoint[11].TCPPosition.Closure.ReferencePosition[3]	270.0

> **NOTE** In the case of an all-zero Reference Position, the robot will take the shorter rotation direction.

4.1.5.3   Summary

The following table provides a brief summary of how MLX will use the Reference Position in different scenarios. Here are some basic guidelines for using the Reference Position variables:

- If using taught positions, keep the TCP Position and the Reference Position up to date if TCP Positions are modified/tweaked manually. This can be done by re-teaching the points after any modifications are made.

- If using TCP positions from an outside source (e.g. camera), set the Reference Position to all-zero values for initial testing. In certain cases (for example cable interference on the T-Axis), these values will need to be further tweaked on an application by application basis.

- Use Axis motions whenever strict linear motion is not necessary to prevent issues with closures or redundant T-Axis positions.

Table 4-1: Summary of Reference Position Usage

	Reference Position	Result
5 or 6 Axis Robots	Same closure as initial position, or a closure switch that does not prevent linear motion	Motion successful. Final axis position will be equal to Reference Position if from taught positions.
	Closure switch from initial position that prevents linear motion	Motion unsuccessful, alarm displayed.
	All-zero values	Motion successful. Final axis position will be in same "closure" as initial position.
4 Axis Palletizing Robots	Non-zero values	Motion successful. Final T-Axis position will be equal to T-Axis position in Reference Position if from taught positions.
	All-zero values	Motion successful. T-Axis position will be T-Axis that results from shortest rotational motion.

4.2   Configuration Instructions

Configuration Instructions refer to the AOIs that modify the current state of the robot system but are not a motion command. These include:

• MLxRobotSetBasePose

• MLxRobotSetToolProperties

• MLxRobotSetUserFrame

• MLxRobotSetCubicIZ

• MLxRobotFrameShift

4.2.1   Using Configuration Instructions

The parameters for these configuration instructions can be set/saved from the MLX-HMI (see *Section 3.3.9 "Using Global Speed Scale" on page 3-19*) for more information on using the HMI). When these parameters are saved from the HMI, they are stored at the proper location inside the ApplicationData tag structure and will stay resident on the PLC until they are changed. The HMI also allows for directly executing a configuration instruction. This is useful for testing and verifying that the parameters have been entered correctly; however, these configuration changes will be reset if the MLX-R application is restarted or the MLX200 Control Module is rebooted or reset. For this reason, it is recommended that all configuration instructions related to particular application be set during that application's initialization routines.

> NOTE   After saving new configuration data on the HMI, be sure to upload all tags and save the RSLogix project to make sure the data is saved inside the project file.

4.2.2   Setting Multiple Configuration Instructions

Applications will often need to set multiple configuration instructions during initialization. These instructions can be placed on a single rung as shown in *Fig.4-15 "Multiple Configuration Instructions"*. However, it is often useful to place them on different rungs just to make the program more readable.

Fig. 4-15: Multiple Configuration Instructions
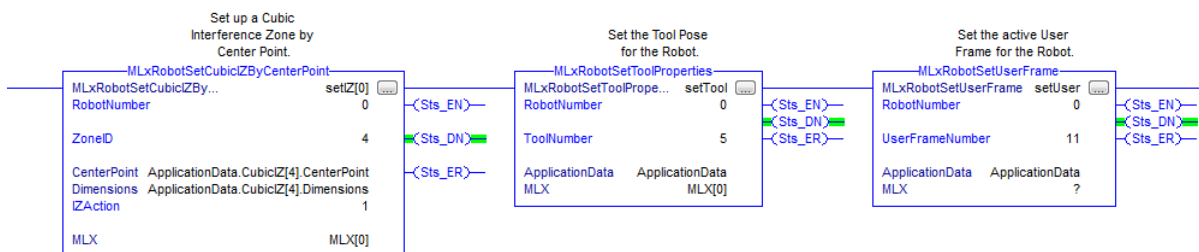
4.2.3 Using Configuration Instructions with Motions

The above sections have described how to set up Configuration Instructions during your application initialization code. However, these instructions are often required during the actual job execution (e.g. to change the Tool Pose after picking up an object). Thus, it is important to understand how these instructions interact. The main difference between Motion Instructions and Configuration Instructions is that the Motion Instructions are added to a motion queue whereas the Configuration Instructions are executed as soon as they are scanned. Thus, in the example code shown in *Fig.4-16 "Incorrect usage of Linear Motion and MLx Robot Set Tool Properties combined"*, the first Linear Motion will be added to the motion queue and will begin executing. Then, as soon as the SetTool is scanned, the Tool Pose of the robot will instantly change. To prevent these Configuration Instructions from causing a change to the robot's position while executing a motion, MLX200 will display the error shown in *Fig.4-17 "Configuration Instruction Execution Error"* in this instance. The following Configuration Instruction commands will produce a similar alarm if placed between Motion Instructions:

- MLxRobotSetToolProperties

- MLxRobotSetBasePose

- MLxRobotSetCubicIZByTwoCorners

- MLxRobotSetCubicIZByCenterPoint

- MLxRobotSetProperties

Fig. 4-16: Incorrect usage of Linear Motion and MLx Robot Set Tool Properties combined
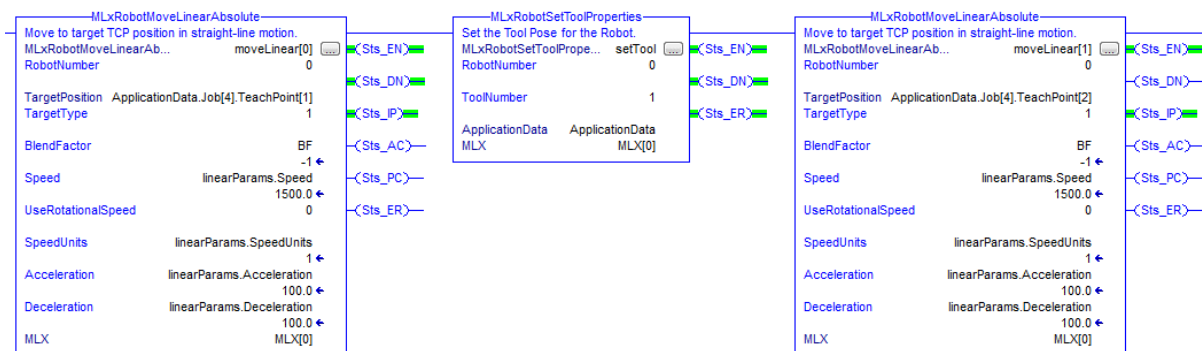
Fig. 4-17: Configuration Instruction Execution Error

This error can be removed by placing the SetTool as the first command on the ladder rung as shown in *Fig.4-18 "Correct Usage of Linear Motion with Tool Change"*. In this case, the new Tool Pose will become active before the Linear Motions are added to the motion queue. Alternatively, this SetTool could be placed on its own ladder rung and the next step triggered off of its STS_DN bit.

> **NOTE**
>
> The other instructions (e.g. setting User Frames or Frame Shifts) can be performed while the robot is executing as they will only apply to motions queued afterwards. Thus, these commands can be placed in-between motion instructions.

Fig. 4-18: Correct Usage of Linear Motion with Tool Change
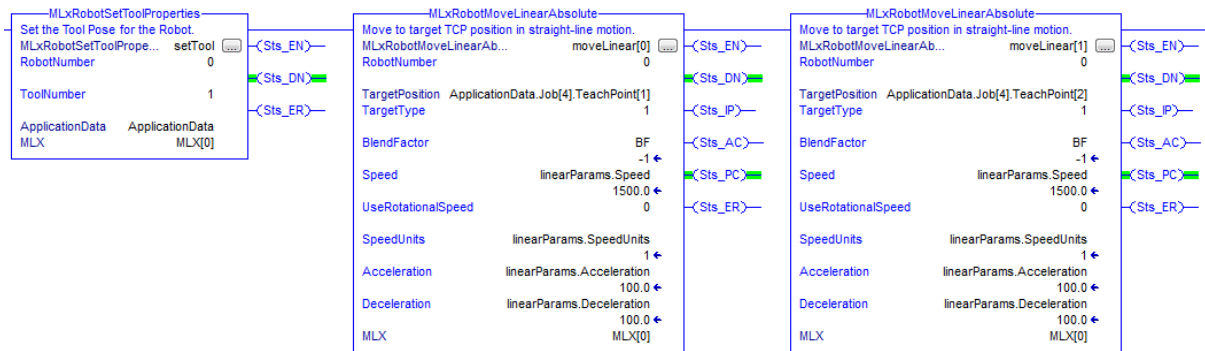
4.3   Using Blend Factors

4.3.1   PC Bit Triggering

As mentioned in the previous section, the simplest way to write an MLX200 program is to use the motion instruction PC bit to trigger the next motion. In this structure, each MLX200 motion instruction should be placed on its own rung of ladder. Once the motion has finished executing (i.e. PC bit becomes high), the program will increment its step and move onto the next ladder rung. An example of a rung of ladder using PC bit triggering is shown in *Fig.4-19 "PC Bit Triggering"*. This provides a very clean and easy to read ladder program with the major disadvantage of being that the Blend Factor parameter cannot be used to reduce cycle times.

Fig. 4-19: PC Bit Triggering

4.3.2   Sequential Motion Instructions

To use blend factors, each motion instruction should be placed on the same rung of ladder as shown in *Fig.4-20 "Sequential Linear Motions with Blend Factor"*. This is done to keep each AOI Enabled so that its status can be properly updated until it is finished executing. In the example below, the first motion instruction is given a Blend Factor of 4. This means that as moveLinear[0] is executing its motion toward Job[4].TeachPoint[1] it will begin to blend into moveLinear [1]'s motion to Job[4].TeachPoint[2]. Note that in this case, the Blend Factor for moveLinear [1] does not matter as we are waiting for this motion to finish executing before moving to a next step. However, if an additional motion were placed after moveLinear [1], multiple points could be blended. In MLX200, up to 25 motions can be queued at a. *Fig.4-21 "Parallel Axis Motion Instructions"* shows an equivalent way to do this by using parallel motion instructions on the same rung of ladder.

Fig. 4-20: Sequential Linear Motions with Blend Factor

Fig. 4-21: Parallel Axis Motion Instructions

4-16
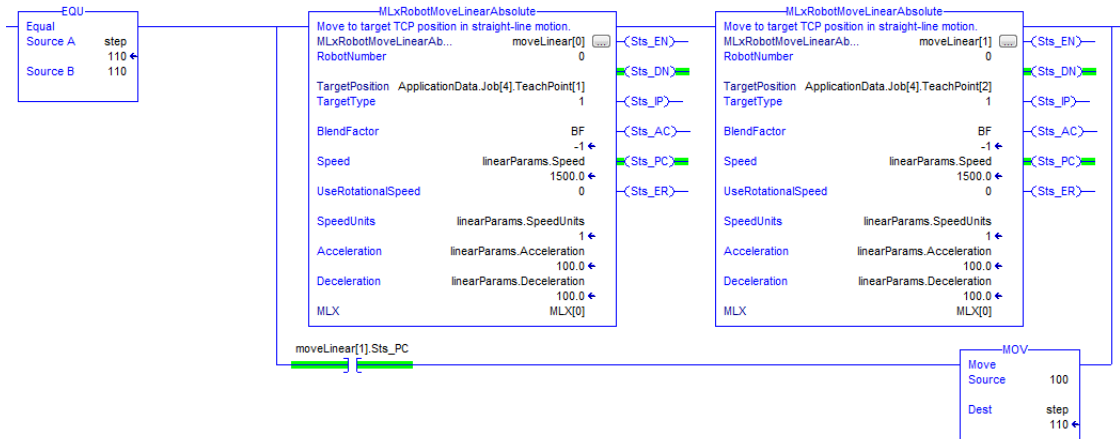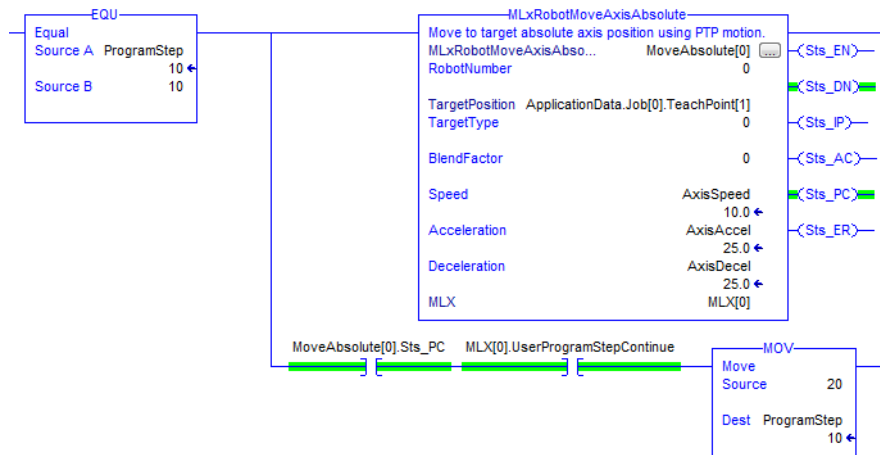
Multiple motion instructions of different types can also be placed on the same rung of ladder as shown in *Fig.4-22 "Sequential Axis and Linear Motions"*. In this case, 2 Axis Motions and 2 Linear Motions will be queued, and the Axis Motion will blend into the Linear Motion using PTP (axis) interpolation.

Fig. 4-22: Sequential Axis and Linear Motions

4.4 Programming Pitfalls and Best Practices

This section will describe some potential problems and issues that can be encountered by incorrect use of the MLX200 instructions.

4.4.1 Incomplete AOI Executions

To understand the potential pitfalls and problems that can come from programming in MLX200, a good understanding of the basic life-cycle of a motion instruction (AOI) is necessary. The basic steps of the AOI are:

1. When an AOI's ladder rung becomes Enabled, the EN and IP bit will become active and all other bits will turn off. At this stage, the AOI will secure an unused internal Buffer Index (up to 25 buffered motions allowed) to hold the specific motion instruction parameters as well as status information.

2. Once the hand-shaking with the MLX200 Control Module is completed and the motion has been queued, the DN bit become active. At this point, the internal Buffer Index is released and the instruction has finished processing and can become disabled. However, the AC/PC bits will only keep updating if the rung stays enabled.

3. When the motion begins executing, AC bit goes high.

4. Once the motion has finished executing, the AC bit goes low, and the PC bit goes high.

5. When the AOI becomes disabled, the DN bit will also go low.

The caveat in this process is that the rung must be enabled until *step 2* has completed and the AOI has had a chance to release its Buffer Index. If the rung becomes disabled before the hand-shaking is complete, the instruction will be left in an indeterminate state. A common programming mistake that can lead to this issue is to trigger a motion instruction directly off of an I/O signal as shown in *Fig.4-24 "Motion Instruction Trigger from an I/O Signal Correct"*. In this case, the application logic says to issue the motion command as soon as the I/O signal linked to Local:1:I.Data.0 is turned on. However, if this I/O signal only turns on for an instant or flickers off temporarily, the motion instruction could be left in an indeterminate state. One potential work around for this is shown in *Fig.4-23 "Motion Instruction Triggered from an I/O Signal Incorrectly"*. Here, a variable called "issueMotion" is latched when the I/O signal is turned on and then unlatched after the motion is complete. This ensures that the instruction stays enabled throughout its execution.

168542-1CD

Fig. 4-23: Motion Instruction Triggered from an I/O Signal Incorrectly

Local:1:I.Data.2

MLxRobotMoveAxisAbsolute
Move to target absolute axis position using PTP motion.
MLxRobotMoveAxisAbso...   moveAxes[0]   ⟨Sts_EN⟩
RobotNumber   0   ⟨Sts_DN⟩
TargetPosition   ApplicationData.Job[4].TeachPoint[2]   ⟨Sts_IP⟩
TargetType   0
BlendFactor   0   ⟨Sts_AC⟩
Speed   100   ⟨Sts_PC⟩
Acceleration   100   ⟨Sts_ER⟩
Deceleration   100
MLX   MLX[0]

moveAxes[0].Sts_PC

MOV
Move
Source   110
Dest   step
               110 ←

Fig. 4-24: Motion Instruction Trigger from an I/O Signal Correct

Local:1:I.Data.2   issueMotion
                                                               ⟨L⟩

issueMotion

MLxRobotMoveAxisAbsolute
Move to target absolute axis position using PTP motion.
MLxRobotMoveAxisAbso...   moveAxes[0]   ⟨Sts_EN⟩
RobotNumber   0   ⟨Sts_DN⟩
TargetPosition   ApplicationData.Job[4].TeachPoint[2]   ⟨Sts_IP⟩
TargetType   0
BlendFactor   0   ⟨Sts_AC⟩
Speed   100   ⟨Sts_PC⟩
Acceleration   100   ⟨Sts_ER⟩
Deceleration   100
MLX   MLX[0]

moveAxes[0].Sts_PC   issueMotion   MOV
                            ⟨U⟩   Move
                                             Source   110
                                             Dest   step
                                                            110 ←

4.4.2 DN BIT Checking

⚠️ **CAUTION**

It is recommended that application logic is written based on the Sts_PC bit. If using the Sts_DN bit, read this section carefully to be informed on possible issues.

As mentioned in the previous section, an instruction will only need to stay enabled until its Sts_DN bit it turned on. However, there are several things to look out for if using the Sts_DN to control application flow instead of the Sts_PC bit. Consider the rung of ladder shown in *Fig.4-25 "DN Bit Triggering"*. In this example, a motion instruction is called when the step=100 and then the step is incremented to 110 as soon as the Sts_DN bit turns on. This ladder rung will become disabled as soon as the motion is finished processing. In this case, the execution of the motion as well as the internal clean-up will be handled; however, the other status bits of the motion cannot be updated if the rung is not enabled (i.e. no Sts_AC or Sts_PC bits). These bits should not be used to control any other logic in the application.

Fig. 4-25: DN Bit Triggering
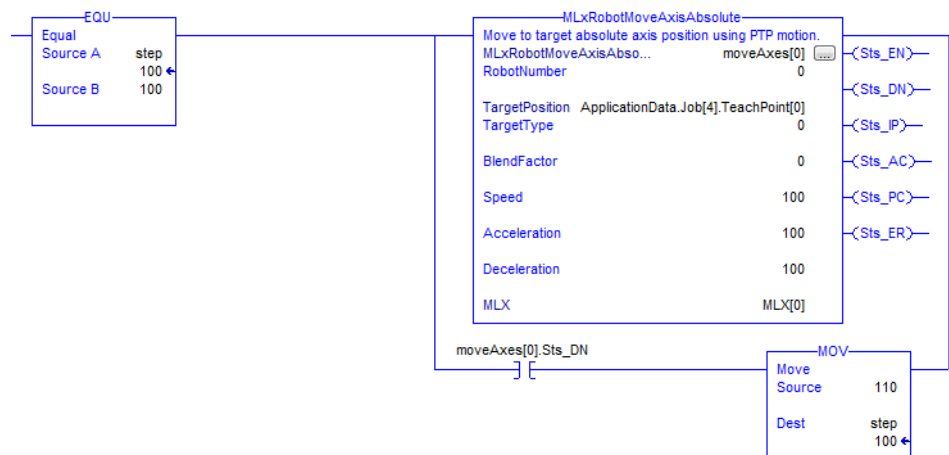
A second issue with Sts_DN bit triggering is that it is easy to accidentally fill the internal motion buffer this way. The hand-shaking between MLX-D and MLX-R will only take on the order of 10 ms, so if the application had two rungs going back and forth based on the Sts_DN, the maximum queue size of 25 will be reached very quickly. In this case, the error shown in *Fig.4-26 "Motion Queue Full Error"* will be displayed on the HMI.

Fig. 4-26: Motion Queue Full Error
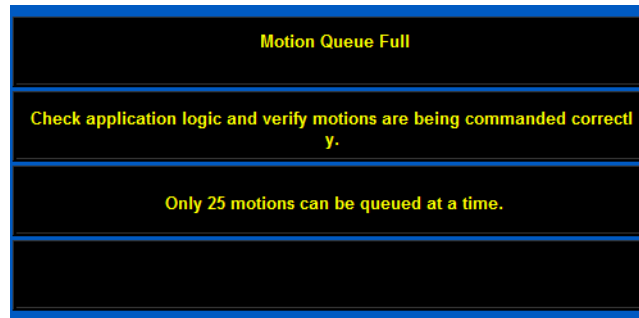
4.4.3   Reused Control Variables

Another potential problem comes from reusing the control variables for an AOI. For example, if you are triggering an MLxRobotMoveAxisAbsolute using the control variable moveAbs[0], this variable cannot be used anywhere else even if it is on a rung that is never active. The reason for this is that every AOI has a special "Enable False" condition that allows it reset its state whenever it is not enabled to be prepared for the next time it is called. If you use the same control variable twice, it will be attempting to execute the instruction in one location and attempting to reset the instruction in the other. This will lead to unresponsive behavior from the system. If an instruction is misbehaving, a quick check can be done by right-clicking on the control variable and selecting "Cross Reference" to see if the variable is reused anywhere else.

> **NOTE**
>
> A control variable can be reused in an UNSCHEDULED program as long as the two programs will never be scheduled at the same time.

Fig. 4-27: Cross-Reference Variable Use
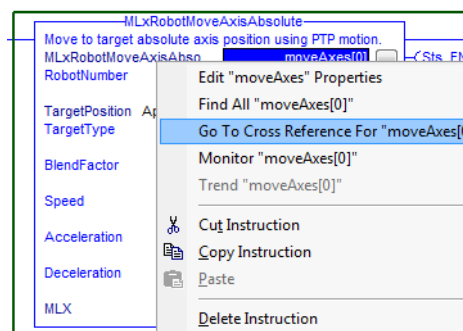
4.4.4 Task Overlaps and CPU Load

MLX200 is designed to run on a variety of CompactLogix and ControlLogix platforms including several older models/processors. On some systems, the RSLogix application can become unresponsive if application tasks (either MLX or other tasks) are taking up too much of the CPU Load. There are two ways to check this. The first is to right-click on the MLX Task, select Properties, and open the Monitor Tab (*Fig.4-28 "Task Overlap Count"*). This page contains a "Task Overlap Count" that tells whether the task is executing in the proper amount of time. If this value is non-zero, go to the Configuration Tab and increase the Task Period until all overlaps go away.

Fig. 4-28: Task Overlap Count
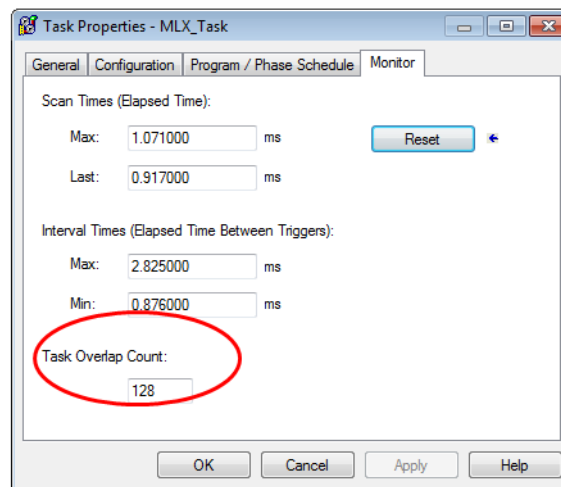
RSLogix 5000 also has a tool called the Logix 5000 Task Monitor Tool that can be used to further diagnose a system. The Task Monitor Tool is an optional item during the RSLogix installation, but can be found for download on the Rockwell website if needed. It can be started under the Tool Menu.

Fig. 4-29: Logix 5000 Task Monitor Tool

ID	Name	Out Update	Task Type	Rate	CPU	Priority	Last Scan	Max Scan	Watchdog	Overlap	Status	% to Rate
1	MainTask	Y	Continuous	10.00	41.39%	10	0.02	10.95	500.00	0	Running	N/A
2	MLX_Task	Y	Periodic	1.00	46.82%	1	0.44	0.69	500.00	0	Running	68.80%
3	MLX_HMITask	Y	Periodic	20.00	3.68%	10	1.54	5.63	500.00	0	Running	28.17%

168542-1CD

5 Collision Detection

5.1 Collision Detection Overview

> ⚠️ **CAUTION**
>
> This function does not completely avoid damage to the peripheral devices; moreover, it does not guarantee the user's safety.
>
> • Make sure to prepare safety measures such as safeguarding, etc.
>
> Failure to observe this caution may result in damage to machinery caused by contact with the manipulator.

The MLX200 Collision Detection feature provides a mechanism to provide extra protection to the robot and surrounding equipment in the case of accidental collisions. This is done by measuring the torque "disturbance" (i.e. difference between measured and expected torque) of each axis during a particular application or operation. If the torque disturbance becomes larger than a configurable maximum, the system will abort with a Collision Detection Error (error code 528). The basic steps for configuring and enabling Collision Detection are:

1. Start Collision Detection in Measurement Mode from either the MLX-HMI or by directly calling the MLxRobotCollisionDetection instruction.

2. While in Measurement Mode, run the system through a typical application cycle. While running in this mode, the system will report back the Measured Torque Disturbance values. The application cycle should be run for at least 30-60 minutes to provide reliable data.

3. Stop Measurement Mode from either the MLX-HMI or the MLxRobotCollisionDetection instruction, and determine suitable Allowable Torque Disturbance values for the application. These values must be larger than the original Measured Torque Disturbance values to prevent false alarms, and the sensitivity of the system can be adjusted by increasing/decreasing the Allowable Torque Disturbance values. A recommended starting point is Allowable Torque Distance = Measured Torque disturbance + 15.

4. At the beginning of the application, use the MLxRobotCollisionDetection instruction to start the system in Execution Mode with the determined Allowable Torque Disturbance values. Collision Detection will now be enabled until another call from MLxRobotCollisionDetection disables it or the MLX PLC Interface Application is restarted.

The following section will describe how to configure Collision Data from the MLX HMI (*Section5.2*) and how to use the MLxRobotCollisionDetection instruction (Section 5.3).

> **NOTE**
>
> • It is also possible to use multiple calls to MLxRobotCollisionDetection inside a single application to change the Collision Detection behavior during different phases of the application (e.g. a pick/place operation).
>
> • When in Manual Mode, the system will use a default Allowable Torque Disturbance value. This will be enabled whenever Collision Detection is running.

5.2   Configuring Collision Detection from the HMI

Fig. 5-1 shows the main {Collision Detection Configuration} screen. This screen has buttons for starting/stopping Measurement Mode as well as displays the current Measured and Allowable Torque Disturbance values. At the top, a Collision "File" can be selected which corresponds to the index inside the ApplicationData.CollisionDetect[] data structure where the current data will be stored. As the process of measuring and configuring Collision Detection is executed, the data will automatically be stored inside this data structure which will allow re-use in the application.

Fig. 5-1: Collision Detection HMI Screen
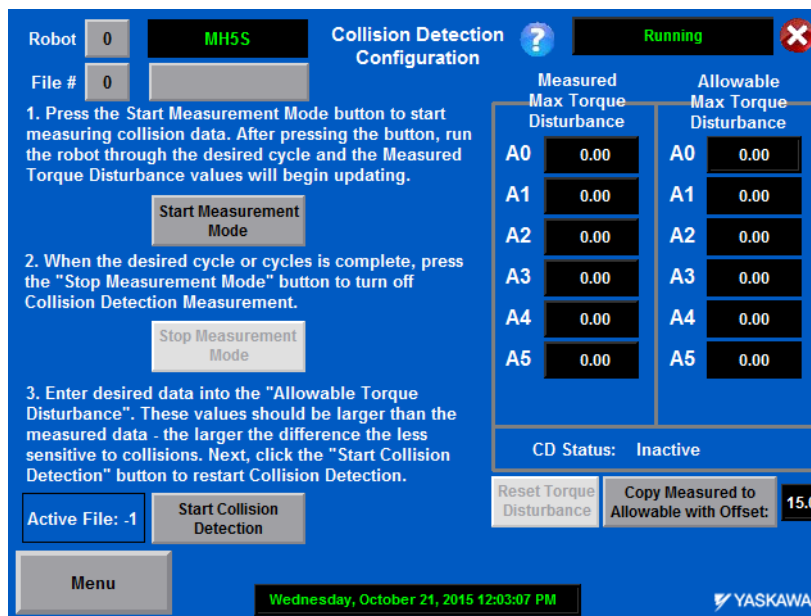
⚠️ **CAUTION**

To get accurate data during Measurement Mode, the tool load must be entered and activated from either the {Tool Properties HMI} screen or the MLxRobotSetToolProperties instruction.

⚠️ **CAUTION**

Tool Load must not exceed rated values for the robot.

To begin configuring Collision Detection, first select the File # to store the data and give it a name. Here, File #2 is used and given the name "VacuumGripper1". Then, press the [Start Measurement Mode] button (Figure 2) and begin running the system through the desired application cycle. An extra pop-up window will show up after pressing [Start Measurement Mode] button as a reminder that the tool mass properties must be set before beginning measurement to ensure correct measurement data.

Fig. 5-2: Start Collision Detection in Measurement Mode

After starting Measurement Mode, the [Start Measurement Mode] button will be grayed out and the "CD Status" indicator on the right part of the screen will update to display "Measurement Mode" as shown in *Fig. 5-3*. While the application is executing, the numerical values under the "Measured Max Torque Disturbance" will begin to automatically update. These values usually become fairly stable after a few application cycles, but it is recommended to run the cycle for 30-60 minutes to get reliable data.

> **NOTE**
> If the speeds or accelerations of the application are changed after configuring Collision Detection, the configuration may not be reliable. It is recommended to re-configure Collision Detection if changes to motion profiles are made.

Fig. 5-3: Collision Detection in Measurement Mode
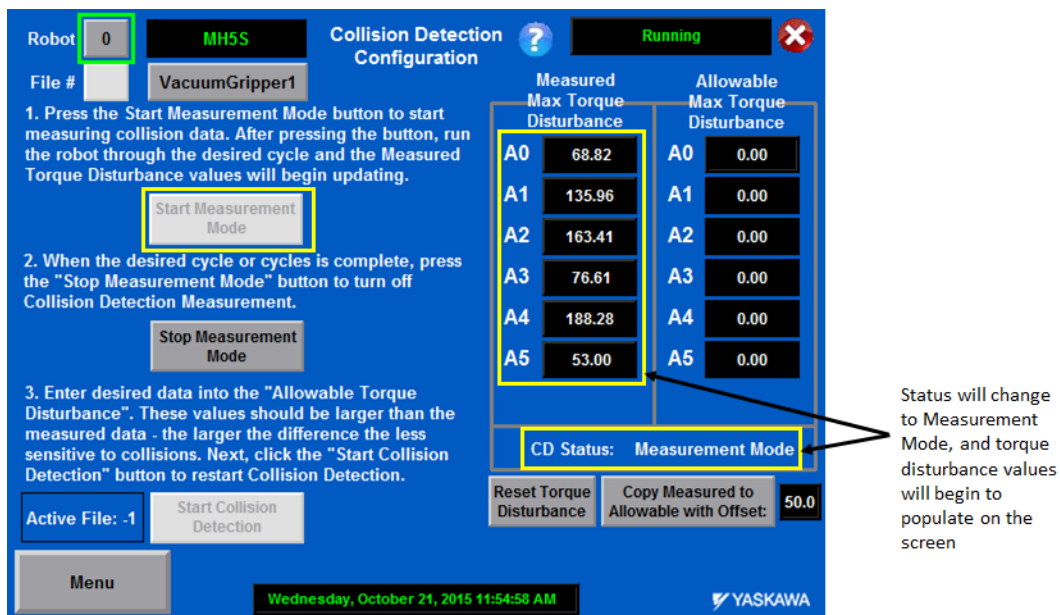
After measuring the torque disturbance values, press the [Stop Measurement Mode] button and the CD Status should change back to "Inactive". The next step is to fill in the Allowable Max Torque Disturbance values based on the measured values. This can be done either by manually entering values into the numerical inputs under Allowable Max Torque Disturbance or by using the [Copy Measured to Allowable with Offset] button. In *Fig. 5-4*, this offset value is set to 10 and the Allowable Disturbances are all then set to 10 higher than the Measured Disturbances. Note that these same values will also be stored inside the ApplicationData structure as shown in *Fig. 5-5*. After setting the Allowable Disturbance values, the [Start Collision Detection] button can be pressed to start running Collision Detection.

Fig. 5-4: Set Allowable Torque Disturbance Values

After Allowable
values have been
set, press Start
Collision Detection

Press button to
copy Measured
values to Allowable
values
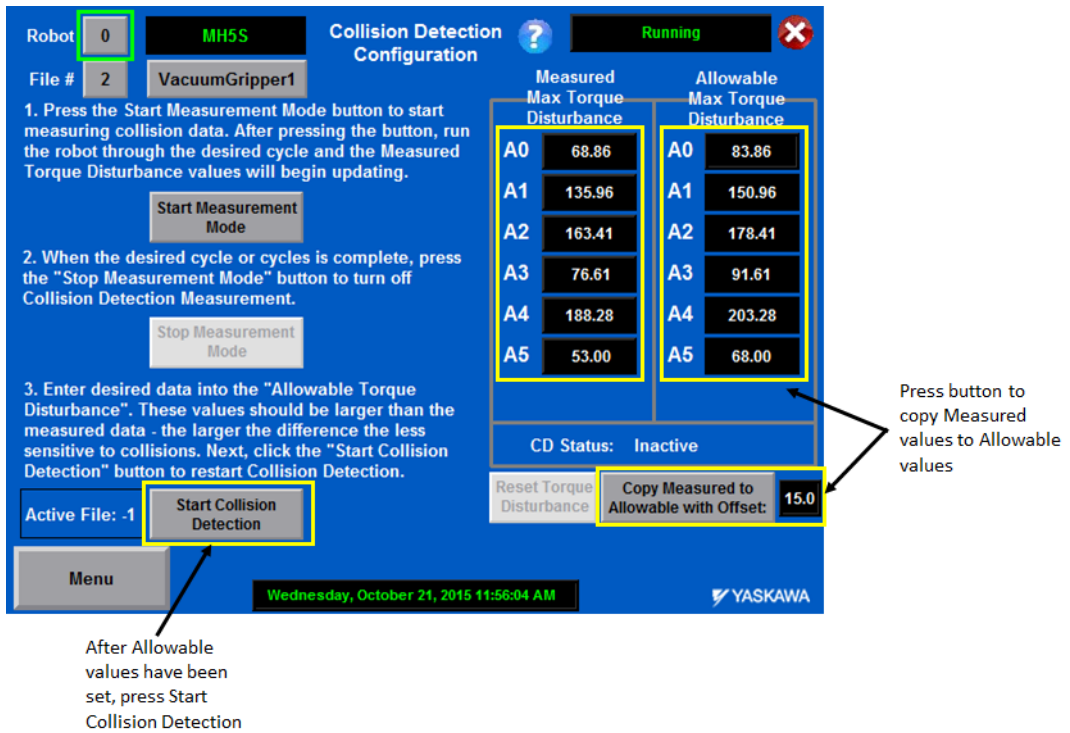
Fig. 5-5: Collision Detection Application Data

5   Collision Detection
5.2 Configuring Collision Detection from the HMI

After starting Collision Detection, there will be three changes to the HMI screen:

1. The CD Status will change to "Execution Mode"

2. The "Active File" will be updated to display the current Collision File being executed. These values can also be viewed at MLX[].Robot[].ActiveCollisionFile.

3. A [Stop Collision Detection] button will be visible which can be used to turn off Collision Detection (e.g. in the event that a robot gets stuck in a collision state).

Fig. 5-6: Collision Detection in Execution Mode

Status changes to Execution Mode and Active File is displayed

Now, any time the system detects a collision, the system will abort. *Fig. 5-7* shows the Collision Detection Screen in the case of a collision, and *Fig. 5-8* shows the detailed error description in this case.

Fig. 5-7: Collision Detection Screen in collision state
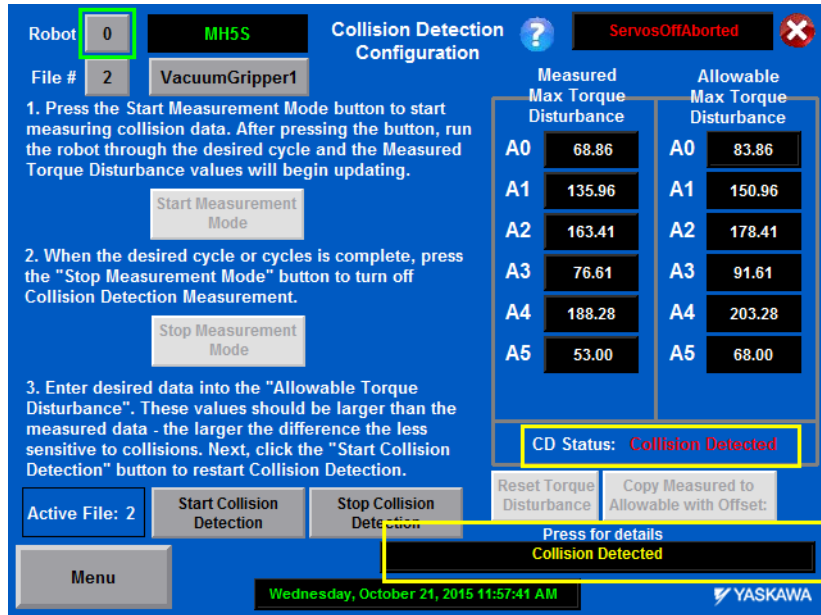
Fig. 5-8: Collision Detection Error Details

168542-1CD
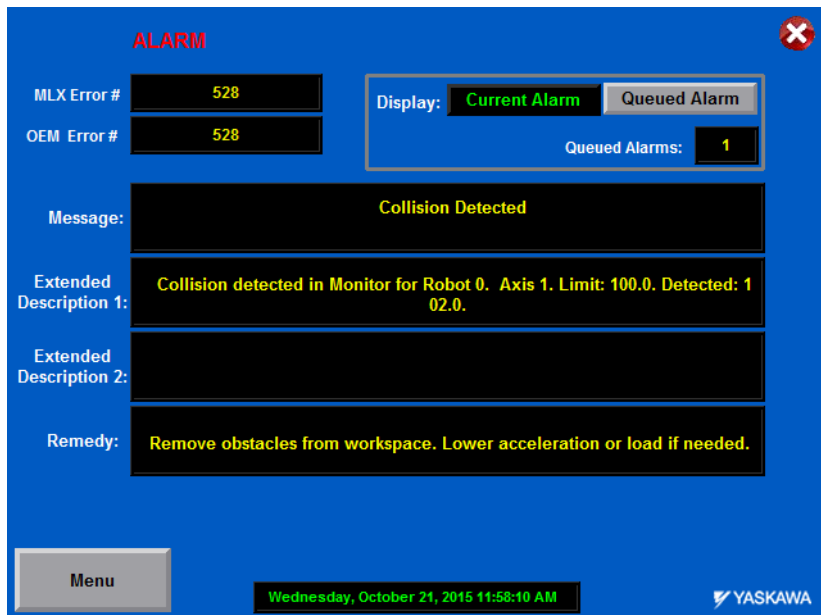
5.3   Using the MLxRobotCollisionDetection Instruction

The previous section described how to configure Collision Detection from the MLX HMI, and this is the recommended method for initially configuring Collision Detection. However, after initial configuration, it is useful to change Collision Detection behavior from the application code itself. This can be useful for properly initializing Collision Detection during application setup or for changing the Collision Detection behavior at different points in the operation of the program (for example, to have stricter behavior at pick/place operations to prevent product damage). This can be done by using the MLxRobotCollisionDetection instruction (*Fig. 5-9*)..

Fig. 5-9: MLxRobotCollisionDetection Instruction
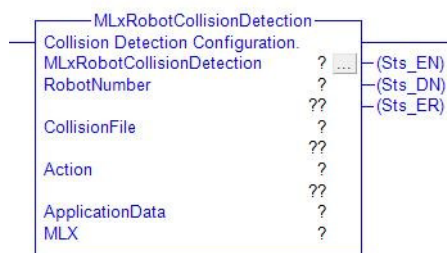
This instruction operates much like other configuration instructions in that the instruction will be complete when the Sts_DN bit is turned on. *Table 5-1* shows the parameters of this instruction and their meanings.

Table 5-1: MLxRobotCollisionDetection Instruction Parameters

Parameter	Description
RobotNumber	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
CollisionFile	The Collision File to use. Valid values are 0 to ApplicationData.NumberOfCollisionFiles-1
Action	Use this to define the action taken by the instruction. Valid actions are: 0 - Start Measurement Mode. 1 - Start Execution Mode. 2 - Stop Collision Monitoring. 3 - Get Maximum Torque Disturbance. 4 - Reset Maximum Torque Disturbance.
ApplicationData	The MLXApplicationData controller scope tag.

5.3.1   Initializing Collision Detection from Application

> ⚠️ **CAUTION**
>
> Collision Detection settings are not retained on the MLX200 Control Module and can be lost if power is cycled or the MLX PLC Interface Application is restarted. Collision Detection should be activated during application initialization.

Because Collision Detection settings are not retained on the MLX200 Control Module, the active settings will be lost if power is cycled or the MLX PLC Interface Application is restarted. The Collision Detection settings should always be activated as part of application initialization. An example of activating Collision File #2 to start in Execution Mode is shown in *Fig. 5-10*.

Fig. 5-10: Example of Activating Collision Detection from Ladder Logic
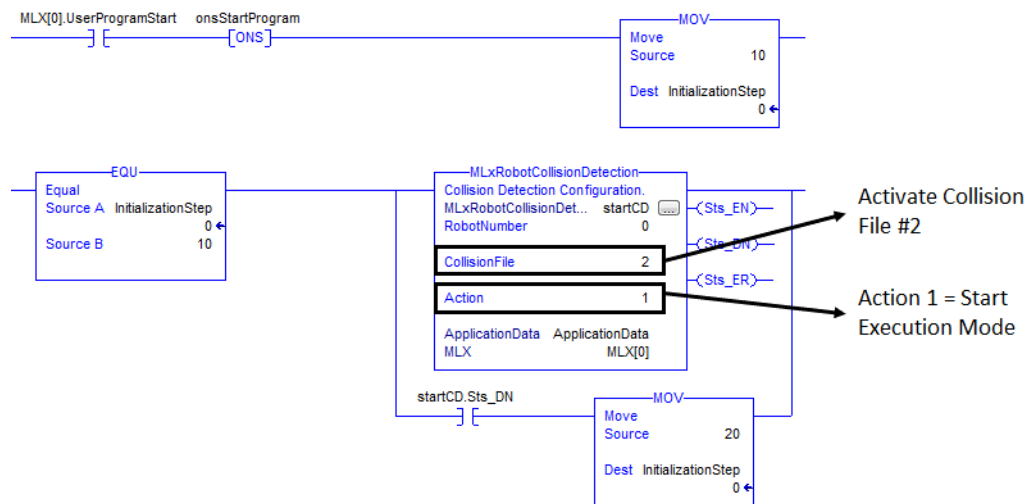
5.3.2 Measuring Collision Detection from Application

⚠️ **CAUTION**

- To get accurate data during Measurement Mode, the tool load must be entered and activated from either the Tool Properties HMI screen or the MLxRobotSetToolProperties instruction.

- Tool Load must not exceed rated values for the robot.

To collect Measured Torque Disturbance Data, the MLxRobotCollisionDetection instruction can be executed with Action = 0. After executing this instruction, the application cycle can be executed and a number of cycles run. When done, call MLxRobotCollisionDetection with Action=2 (Stop Collision Detection) which will also automatically update the Measured Torque Disturbance Data inside the ApplicationData structure. *Fig. 5-11* shows a simple example of this.

Fig. 5-11: Measuring Torque Disturbance Data from Application
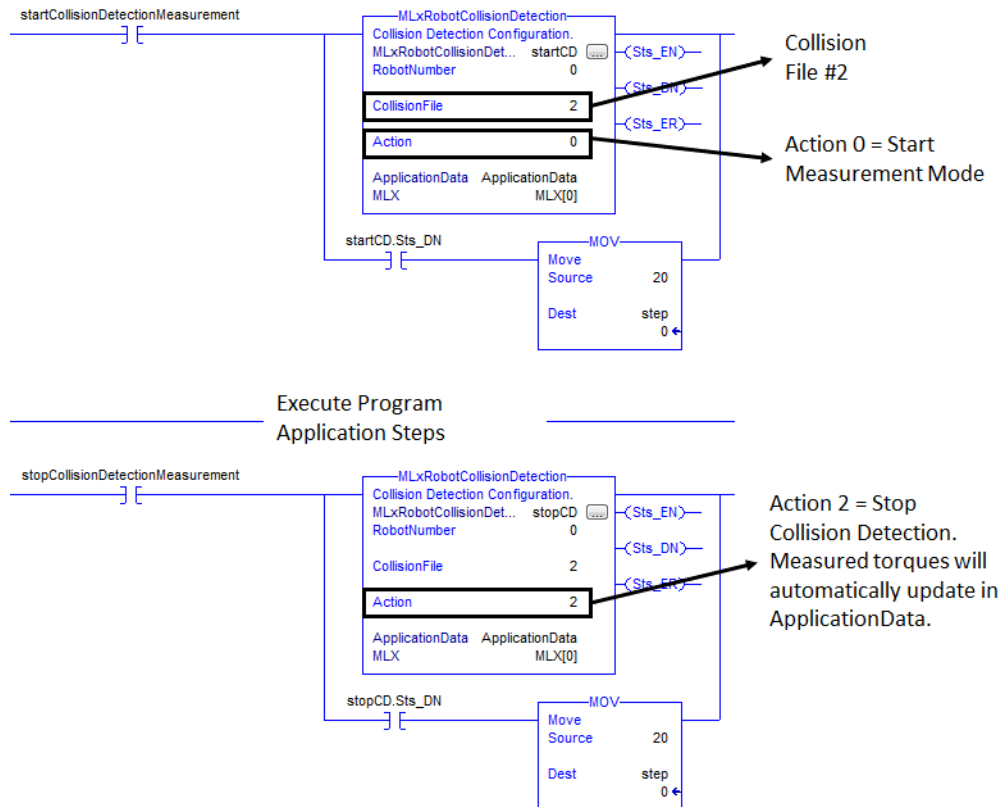
NOTE If the speeds or accelerations of the application are changed after configuring Collision Detection, the configuration may not be reliable. It is recommended to re-configure Collision Detection if changes to motion profiles are made.

5.3.3 Changing Collision Detection Behavior during Application

Another potential use of Collision Detection is to change the behavior based on certain parts of the application cycle. For example, it might be useful to use a different Collision File for Pick/Place operations than for general transfer motions. To do this, simply call the MLxRobotCollisionDetection instruction with the desired Collision File # before these motions, and then set it back to the normal setting afterwards. *Fig. 5-12* shows an example of this. On ApplicationStep=100, the Collision File # is changed to 5, and then the pick/place operation is executed. Afterwards, the Collision File is switched back to 2.

Fig. 5-12: Changing the Collision File # during an Application

168542-1CD

5   Collision Detection
5.3 Using the MLxRobotCollisionDetection Instruction
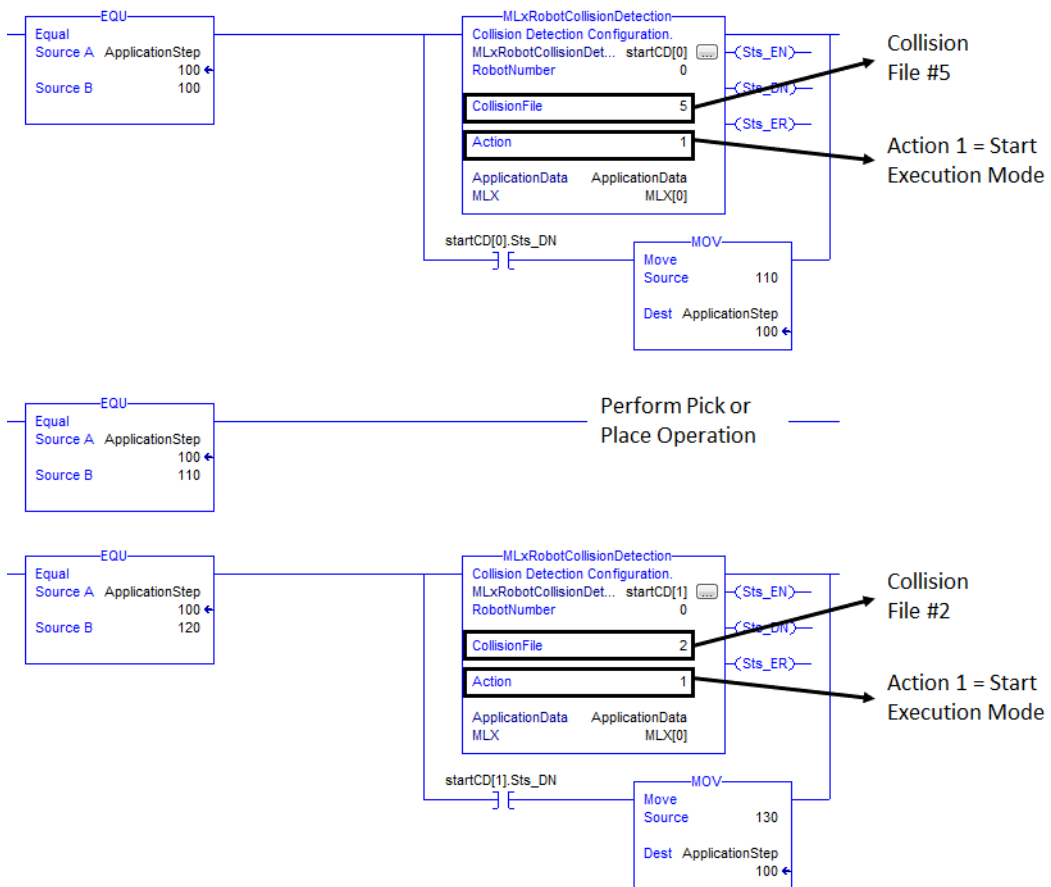
This can be expanded for more complicated application scenarios as well. For example, consider the simple picking sequence shown in *Fig. 5-13*. The robot moves from P1 to an approach point at P2 and then to a pick location at P3. Then, the robot moves back through the approach point at P2 to a waypoint at P1. In this kind of application it may be desirable to activate many different Collision Files throughout the application: one for general point-to-point motions without product, one specifically for picking/placing parts, one for general point-to-point motions with product, etc.

Fig. 5-13: Simple Picking Sequence
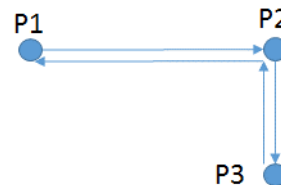
To achieve this, the Collision File can be changed throughout the application cycle as shown as a flowchart in *Fig. 5-14*. In this example, an initial Collision File (#1) is activated at the beginning of the sequence. Then, the Collision File is changed to #2 before the motion down to grasp the object. Then, a new Collision File (#3) is activated after the object has been picked to more accurately match the loading of the system. This application logic could be further expanded to do a similar sequence on the place side as well.

Fig. 5-14: Flow Chart of Changing Collision Files during Application
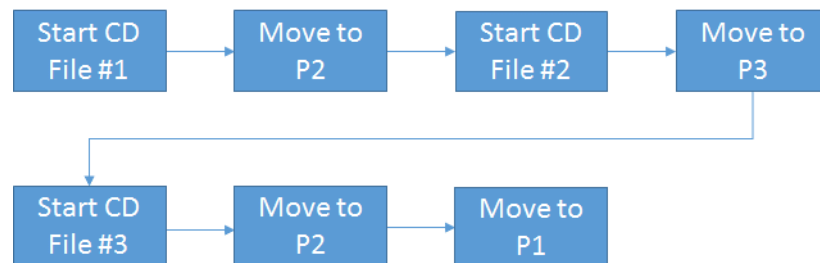
6   Conveyor Tracking

This section describes the configuration and use of the MLX200 Conveyor Tracking feature. The AOIs and other functionality listed in this section are only available if the Conveyor Tracking option has been purchased and is enabled in the MLX200 license file.

6.1   Conveyor Tracking Overview

The conveyor tracking AOIs enable the robot to perform movements relative to a part as it travels down a conveyor. The positions are originally taught with the conveyor stopped. During playback, any motion commands in between the MLxRobotConvSyncStart and MLxRobotConvSyncStop AOIs will be synchronized with the linear motion of the part moving on the conveyor.

Fig. 6-1: Conveyor Tracking Cell Configuration
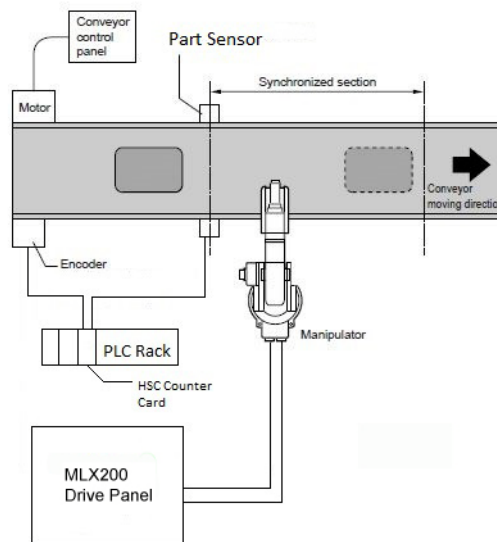
Referring to the example in *Fig.6-2 "Robot Taught Positions with Conveyor Off"*, the conveyor is stopped with the part at a known distance from the photo eye. Seven points are taught near the part. The move commands to points P2-P6 will be placed in the job in between the MLxRobotConvSyncStart and MLxRobotConvSyncStop AOIs. In Automatic mode, when the job is executed, the robot will move to the points taught relative to the part, while it moves down the conveyor. The conveyor motion will be interpolated or added to the taught positions to create a tracking sequence (*Fig.6-3 "Robot Path During Playback"*)

Fig. 6-2: Robot Taught Positions with Conveyor Off
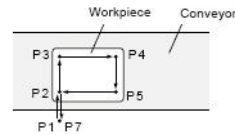
Fig. 6-3: Robot Path During Playback
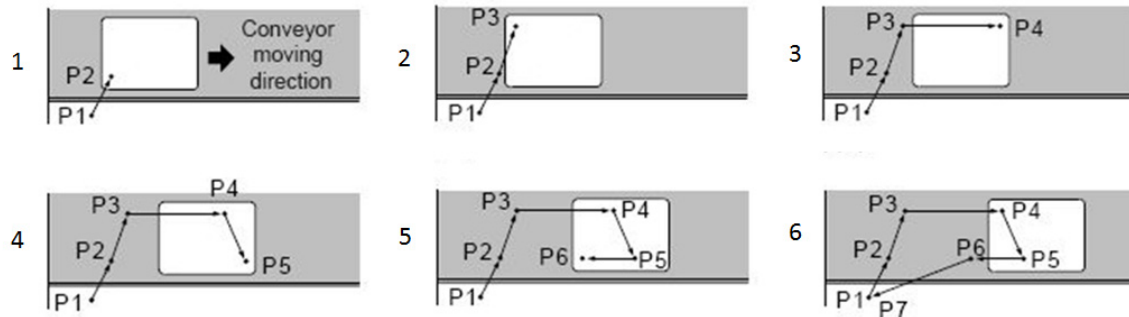
6.2   Conveyor Tracking Requirements

At a high level, a customer requires four things to setup conveyor tracking. These are:

1. A conveyor.

2. An incremental encoder that is mounted to the conveyor for reading conveyor position.

3. A counter card (or something comparable) for ControlLogix/ CompactLogix that is used to read the encoder position.

4. A sensor that generates a trigger output that is used to capture the position of an object on the conveyor. Normally, this output will be wired in to the "latch" input of the counter card.

Conveyor tracking has been tested with MLX200 using the following hardware/software:

1. Conveyor - Any straight line conveyor with speeds up to 150 ft/minute.

2. Counter Card - The 1756-HSC/B Version 3.4 card for ControlLogix and the 1769-HSC/B Version 2.1 card for CompactLogix.

3. Incremental Encoder - Allen Bradley 845H

4. Latch Input - Photo Eye/Switch (e.g. Allen Bradley 42EF-D1JBCK-F4 series A)

5. MLX200 Control Module with Conveyor Tracking feature enabled.

6. RSLogix 5000 development environment Version 20.

MLX200 is delivered with a sample conveyor tracking program. The sample program consists of a basic conveyor tracking program that uses the MLX200 Conveyor tracking AOI's. This sample program can be used as a starting point to building a full conveyor tracking application.

6.3   Configuring Conveyor Tracking

⚠ WARNING
Failure to set up the Conveyor Tags correctly can lead to unexpected behavior during conveyor tracking.

To run Conveyor Tracking, the MLX Conveyor Task will need to be imported and then scheduled inside the MLX_Task. The MLX Conveyor Task should be scheduled to run before the MLX200 Communications Task as shown on the right side of *Fig.6-4 "Importing and Scheduling the Conveyor Task"*.

Fig. 6-4: Importing and Scheduling the Conveyor Task
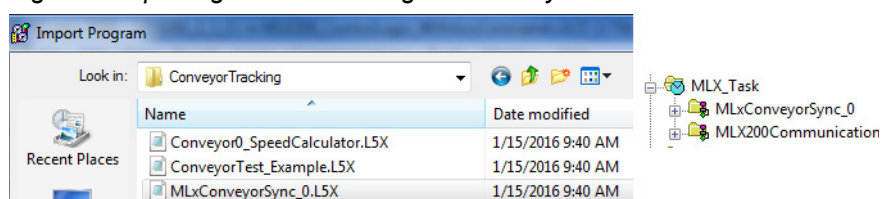
After importing the Conveyor Task, several new tags will show up in the Control Module Scope tags. These tags will have the prefix "Conveyor0" for the first conveyor, "Conveyor1" for the second conveyor, etc… These tags will need to be aligned/linked to the proper HW devices that are being used for reading/latching the conveyor position. *Table 6-1 "Control Module-Scope Conveyor Tag Descriptions"* shows a brief introduction to what each of these tags means. *Section 6.4 "1756-HSC Counter Card Configuration" on page 6-6* will provide an example of setting on a 1756 HSC Counter card for ControlLogix. A similar procedure can be followed for other devices.

Fig. 6-5: Control Module-Scope Conveyor Tags

Table 6-1: Control Module-Scope Conveyor Tag Descriptions

Tag Name	Input/ Output	Description
Conveyor0_CameraState	Input	This bit should turn on when an object is in front of the camera.
Conveyor0_NewData	Input	This bit should turn on when a new object has passed by the camera and stay on until the ResetNewData signal is sent
Conveyor0_ResetCounter	Output	Turning this bit on should reset the counter card position to 0.
Conveyor0_ResetNewData	Output	Turning this bit on will reset the NewData flag (used internally to tell the counter that the current object has been processed and queued).
Conveyor0_CurrentValue	Input	This should contain the current position of the conveyor in encoder counts.

Tag Name	Input/ Output	Description
Conveyor0_LatchedValue	Input	This should contain the current latched position of the conveyor. When a new object passes the camera eye, this position will be stored until the ResetNewData flag is sent.
Conveyor0_RollOver	Input	The RollOver value of the conveyor. This must be set to the proper RollOver value or Conveyor Tracking will not function properly during rollover conditions.
Conveyor0_Speed	Output	This tag can be set to the current conveyor speed for automatically calculating offset based on speed. It can be entered manually, calculated from feedback data, or aliased to a velocity value if using a servo-controlled conveyor. It will be scaled by ApplicationData.ConveyorData[].SpeedConversionFactor.
Conveyor0_CameraAcquire	Output	This tag is only used for MotoPick-MLX option. See separate MotoPick-MLX documentation for usage details.

The following sections will always use the "Conveyor0" prefix to describe the setup, configuration, and operation of Conveyor Tracking. If using a different conveyor number, simply substitute the correct into the prefix.

NOTE
The following sections will always use the "Conveyor0" prefix to describe the setup, configuration, and operation of Conveyor Tracking. If using a different conveyor number, simply substitute the correct into the prefix.

6.4   1756-HSC Counter Card Configuration

This section will describe how to set up the Rockwell Automation High
Speed Counter card (1756-HSC) and the conveyor.

6.4.1   Wiring the 1756-HSC

The HSC card can be configured in several different ways. Each HSC
card supports two independent counters. The setup described in this
document uses only one of the two counters.

Each encoder that is connected to a conveyor needs one counter.
MLX200 is capable of tracking 2 conveyors per system. For a given
encoder, its wires should be tied to the HSC in the following manner.

- HSC card Input A: Tied to conveyor encoder

- HSC card Input B: Tied to conveyor encoder

- HSC card Input Z: Tied to photo switch or any other input that acts as
  a trigger for object detection (could be a vision system)

Refer to the HSC USERS GUIDE SECTION 4 for details on wiring an
HSC card to a conveyor.

The Z input is used to automatically latch the conveyor position in the HSC
card. See the HSC USERS GUIDE PAGE 20 for details on this mode of
operation.

6.4.2   Configuring the 1756-HSC in RSLOGIX

To add an HSC to the project, right-click on the Backplane in the Control
Module Organizer, select "New Module" and navigate to the HSC card,
see *Fig.6-6 "Adding the 1756-HSC Card to RSLogix"*:

Fig. 6-6: Adding the 1756-HSC Card to RSLogix
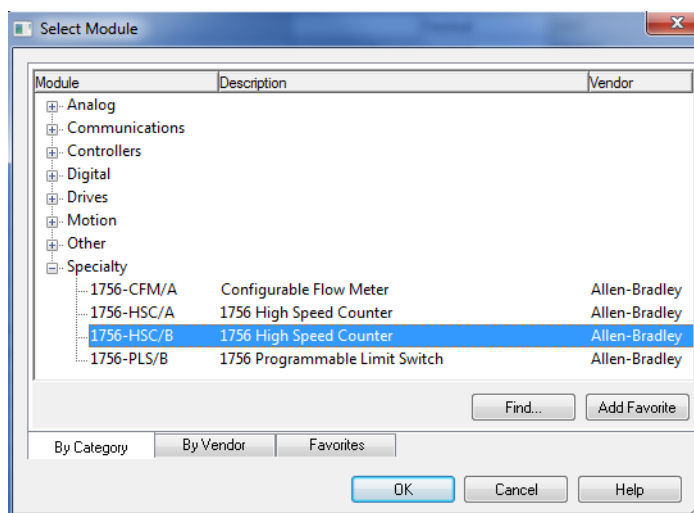
Then, right-click on the HSC and bring up its Properties. On the {General} screen, set the correct slot number for the HSC. On the {Counter Configuration} screen, set the Operational Mode to "Encoder x4 Mode" and the Storage Mode to "Store and Continue Mode" see *Fig.6-7 "1756-HSC Module Properties Configuration"*.

Fig. 6-7: 1756-HSC Module Properties Configuration
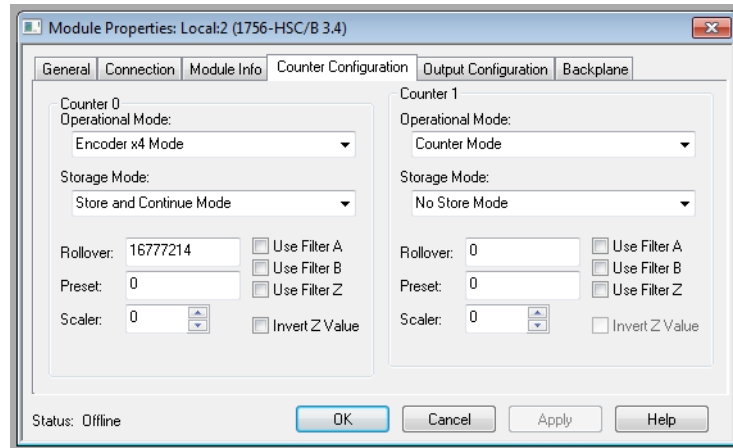
NOTE	• Encode x1 Mode can also be used, but this will have a lower resolution. • For MLX200 Conveyor Tracking to work properly during counter rollover, the Rollover value must be set.

6.4.3   Linking the Conveyor Tags for a 1756-HSC

After configuring the HSC, the Conveyor Tags can be aliased to the proper HSC tags as shown in *Fig.6-8(a) "Conveyor Tags Linked to HSC"* and *Fig.6-8(b) CompactLogix: Conveyor Tags Linked to HSC*.

Fig. 6-8(a): Conveyor Tags Linked to HSC

Fig. 6-8(b): CompactLogix: Conveyor Tags Linked to HSC

6.5   Conveyor Parameter Configuration for MLX200

The information for configuring a specific conveyor for MLX200 is located in the ApplicationData.ConveyorData tag structure.

Fig. 6-9: Conveyor Data Inside Application Data

The ConveyorData tag structure contains the following variables:

Fig. 6-10: Conveyor Application Data

Table 6-2: Conveyor Tag Descriptions

Parameter Name	Description
Conveyor Name	User defined name of conveyor.
IsActive	Turns conveyor reading on for this conveyor. This must be turned on for any active conveyors and should be turned off for unused conveyors to preserve MLX_Task load.
ControlIndex	If using multiple MLX200 Control Modules, this points to the correct module. Default value is 0.
ConveyorPosition	Feedback conveyor position in mm (Note this value is for reading only - modifying this value will have no effect.)
ConveyorStartPosition	Conveyor Position where tracking operations should begin (note this value is stored here only for information purposes - this will still need to be inputted to the conveyor instructions)
ConveyorTeachPosition	Conveyor Position where positions were taught (note this value is stored here only for informational purposes - this will still need to be inputted to the conveyor instructions.)
UserFrameNumber	User frame number attached to conveyor (note: this value is stored here only for informational purposes - this still need to be made active using the MLxRototSetUserFrame instructions)

Parameter Name	Description
ConveyorType	0 = Standard (Photo Eye/Optic Sensor), 1 = MotoPick-MLX (requires MotoPick-MLX optional feature to be enabled)
EncoderToMMConversion	The mm per encoder count. i.e. [EncoderCount] * EncoderToMMConversion = Distance (mm))
SpeedConversionFactor	Set this to 1 if manually specifying the speed value at controller scope tag ConveyorX_Speed or if that value is calculated from position in mm/s. If ConveyorX_Speed tag is aliased to actual speed input from the conveyor, set this to the value that will yield ConveyorSpeed in mm/s. ConveyorSpeed = ConveyorX_Speed * SpeedConversionFactor.
NbrOfPointsToLinearFit	The number of points used to calculate a linear fit of the encoder positions. Useful mainly for conveyor running at close to constant speeds. The valid range for this parameter is 0 - 50.
NbrOfPointsToAverage	The number of points used to calculate a moving average of the encoder positions. Useful for noisy encoder signal. The valid range for this parameter is 0 - 50.
LagOffset	An offset in mm along the x direction of the conveyor. This can be used to offset a constant error along the conveyor. This value can be used to fine tune the execution positions of motion instructions that execute after a call to MLxRobotConvSyncStart. A positive value corresponds to an adjustment in the +X direction.
NbrOfPartsInPattern	This value is used for Pattern-Based Distribution in the case of running multiple robots on the conveyor. See *Section 6.7.2.3 "Advanced Application Options"*.
PartPattern	This value is used for Pattern-Based Distribution in the case of running multiple robots on the conveyor. See *Section 6.7.2.3 "Advanced Application Options"*.
PatternPosition	This value is used for Pattern-Based Distribution in the case of running multiple robots on the conveyor. See *Section 6.7.2.3 "Advanced Application Options"*.

6.6   Conveyor Tracking Setup Procedure

6.6.1   Verify Counter Card is Functional

Before attempting Conveyor Tracking operations, the functionality of the counter card (or comparable device) should be verified. To do this, first temporarily unschedule the MLX_ConveyorSync_0 program from the MLX_Task so that the MLX200 is not interacting with the conveyor setup. Verify the following steps:

> ⚠️ **WARNING**
>
> If the behavior of the Conveyor Tracking tags does not match what is described below, Conveyor Tracking operations will not work correctly and could cause unexpected robot behavior.

1. Turn conveyor on and start moving it. The Conveyor0_CurrentValue should start increasing.

2. Place an object on the conveyor and have it move past the photo eye. As it crosses, the Conveyor0_CurrentValue should be stored in the Conveyor0_LatchedValue tag, and the Conveyor0_NewData bit should turn to 1.

3. Toggle the Conveyor0_ResetNewData bit. The Conveyor0_NewData bit should turn to 0.

If the above steps work correctly, the hardware is configured properly. If not, it is important to debug the counter card before moving on to the next step. Re-schedule the MLX_ConveyorSync_0 program under the MLX_Task.   Move it to the top of all scheduled programs.

6.6.2   Calculate Conveyor Resolution

With the conveyor stopped, set a part on a conveyor. Place a ruler next to the conveyor. Record the Conveyor0_CurrentValue for the current position. Operate the conveyor to move the part approximately 1000 mm. Measure the distance that the part traveled (mm). Record the new Conveyor0_CurrentValue, and subtract the previous value to determine the change in encoder pulse counts. Calculate the pulse count conversion factor (distance traveled / pulse counts), and enter the value in ApplicationData.ConveyorData[].EncoderToMMConversion tag.

To verify the tag is correct, operate the conveyor again and verify that the changing ApplicationData.ConveyorData [].ConveyorPosition variable is reporting the distance traveled in mm.

6.6.3   Teach a User Frame

Install a pointing device on the robot and enter the tool data. Teach a user frame for the conveyor. The direction of travel for the conveyor should be the +X direction of the conveyor frame. Record this frame number in the ApplicationData.UserFrame[].UserFrameNumber tag and add a call to MLxRobotSetUserFrame in your application logic.

6.6.4   Teach Point and Setup Tracking Parameters

■   **Conveyor Teach Position Value**

> ⚠️ **WARNING**
>
> The motions performed while conveyor tracking will be different from those taught with the conveyor stationary. Thus, certain errors may be encountered while tracking that would not show up otherwise (e.g. speed/limit violations).

Operate the conveyor to move a part past the photo eye and stop the conveyor at a position where the part will be taught. Determine how far the part traveled past the photo eye. This can be obtained by either measuring the distance from the sensor to the front edge of the part, or by referring to the ApplicationData.ConveyorData[].ConveyorPosition tag. Note that every time a part crosses the photo eye, this tag should reset to zero.

Distance = ApplicationData.ConveyorData [].ConveyorPosition

Record this value inside the ApplicationData.ConveyorData[] tag structure or enter it using the {Conveyor Tracking HMI} screen (*Fig.6-11 "Conveyor Tracking Setup Screen"*).

Teach all the robot positions relative to the part with the conveyor stopped. Typical points taught will be an approach position above the part, a grip position at the part, and a depart position above the part. During playback, Linear Motion instructions can be used to move the robot TCP to these points while tracking the part motion on the conveyor, as long as they fall between a MLxRobotConvSyncStart and a MLxRobotConvSyncStop.

Enter the remaining parameters on the {HMI Conveyor Tracking Setup} screen:

Fig. 6-11: Conveyor Tracking Setup Screen
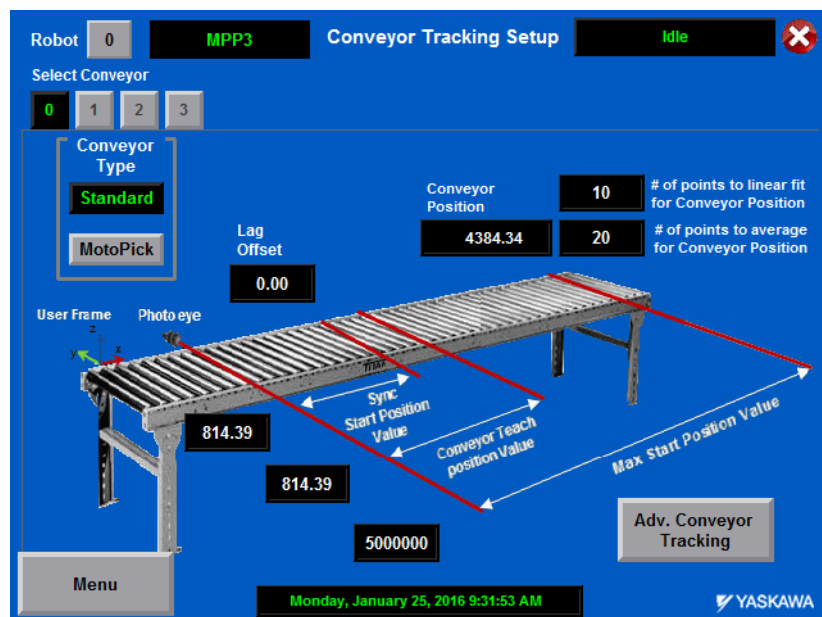
■ **Conveyor Type**
This defines whether the conveyor is a standard conveyor operating using a Photo Eye/Optical Sensor or a MotoPick-MLX conveyor using a camera to parse data. The instructions in this chapter assume "Standard" conveyor, and this should be selected.

■ **Sync Start Position Value**
Determine how far past the photo eye the conveyor tracking sequence should begin. The robot will wait at the MLxRobotConvSyncStart instruction until the part has traveled to this distance past the sensor. At this point, the robot will begin to move.

■ **Max Start Position Value**
Estimate the worst case distance past the photo eye that the robot will be able to begin tracking a part and successfully pick it up without reaching a software limit. This parameter is used for queued parts. After moving a part, the robot will return to the pounce position. If a new part had already crossed the photo eye and is in a queue, the MLxRobotConvSyncStart AOI will determine if its location is past the Max Start Position Value, which will indicate the pickup cannot be reached within the work space. Whenever the speed of the conveyor is changed, this value should be adjusted. Note that in the data structure the Max Start Position is an array, so that a different value can be inputted for each robot.

■ **# of Points to Average/Linear Fit for Conveyor Position**
These parameters will allow you the smooth out feedback from the conveyor motion. Up to 50 encoder position readings can be averaged or linearly fit.

■ **Conveyor Position**
This field is for conveyor position reference only and cannot be changed.

■ **Lag Offset**
The lag offset parameter can be adjusted to speed up or slow down the SyncStart instruction execution time, so that parts can be picked up at a consistent location on the part. It can compensate for I/O hardware delays.

6.6.5 Debugging Pickup Position Errors

> **NOTE** This section should be utilized after the ladder program is developed.

6.6.5.1 Part is Gripped at Different Locations on Part

If the part is gripped at different locations on the part, the conveyor resolution is not as accurate as it needs to be. This is typically observed when the part is picked from different locations on the conveyor.

To improve this resolution, slow the conveyor down, and add a long delay in the program after the move position above the part. During the playback, the robot will be tracking the conveyor with the gripper above the part. If the conveyor resolution tag is accurate, the robot should maintain its position relative to the moving part, while it is executing the

168542-1CD

long delay. If it is observed that the robot gripper is drifting forward or backward in respect to the part, modify the EncoderToMMConversion tag until the drift is eliminated.

6.6.5.2   Part is Gripped Consistently at the Wrong Location on the Part

■   **Part is Gripped Consistently at the Wrong Location on the Part**
If the part is picked up anywhere on the conveyor with the gripper consistently at the wrong location on the part, there is a small execution delay that needs to be calibrated. Typically, this is caused by the response time of hardware I/O. In most cases, it is desired to program the gripper to pick up the part at a center location on the part. On the {HMI Conveyor Tracking Setup} screen, the Lag Offset can be modified to fine tune the pickup of position of the gripper on the part.

Whenever the conveyor speed is changed, the Lag Offset should be modified. The corresponding tag in the ladder is ApplicationData.ConveyorData[x].LagOffset.

6.7 Developing a Conveyor Tracking Application

This section will describe how to develop a conveyor tracking application using MLX200. The first section will introduce the instructions specific to Conveyor Tracking. Then, several examples of how to structure a Conveyor Tracking application are shown. Finally, several potential pitfalls that can be encountered while using Conveyor Tracking are discussed.

6.7.1 Conveyor Tracking Instructions

MLX200 has 4 instructions related to Conveyor Tracking. Two basic instructions are provided to turn conveyor tracking on/off:

■ **MLxRobotConvSyncStart**
Starts conveyor tracking

■ **MLxRobotConvSyncStop**
Stops conveyor tracking

There are also two instructions for blending motion within the sync commands. These commands are available to assist with increasing pick rates.

■ **MLxRobotConvSyncStopWithLinearMot**
While turning off conveyor tracking operations, blend motion into a linear path to the next point.

■ **MLxRobotConvSyncStopWithAxisMot**
While turning off conveyor tracking operations, blend motion into a joint path to the next point.

6.7.1.1 MLxRobotConvSyncStart Instruction

■ **MLxRobotConvSyncStart**
AOI is used to begin conveyor tracking operations. When called, this AOI will wait for an object to be added to the queue (this happens automatically when an object passes the photo eye) and then turn on the Sts_DN bit.

Fig. 6-12: Conveyor Synchronization Start Instruction
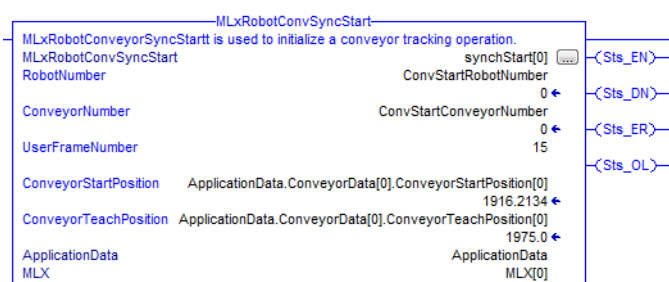
■ **ConveyorStartPosition**
This is the position along the conveyor that the robot will start to track queued objects. If you are familiar with Motomans INFORM language then ConveyorStartPos variable is analogous to the STP parameter. Range of valid values is 0-MaxStartPosition.

■ **ConveyorTeachPos**
The ConveyorTeachPos (analogous to the CTP parameter in INFORM) is the position along the conveyor where the robot programmed points were taught with the conveyor stopped.

■ **UserFrameNumber**
The User Frame Number that defines the X-direction of the conveyor.

■ **.Sts_DN**
This output will be asserted once a part has crossed the ConveyorStartPos.

■ **.Sts_OL**
This output is asserted when a queued object has gone beyond the MaxStartPosition value.

Fig. 6-13: MLxRobotConvSyncStart Instruction Parameters
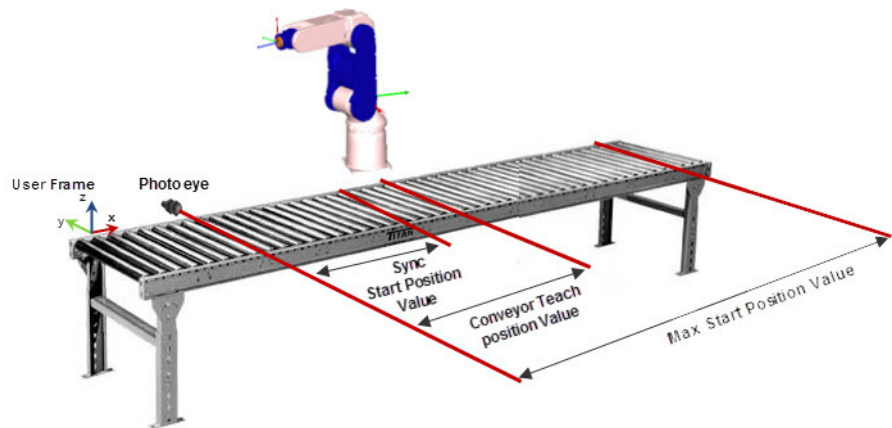
6.7.1.2   MLxRobotConvSyncStop Instruction

MLxRobotConvSyncStop is used to turn off conveyor synchronization and updates the internal object queue. Currently, the Conveyor0_CurrentValue for the correct conveyor must be passed into the AOI. In addition, there are two parameters (KeepInQueue and NewRobotQueue) which can be used to handle how the object queue is updated. If KeepInQueue is set to 0, the object is flushed from the queue and no longer tracked. In this case, the value of NewRobotQueue does not matter. If KeepInQueue is set to 1, the value of NewRobotQueue will move the object to that robot's queue. If NewRobotQueue is equal to the current robot, the object will be kept in the current robot's queue and will be the first object in the queue when another ConvSyncStart is called. This is useful if an application requires stopping conveyor tracking operations temporarily while still tracking the same object.

Fig. 6-14: Conveyor Synchronization Stop Instruction

168542-1CD

MLX200 Software and Operations	6 Conveyor Tracking
	6.7 Developing a Conveyor Tracking Application
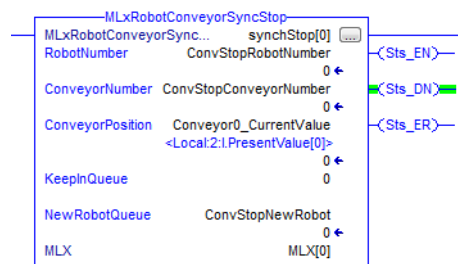
6.7.1.3 MLxRobotConvSyncStopWithLinearMot Instruction

MLxRobotConvSyncStopWithLinearMot is basically a combination of a Linear Motion and a Conveyor Sync Stop. This instruction will turn off conveyor tracking operations and update the object queue like a normal Conveyor Stop; however, instead of stopping the robot motion, it will instead blend directly into a linear motion at another target. This can help decrease cycle times in demanding applications. The rest of the parameters for the Linear Motion are identical to a normal motion instruction. The Blend Factor in this case will be used to blend into any motions that are added to the back of the command.

Fig. 6-15: Conveyor Synchronization Stop Followed by a Linear Move Instruction
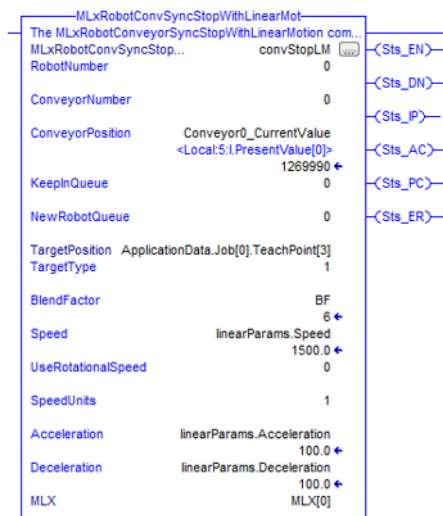
6.7.1.4 MLxRobotConvSyncStopWithAxisMot Instruction

MLxRobotConvSyncStopWithAxisMot is basically a combination of a Linear Motion with a Conveyor Sync Stop. This instruction will turn off conveyor tracking operations and update the object queue like a normal Conveyor Stop; however, instead of stopping the robot motion, it will instead blend directly into a PTP motion at another target. This can help decrease cycle times in demanding applications. The rest of the parameters for the Axis Motion are identical to a normal motion instruction. The Blend Factor in this case will be used to blend into any motions that are added to the back of the command.

Fig. 6-16: Conveyor Synchronization Stop

168542-1CD

6.7.2 Programming Structure for a Conveyor Tracking Application in Ladder

The following sections will describe the basic methodology to program a conveyor tracking application.

6.7.2.1 Program Structure Overview

Fig.6-17 "Basic Program Structure for a Conveyor Tracking Application" shows the basic structure of a simple Conveyor Tracking program. On Step 10, an Axis Motion (e.g. MLxRobotMoveAxisAbsolute) is used to move the robot to its "pounce" position. This is the position from which the robot will wait to start tracking objects. After this motion is complete, Step 20 will wait at an MLxRobotConvSyncStart command until a part crosses the photo eye and moves past the defined ConveyorStartPosition. After this happens, the program will move to Step 30 where the linear motions to the taught points relative to the part are executed while tracking the moving part. When these motions are complete, Step 40 will call an MlxRobotConvSyncStop command which will stop the tracking action and also remove the part from the queue.

> NOTE
> - The points taught for the linear motion during conveyor tracking, need to be taught in world coordinate, not in the conveyor user frame.
> - Only linear moves are supported when conveyor tracking is on. Any other type of motion will result in an error.

Fig. 6-17: Basic Program Structure for a Conveyor Tracking Application
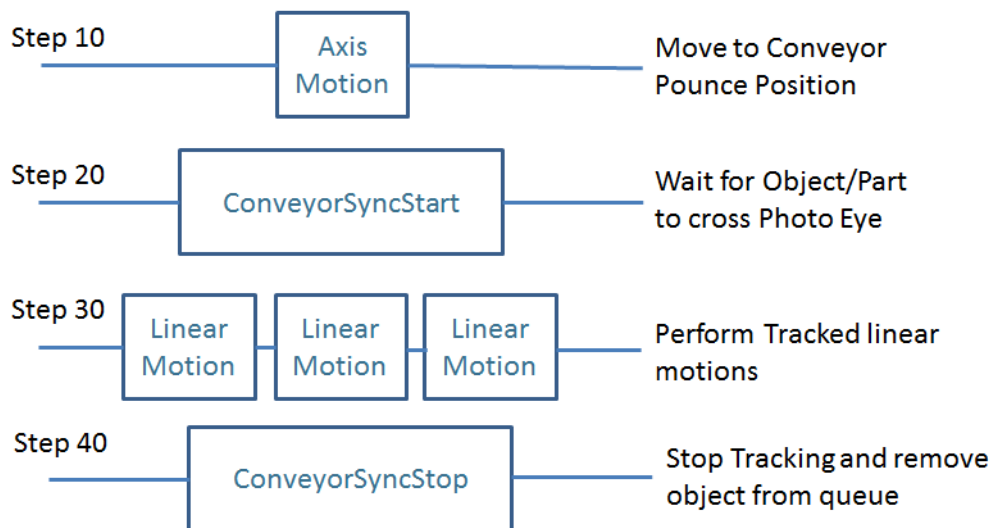
The ConvSyncStop on Step 40 could also be replaced with a ConvSyncStopWithAxisMotor or ConvSyncStopWithLinearMot to allow smooth blending out of the conveyor tracking operations.

168542-1CD

6.7.2.2   Program Structure Details

⚠ **CAUTION**

Output Latch (OTL) and Output Unlatch (OTU) should be used to toggle the conveyor output signals. Do not use Output Energize (OTE) as this could interfere with internal tasks.

This section will expand on the information from the previous section and show how the conveyor tracking instructions will look like in actual application code. The first step of the application is to activate the User Frame that is attached to the conveyor and to reset the conveyor state (*Fig.6-18 "Conveyor Initialization Step"*). The following actions are performed to reset the conveyor:

- Moving 0 into the robot's QueueControl.POS variable will act to flush the queue. Thus, any parts remaining in the queue will be removed.

- The Conveyor0_ResetNewDataFlag variable lets the MLX200 know that the last value that was added to Conveyor0_LatchedValue has been processed. In the case that a system is aborted between a part being detected and processed, adding additional points may not work correctly if this is not reset.

- The Conveyor0_ResetCounter variable will simply reset the counter's current encoder count to 0. This is not technically required but will ensure the application starts from the same configuration every time.

Fig. 6-18: Conveyor Initialization Step
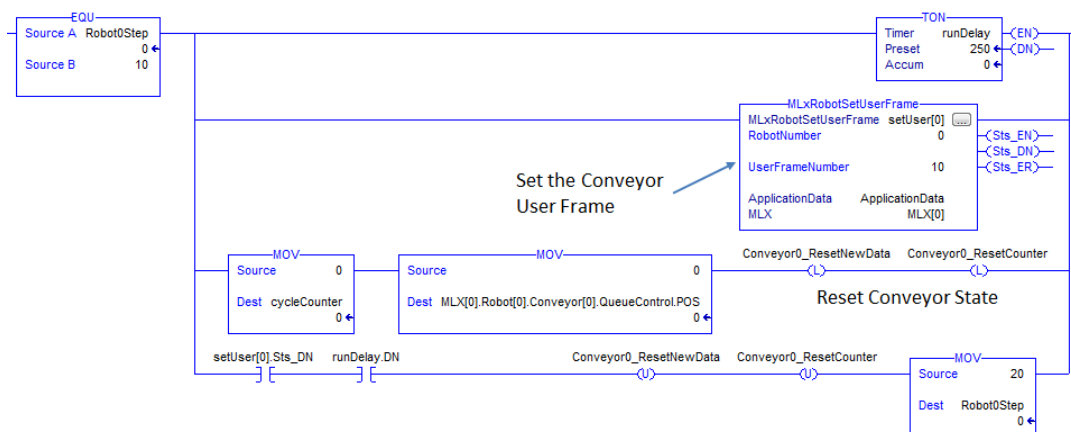
After the initialization is complete, the program step is incremented to 20. At step 20, an Axis Motion to the conveyor pounce position is executed. Note that this step is incremented to 30 when the Sts_DN bit (instead of Sts_PC bit) of the motion instruction turns on. This means that the program will move to Step 30 (Conveyor Sync Start) as soon as the motion is queued rather than waiting for the motion to complete. Thus, if a part crosses the photo eye while this motion is taking place, it can blend into the tracked motions. For more information on using the Sts_DN to trigger application logic, review *Section 4.4.2 "DN BIT Checking" on page 4-20*.

168542-1CD

After the motion to the pounce position is called, the step will increment to Step 30 where the Conveyor Sync Start instruction is executed (*Fig.6-19 "Move to Conveyor Pounce Position"*). As mentioned earlier, this instruction will wait for an object to pass the photo eye and move to the Conveyor Start Position before turning on its Sts_DN bit. This instruction also has a Sts_OL bit which will turn on only if the object has moved past its Max Start Position. If the Sts_OL bit is high, the application will increment straight to the Conveyor Sync Stop rung (Step 50) - otherwise, it will move to the Tracked Motions (Step 40) (*Fig. 6-20*).

Fig. 6-19: Move to Conveyor Pounce Position
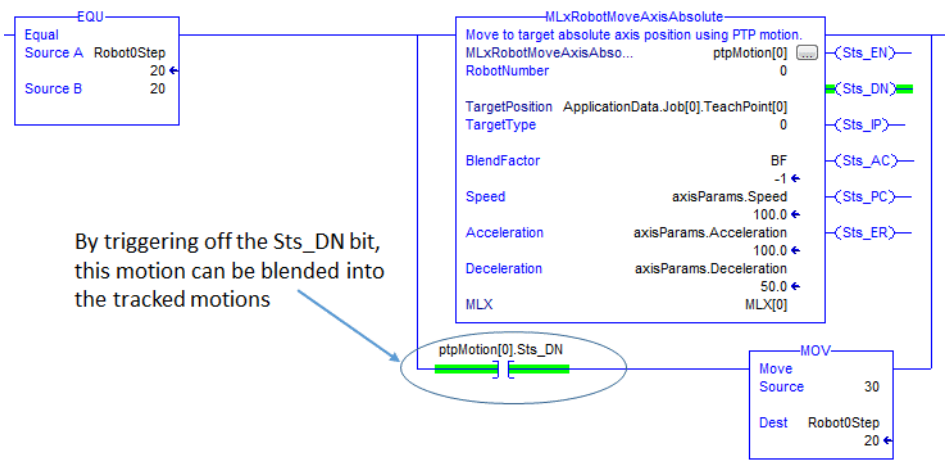
Fig. 6-20: Conveyor Sync Start Step
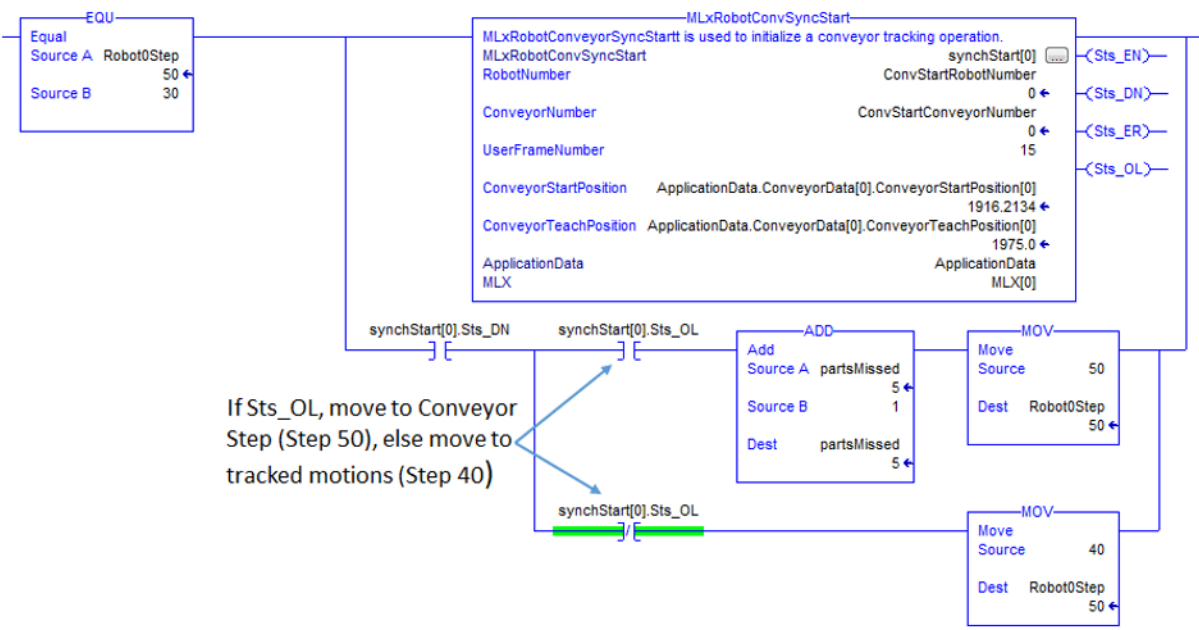
Fig.6-21 "Execute Tracked Motions Step" shows the rung that executes the actual tracked motions. Here three simple linear motions are being executed.

Fig. 6-21: Execute Tracked Motions Step
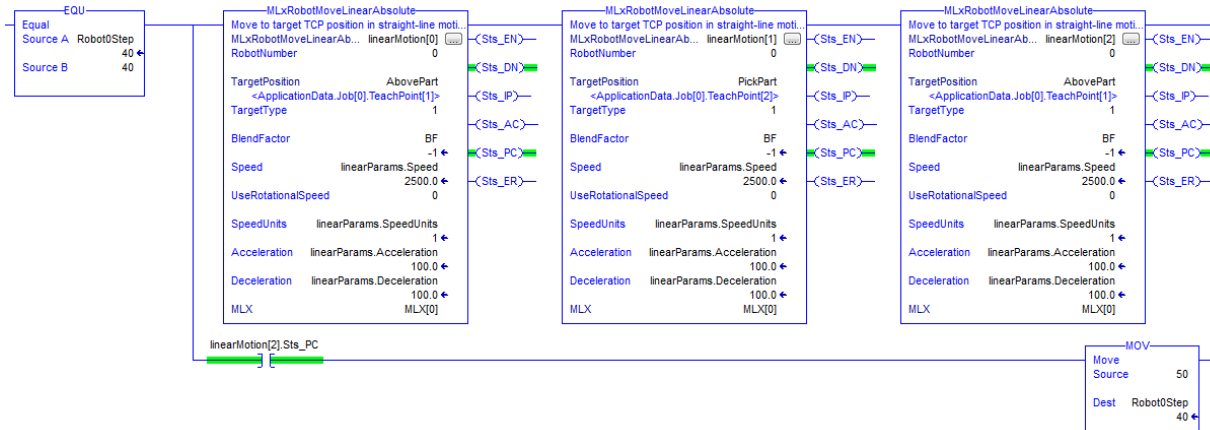
Finally, *Fig.6-22 "Conveyor Sync Stop Step"* shows the Conveyor Sync Stop step. The Conveyor Sync Stop instruction will turn off conveyor tracking operations and update the internal object queue (based on the values of KeepInQueue and NewRobotQueue as detailed in *Section 6.7.1.2 "MLxRobotConvSyncStop Instruction"*). After the Sts_DN bit is set, this application also checks if there are any additional objects in the queue. If the queue is not empty, the application returns immediately to the Conveyor Sync Start step (Step 30). Otherwise, it returns the Step 20 where the motion to the pounce position is executed.

Fig. 6-22: Conveyor Sync Stop Step

168542-1CD
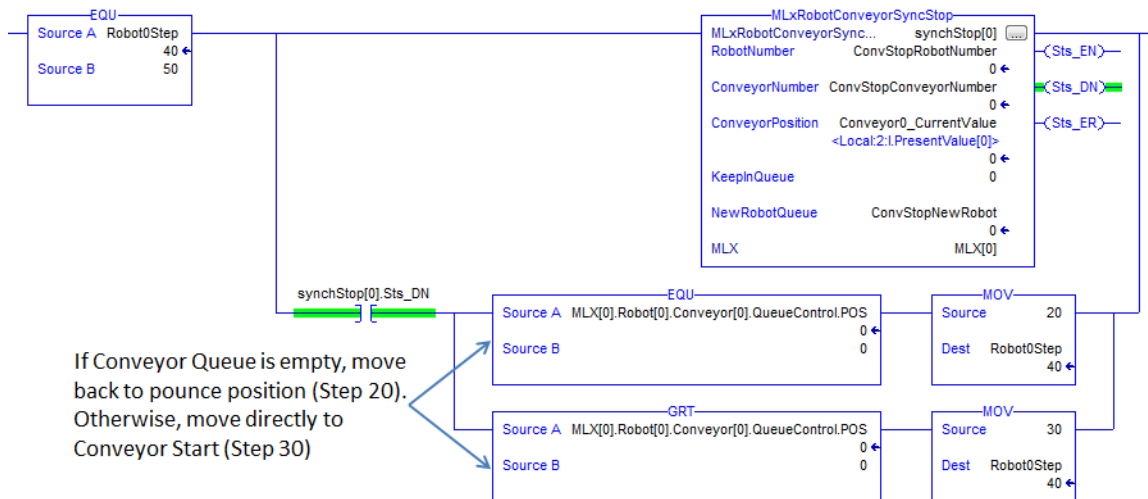
6.7.2.3   Advanced Application Options

MLX200 also contains several advanced application options that are useful when using multiple conveyors or multiple robots on one conveyor:

- Pattern-Based Distribution - the ability to route parts on a conveyor to multiple robots based on a set pattern.

- Dynamic Load-Balancing - the ability to move objects between different robot queues.

The following sections will discuss these concepts and briefly describe some of the more advance application scenarios that they support.

■   **Pattern-based Distribution**
In the case of multiple robots working on the same conveyor, MLX200 allows parts to be routed to each robot based on a pattern. For example, parts could just simply be routed every other part to two robots (i.e. part 0 to Robot 0, part 1 to Robot 1, part 2 to Robot 0, part 3 to Robot 1, etc…). This pattern can be set up directly in the tags shown in *Table 6-3 "Pattern-Based Distribution"* or through the {HMI} screen shown in *Fig.6-23 "Pattern-based Distribution HMI Screen"*.

Table 6-3: Pattern-Based Distribution

Pattern	Description
NbrOfPartsInPattern	The number of parts in the repeating pattern (up to 16)
PartPattern	An array that defines which robot each part in the pattern will be routed to (only the first *NbrOfPartsInPattern* elements are used.)
PatternPosition	Defines the current position in a pattern. This value can also be set during application initialization to define where to start the pattern (e.g. if recovering from an error condition).

Fig. 6-23: Pattern-based Distribution HMI Screen
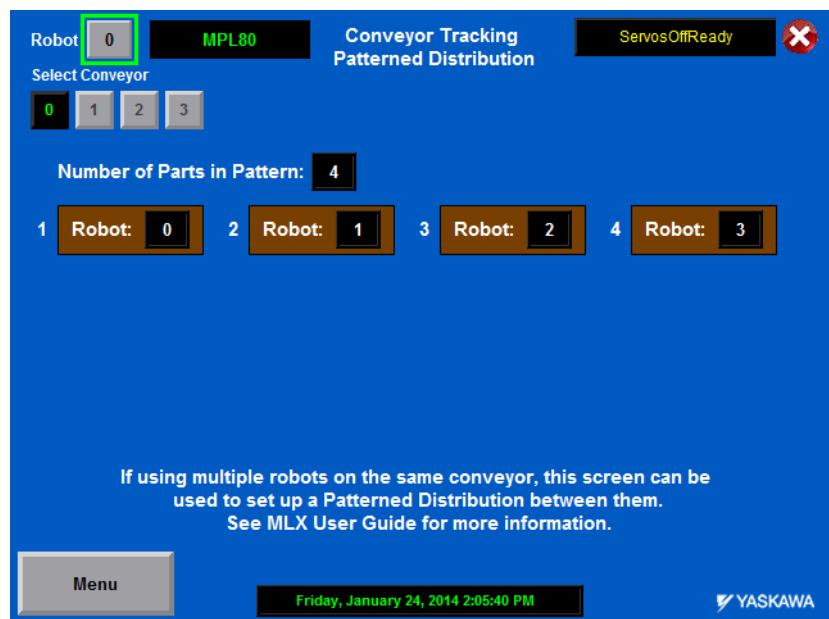
For example, consider the pattern defined in *Fig.6-24 Conveyor Pattern Configuration*. This pattern has two parts that are cycled between two robots, and this pattern will be repeated as more parts come. Since every other part goes to each robot, the routing of parts coming down a conveyor would like *Fig.6-25 "Pattern-based Distribution Example"*.

Fig. 6-24: Conveyor Pattern Configuration

Fig. 6-25: Pattern-based Distribution Example
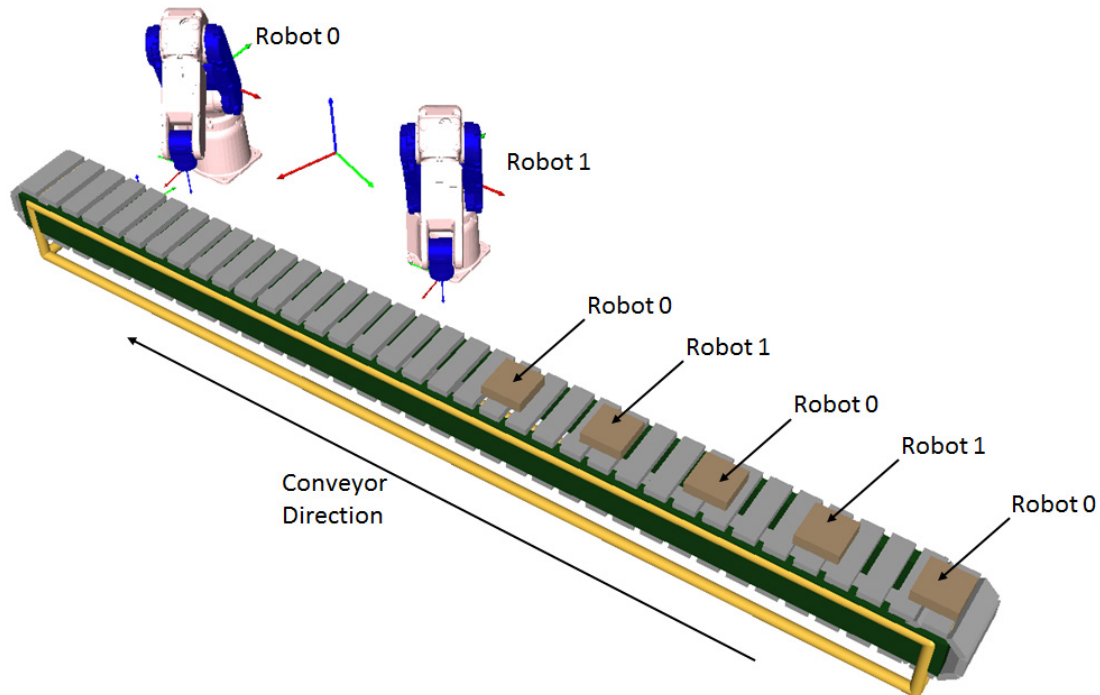
■ **Dynamic Load-balancing**

Dynamic Load-Balancing refers to the process of using the KeepInQueue
and NewRobotQueue variables in the Conveyor Sync Stop instruction to
dynamically handle the object queue. The use of these variables was first
described in *Section 6.7.1.2 "MLxRobotConvSyncStop Instruction"*. This
section will show how these variables can be used to implement several
application scenarios.

First, consider the example shown in *Fig.6-26 "Two Robots on One
Conveyor Example"* with two robots on one conveyor. In the previous
section, we introduced how the parts could be routed between the two
robots using Pattern-Based Distribution. Another way to handle this
application would be to route all parts to Robot 0, and then have Robot 1
pick any parts that the first robot missed. To do this, the application logic
could check the Sts_OL bit from Robot 0's Conveyor Sync Start
instruction. If this was set to 1, the Conveyor Sync Stop command could
be called with KeepInQueue=1 and NewRobotQueue=1. This would move
the part to Robot 1's queue where it could be picked further down the line.

Fig. 6-26: Two Robots on One Conveyor Example
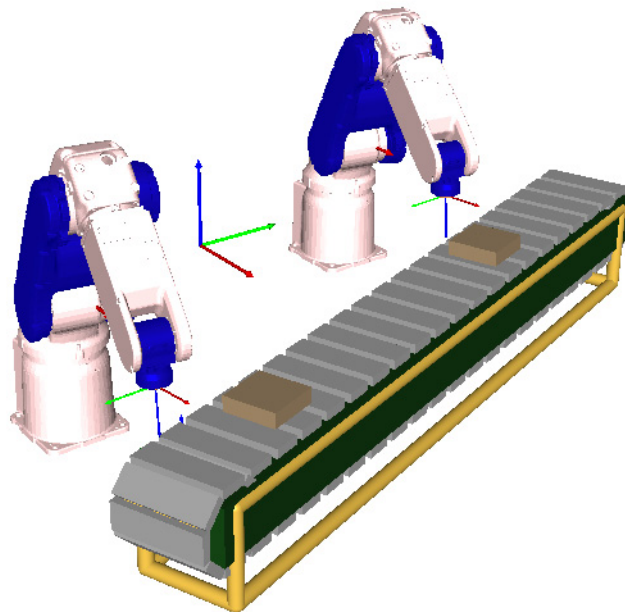
> **NOTE**
> It is also possible to combine Pattern-based Distribution and
> Dynamic Load-Balancing. For example, in the above
> scenario, a pattern of 2 parts to Robot 0 and then 1 part to
> Robot 1 could be established, with Robot 0 still moving the
> parts it missed to Robot 1.

Another example scenario involves a robot is picking parts off of one conveyor and placing them onto a tracked part on another conveyor. For example, consider the application shown in *Fig.6-27 "One Robot with Two Conveyors Example"*. In this example, parts are being tracked and picked on Conveyor 0 and placed onto a tracked tray on Conveyor 1. After four parts have been placed on the tray, the robot will remove that tray from its queue and start placing parts on the next tray. To do this, the Conveyor Sync Stop command on Conveyor 1 would be called with KeepInQueue=1 and NewRobotQueue=0. This tells the system to stop tracking that conveyor but keep the part in the queue for the next time. After the tray is full, a Conveyor Sync Stop with KeepinQueue=0 could be called to flush the object.

Fig. 6-27: One Robot with Two Conveyors Example

Parts are placed on tray on Conveyor 1

Parts are picked off of Conveyor 0
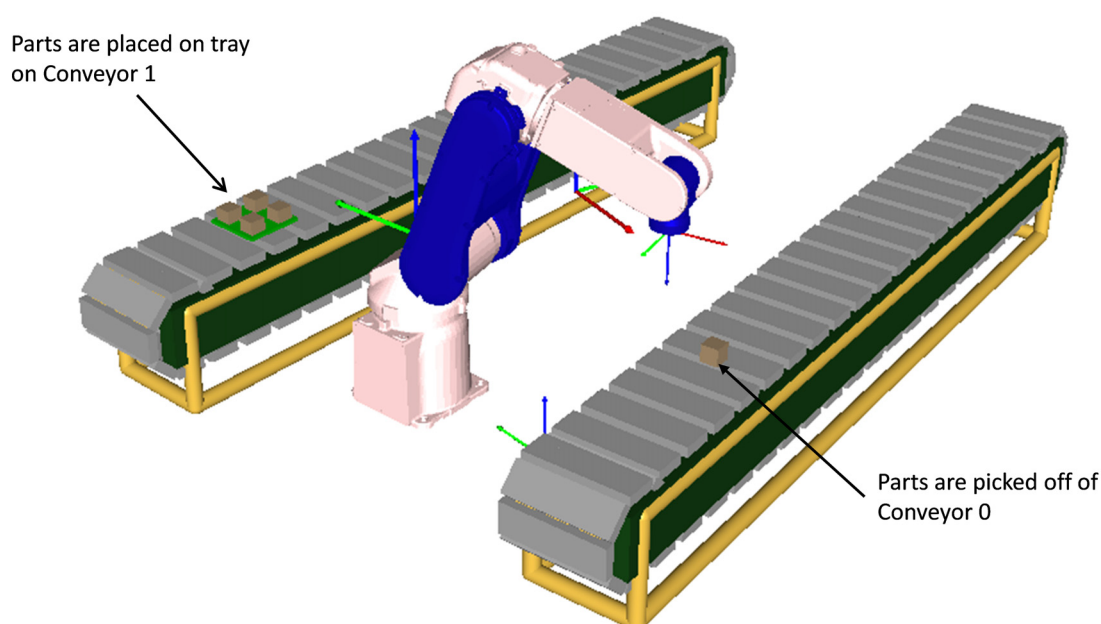
6.7.2.4   Conveyor Tracking Programming Pitfalls

The following sections describe some basic pitfalls that can come from programming Conveyor Tracking applications.

■ **DELAY AFTER CHECKING MAXSTARTPOSITION**
The start position of an object is recorded and checked against the MaxStartPosition when the MLxRobotConvSyncStart command is executed. If there is a delay between this call and the first motion (as in *Fig.6-28 "Basic Conveyor Program Structure"*), a part may move beyond the MaxStartPosition without an error occurring. For this case, the part might not be reachable and an axis limit might be exceeded.

Fig. 6-28: Basic Conveyor Program Structure
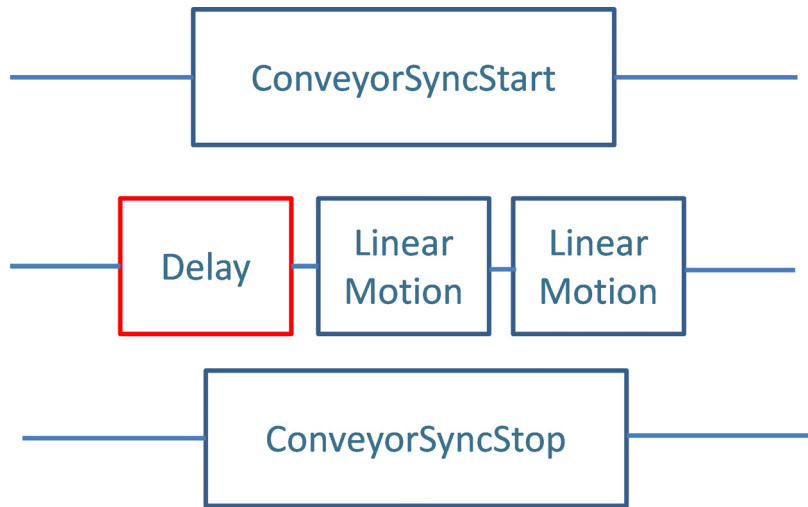
- **STS_OL USAGE**
  The MLxRobotConvSyncStart command has a status bit called Sts_OL
  that turns on if the part is past its MaxStartPosition when the instruction is
  called. It is then up to the application logic how to handle this scenario
  (e.g. continue to try to track the part anyway, or abort the system, or skip
  the part, etc). The most common scenario will be to skip the part and
  record that a part was missed. In this case, a Conveyor Sync Stop
  command should be executed to flush the part from the queue. An
  example of this ladder code is shown in *Fig.6-29 "Example Use of Sts_OL
  Bit"* Example Use of Sts_OL bit.

  Note that an MLxRobotConvSyncStopWithAxisMot or WithLinearMotion
  will always attach a motion after the stop. However, in the case of a
  Sts_OL bit, the application should only update the queue/turn off conveyor
  synchronization. Thus, in this case, a normal MLxRobotConvSyncStop
  should be called to prevent an unwanted motion from being added to the
  queue. Thus, it may be useful to have both a regular SyncStop and a
  SyncStopWithMotion in the application.

Fig. 6-29: Example Use of Sts_OL Bit
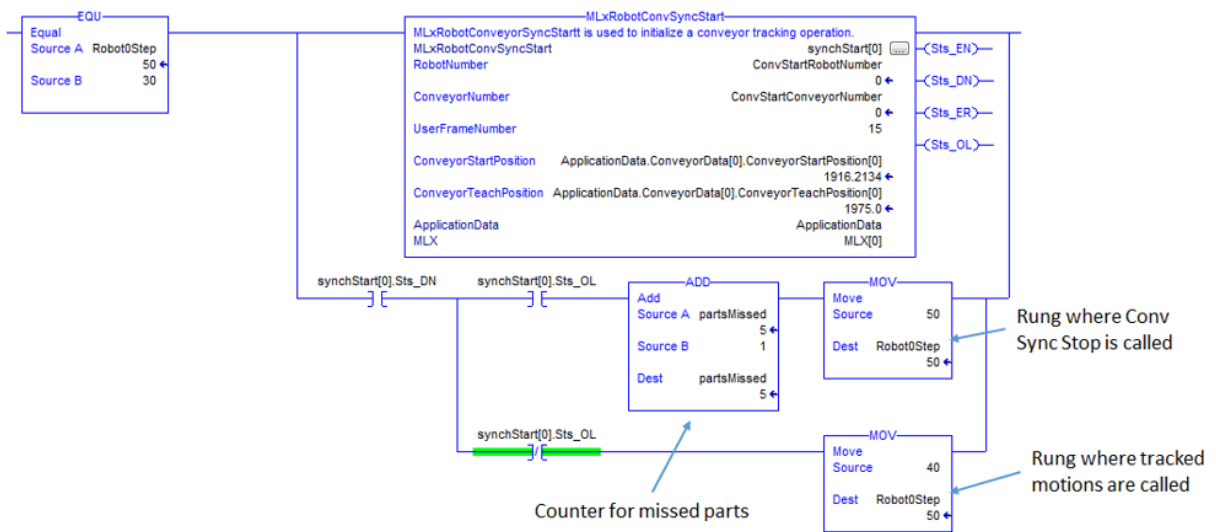
7   Configuration and Maintenance of MLX200 Control Module

The MLX200 Control Module comes pre-configured with a Status Display and Maintenance Tool which allows the user to get a quick glance at the current status of the MLX200 Control Module and the PLC Interface software. This information will serve the purpose of being able to perform basic troubleshooting of PLC/network connectivity issues and startup or configuration issues. The tool also provides the user with the capability to perform maintenance operations such as retrieving log files, backing up and restoring configuration files or firmware.

7.1   MLX200 Control Module Status Display

The display output of the MLX200 Control Module will continuously display the {Status Display} screen as shown in *Fig.7-1 "Status Display of MLX200 Control Module"*. This screen can be viewed by connecting a standard DVI or VGA monitor to the MLX200 Control Module.

Fig. 7-1: Status Display of MLX200 Control Module

The top half of {Status Display} screen shows the IP Address of the Control Module and the license configuration. The bottom half displays information regarding the MLX200 PLC Interface Application (MLX-R.exe) such as whether the application is currently running, whether it is connected to the PLC, and whether the system is currently in a fault state. The information presented in this screen can be used for troubleshooting configuration, connectivity and startup problems. For example, the user can quickly determine whether MLX200 PLC Interface Application was able to establish connection to the PLC without having to connect to the PC using a separate computer or laptop.

7.1.1 Connecting to MLX200 Control Module Display Remotely

In some installations, it may not be possible to easily access the MLX200 Control Module in order to connect a monitor to view the {Status Display} screen. A remote display application, named cerhost.exe, is supplied along with the MLX200 system for such a situation. A user can run this application on a separate computer and remotely access the display of the MLX200 Control Module. This section describes how to run the remote display application and establish a connection.

First step is to ensure that the User PC is connected to the same network as the MLX200 Control Module, using a wired Ethernet connection. The User PC should be running a Windows operating system (XP or later) in order to be able to run the remote display application.

> **NOTE**
>
> The IP addresses of the User PC, and the MLX200 Control Module must be such that they are in the same subnet, in order to be able to communicate with each other. The IP addresses of these must be 192.168.1.xxx, where xxx can be any number between 1 and 255 that is not used by other devices in the same network. The default IP address of MLX200 Control Module is 192.168.1.200 and the user can change the last number of this IP Address using the {Maintenance} screens (See *Section 7.2.3 "Changing the IP Address of the MLX200 Control Module" on page 7-7* for more information). If the IP address of the User PC is not 192.168.1.xxx, then the user has to change the IP address of the User PC using the Windows network settings before attempting to connect to the MLX200 Control Module.

The remote display application can be run by double clicking on cerhost.exe using Windows Explorer. This will bring up a screen as shown in *Fig.7-2 "MLX200 Control Module Remote Display Application Initial Screen"*.

Fig. 7-2: MLX200 Control Module Remote Display Application Initial Screen

7 Configuration and Maintenance of MLX200 Control Module
7.1 MLX200 Control Module Status Display
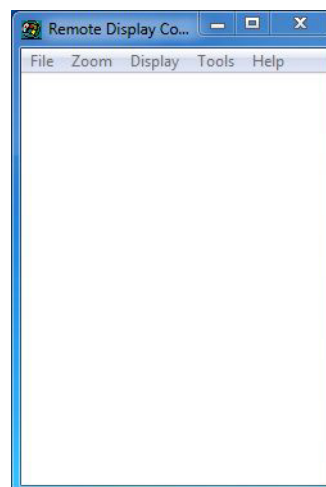
Once the application is running, click on "File" menu item and click on "Connect…" from the drop down menu as shown in *Fig.7-3 "Remote Display Application Connection Process" (left)*. This will bring up a dialog showing a list of active target devices as shown in *Fig.7-3 "Remote Display Application Connection Process" (right)*. The MLX200 Control Module will appear in the list of devices in a few seconds. Once it appears, click on the MLX200_Module listing to select it and press the [OK] button. The display screen of the MLX200 Control Module will now appear on the User PC in a few seconds, as shown in *Fig.7-5 "Status Display screen of MLX200 Control Module viewed through Remote Display Application"*.

Fig. 7-3: Remote Display Application Connection Process
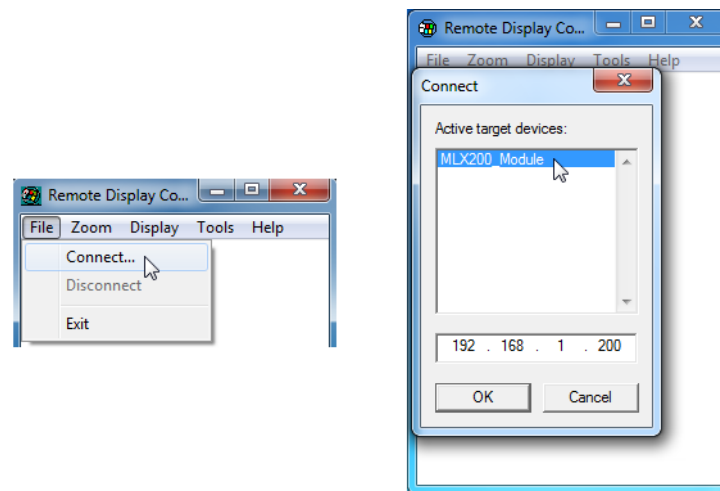
When you click on the "Connect…" drop down menu item in the remote display application for the first time, Windows Firewall will ask the user to allow the connection (*Fig.7-4 "Windows Security Alert Dialog Box"*). Click on the [Allow access] button to allow the connection. This dialog box will be shown only once.

Fig. 7-4: Windows Security Alert Dialog Box

7-3

168542-1CD

> **NOTE**
>
> In some cases, the network connection between the User PC and the MLX200 Control Module may be treated by the User PC as a "Public Network" even though it is actually a private network. In such cases, allow the remote display application to communicate on public networks as well, by clicking on the check box next to "Public Networks" in the Windows Security Alert dialog box. This check box is unchecked by default as shown in *Fig.7-4 "Windows Security Alert Dialog Box"*.

Fig. 7-5: Status Display screen of MLX200 Control Module viewed through Remote Display Application

168542-1CD

| MLX200 Software and Operations | 7   Configuration and Maintenance of MLX200 Control Module
7.2 Maintenance and Configuration Operations |

7.2   Maintenance and Configuration Operations

The Status Display and Maintenance Tool running on the MLX200 Control Module can be used to perform certain configuration and maintenance operations such as changing IP address of the Control, retrieving log files and performing backup and restore of the Control Module. These interactive operations can be done either physically on the Control Module itself by connecting a USB keyboard and USB mouse to the USB ports of the Control Module, or through the Remote Display application running on the User PC. Using the Remote Display application, the user can interact with the {Maintenance Operations} screen using the keyboard and mouse of the User PC itself. The following sections describe the procedure to perform some of the available configuration and maintenance operations.

7.2.1   Logging in to Perform Maintenance Operations

In order to perform any interactive maintenance operation, the user needs to log in to the MLX200 Control Module first. Click on the [Login] button in the {Status Display} screen. This displays the login pop-up dialog as shown in *Fig.7-6 "Logging in to Perform Maintenance Operations"*.

Fig. 7-6: Logging in to Perform Maintenance Operations
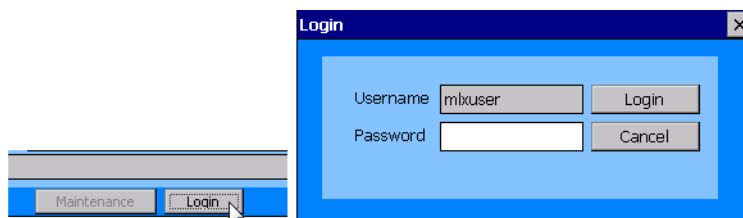
Enter the password that has been configured and click on the [Login] button. If no password has been configured, the default password is mlx200. The default password can be changed to a custom password as described in *Section 7.2.2 "Changing the Password of the MLX200 Control Module"*. Once the user is logged in, the [Maintenance] button in the {Status Display} screen will become enabled. Clicking on the [Maintenance] button will take the user to the MLX200 Control Module {Maintenance Operations} screen as shown in *Fig.7-7 "Maintenance Operations Screen of MLX200 Control Module"*.

168542-1CD

Fig. 7-7: Maintenance Operations Screen of MLX200 Control Module
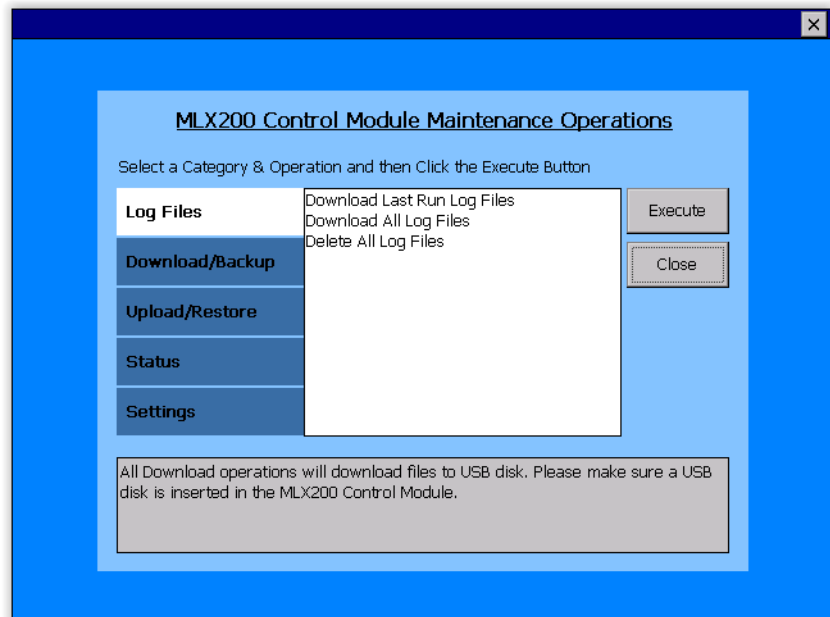
The user can now select a category of operation on the left, then select a specific operation in the middle within that category and then click the [Execute] button to perform the operation. The message display area in the bottom will display status and instructions regarding each operation. The user can click on the [Close] button to return to the {Status Display} screen.

7.2.2   Changing the Password of the MLX200 Control Module
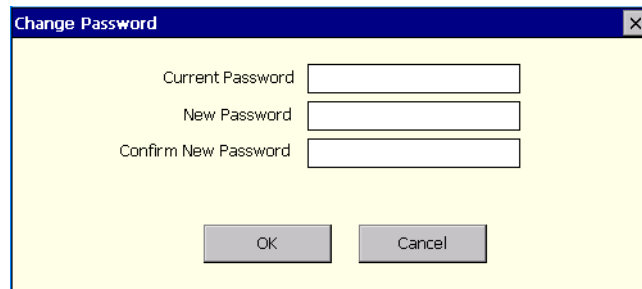
The password required to perform maintenance operations on the MLX200 Control Module can be changed through the following procedure. Click on the "Settings" category in the {Maintenance} screen and select "Change IP Address" from the list of operations as shown in *Fig.7-8 "Selecting the Change Password Operation"*.

Fig. 7-8: Selecting the Change Password Operation
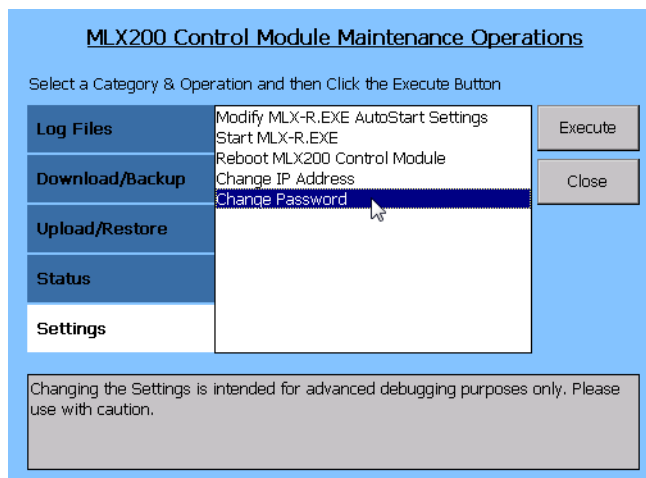
Click on the [Execute] button after selecting the "Change Password" operation and the user will be presented with the Change Password dialog as shown in *Fig.7-9 "Change Password Dialog"*.

Fig. 7-9: Change Password Dialog
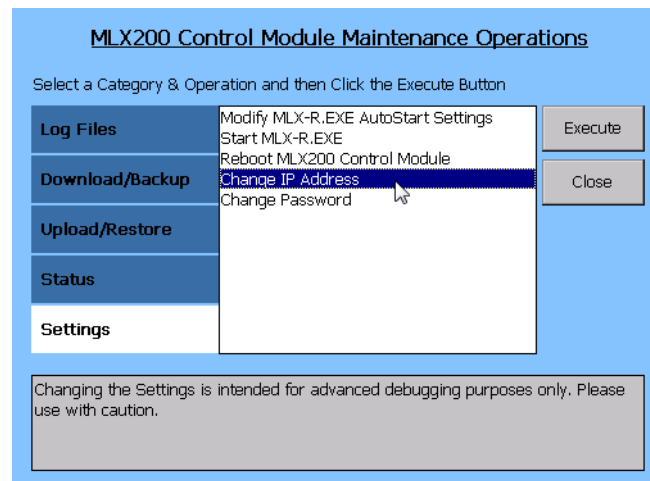
Enter the current password, new password, confirm the new password once again and then click the [OK] button. This will change the password for the MLX200 Control Module.

7.2.3   Changing the IP Address of the MLX200 Control Module

In certain situations, it might be necessary to change the IP Address of the MLX200 Control Module. This might be necessary if the default IP address - 192.168.1.200 - is not usable. The IP Address can be changed using the following procedure. Click on the "Settings" category in the {Maintenance} screen and select "Change IP Address" from the list of operations as shown in *Fig.7-10 "Selecting the Change IP Address Operation"*.

Fig. 7-10: Selecting the Change IP Address Operation

Click on the [Execute] button after selecting the "Change IP Address" operation and the user will be presented with the Change IP Address dialog as shown in *Fig.7-11 "Change IP Address Dialog"*.

Fig. 7-11: Change IP Address Dialog
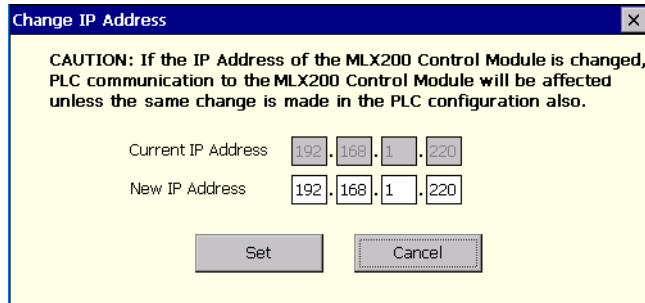
Enter the new IP Address and click on the [Set] button. This will update the IP Address of the MLX200 Control Module. The Control Module should be rebooted for the IP Address change to take effect. Rebooting the Control Module can be done as described in *Section 7.2.4 "Rebooting the MLX200 Control Module"*.

> **NOTE**
>
> If the IP Address of the MLX200 Control Module is changed, the new IP Address should also be updated in the MLX200 Control Module Communications section of the RSLogix project for the PLC, as outlined in *Section 2.2.1.2 "Configuring an MLX200 Control Module Communication" on page 2-8*. If this is not done, the MLX200 Control Module will fail to establish communication with the PLC after its IP Address is changed.

7.2.4 Rebooting the MLX200 Control Module

> ⚠️ **CAUTION**
>
> Rebooting the MLX200 Control Module should be performed only after ensuring that the Control Module is not actively controlling a running robot cell. Failure to do so might result in sudden stoppage of the cell which could lead to equipment damage.

The MLX200 Control Module may need to be rebooted in order to perform
a firmware update or restore operation or if the IP Address of the Control
Module has been updated. The Control Module can be rebooted using the
following procedure. Click on the "Settings" category in the {Maintenance}
screen and select "Reboot MLX200 Control Module" from the list of
operations as shown in *Fig.7-12 "Selecting the Reboot Operation"*.

Fig. 7-12: Selecting the Reboot Operation
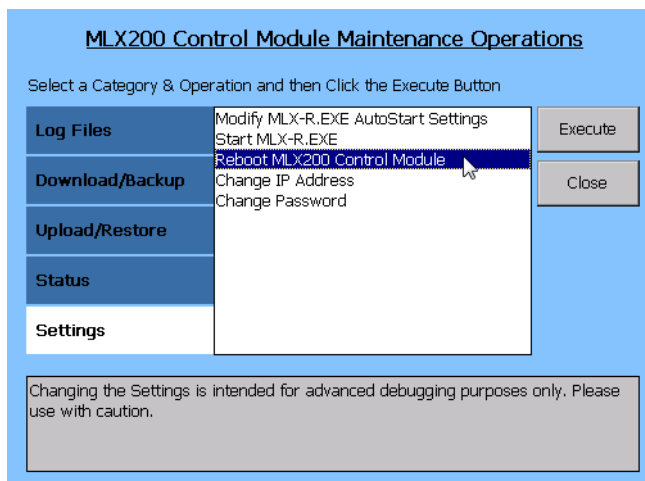
Click on the [Execute] button after selecting the "Reboot MLX200 Control
Module" operation and the user will be presented with the Reboot
Confirmation dialog as shown in *Fig.7-13 "Reboot Confirmation Dialog"*.

Fig. 7-13: Reboot Confirmation Dialog
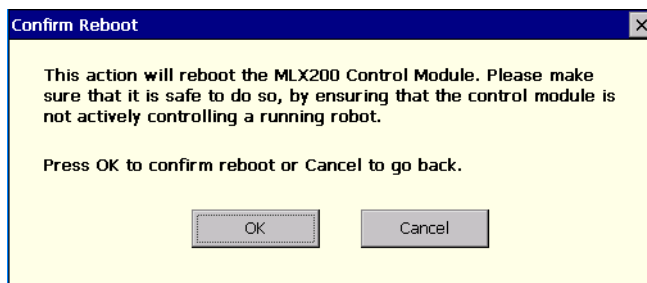
7.2.5 Retrieving Log Files

The MLX200 PLC Interface application running on the MLX200 Control
Module generates log files during its normal course of operation. These
log files contain initialization and configuration information as well as a log
of certain significant events and faults. Retrieving and reviewing the log
files can prove to be useful for trouble shooting alarms or faults and other
connectivity issues. The {Maintenance Operations} screen of the MLX200
Control Module can be used to retrieve the log files using the following
procedure. Click on the "Log Files" category in the {Maintenance} screen
and select one of the operations from the list of operations as shown in
Fig.7-14 "Selecting Log Files Operations".

Fig. 7-14: Selecting Log Files Operations
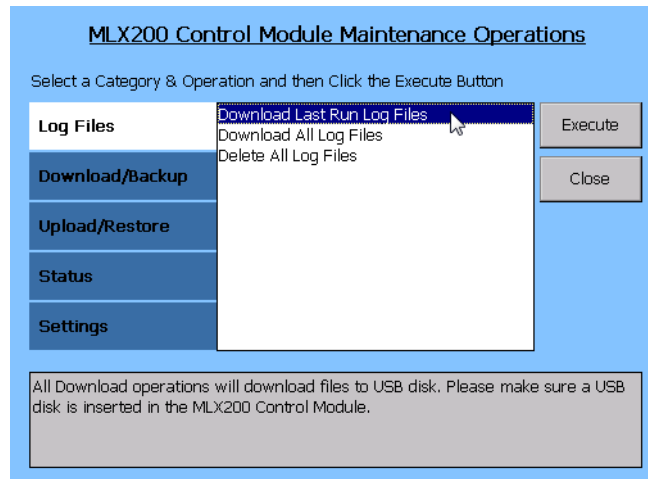
The user can choose to download all log files, or just the log files pertaining to the last run of MLX-R.exe. Select an appropriate operation and then click on the [Execute] button. The log files selected will be downloaded to the USB disk inserted on the MLX200 Control Module. The downloaded log files will be written to a folder on the USB Disk with the folder name including a time stamp of the operation. The folder name used for the operation will be displayed in the message box area towards the bottom of the screen.

NOTE	All download operations from the {Maintenance} screen including the download log file operations require a USB Disk inserted into an available USB port on the MLX200 Control Module.

7.2.6   Updating Configuration and License Files

The MLX200 Control Module contains configuration files and license files that describe the robot cell configuration to the MLX200 PLC Interface Application. In certain situations, it may be necessary to update these files. For example, a new license file may need to be uploaded to enable an optional feature or a new configuration file may need to be uploaded due to a change in hardware configuration. In some cases, a support person from the manufacturer or system integrator might need to review the current license or configuration file in order to troubleshoot a problem. These operations can be performed from the {Maintenance Operations} screen.

In order to download the current configuration file or license file to a USB Disk, click on the "Download/Backup" category on the left and select the respective operation in the middle as shown in *Fig.7-15 "Selecting a Download Operation"*.

Fig. 7-15: Selecting a Download Operation
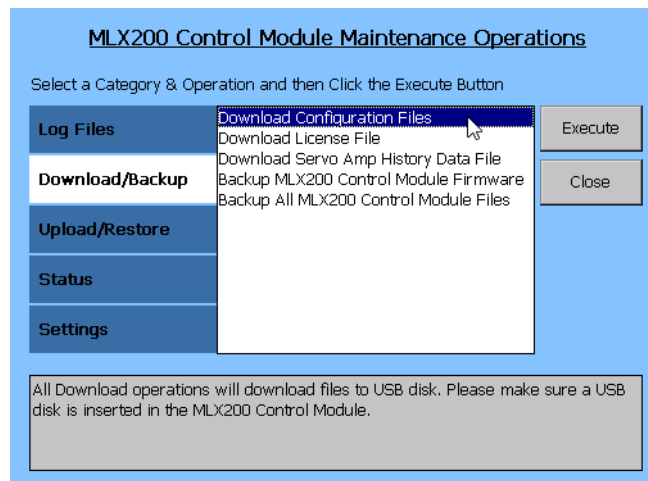
In order to upload a new configuration file or license file, select the
"Upload/Restore" category on the left and select the respective operation
in the middle as shown in *Fig.7-16 "Selecting a Upload Operation"*.

Fig. 7-16: Selecting a Upload Operation
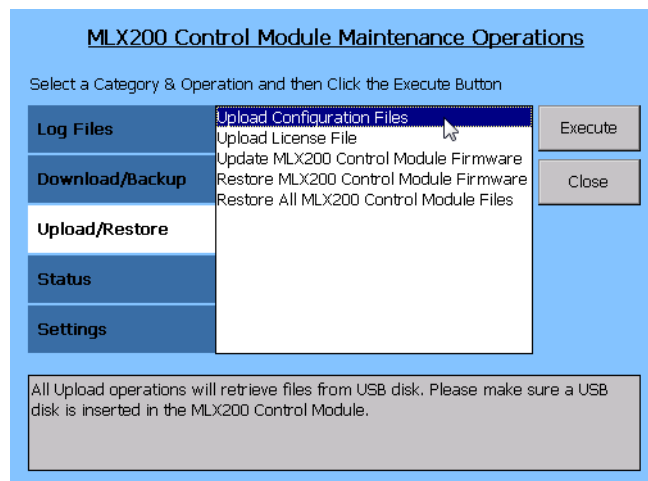
> **NOTE**
>
> All upload operations from the {Maintenance} screen
> require a USB Disk that contains the files to be uploaded, to
> be inserted into an available USB port on the MLX200
> Control Module.

Select the appropriate operation and then click on the [Execute] button. A
new pop-up dialog will be shown as shown in *Fig.7-17 "Selecting a File to
Upload to the MLX200 Control Module"* that allows the user the select the
appropriate configuration file or license file to be uploaded. Select the file
that should be uploaded and then click on the [Select] button. The
selected file will be uploaded to the MLX200 Control Module.

Fig. 7-17: Selecting a File to Upload to the MLX200 Control Module
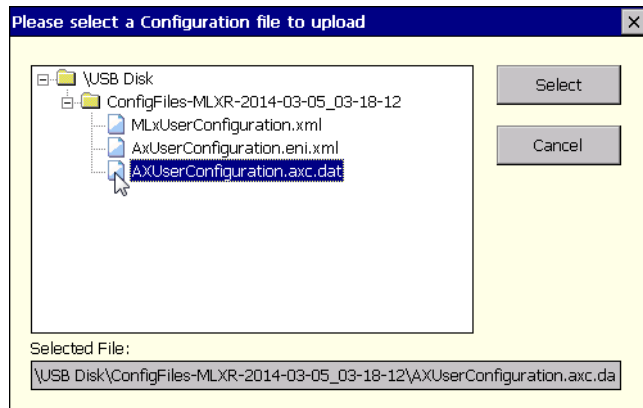
7.2.7 BACKUP AND RESTORE OPERATIONS

The {Maintenance Operations} screen provides the user with the capability to take a backup of the MLX200 Control Module firmware or take a backup of all the Control Module files a precautionary step, so that the firmware or the Control Module can be recovered or restored from the backup if needed. The user can create a backup of just the firmware files which includes the MLX200 PLC Interface Application and its components and dependencies. The user can also create a backup of all Control Module files, which include the system files in addition to firmware files. The firmware or the Control Module can thus be restored from these backups should the need arise.

> **NOTE**
>
> It is recommended to take a firmware backup as well as a full Control Module backup when a new MLX200 system is commissioned, and before making significant changes to the application or system configuration.

The procedure to backup firmware or to backup all Control Module files is similar. Click on the "Download/Backup" category on the left in the {Maintenance Operations} screen, and select the appropriate backup operation as shown in *Fig.7-18 "Selecting a Backup Operation"*.

Fig. 7-18: Selecting a Backup Operation

7   Configuration and Maintenance of MLX200 Control Module
7.2 Maintenance and Configuration Operations
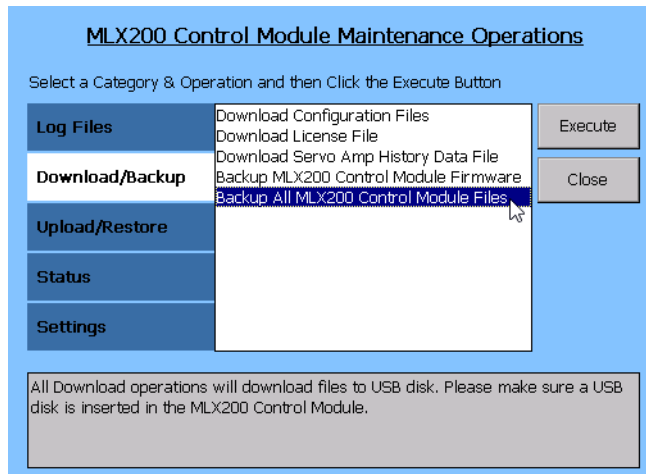
Click on the [Execute] button after selecting the appropriate backup operation. The backed up files will be written to folder on the USB Disk with the folder name including a time stamp of the operation. The folder name used for the operation will be displayed in the message box area towards the bottom of the screen.

> **NOTE**
>
> Performing a backup operation is not permitted when the MLX200 PLC Interface software (MLX-R.exe) is currently running. Before attempting to perform the backup operation, please ensure that the PLC Interface Application is not running through the procedure described in *Section 7.2.9.1 "Disabling Automatic Restart of MLX-R.exe"*.

The procedure to restore the firmware from a backup or restore all Control Module files from backup is similar. Click on the "Upload/Restore" category on the left in the {Maintenance Operations} screen, and select the appropriate restore operation as shown in *Fig.7-19 "Selecting a Restore Operation"*.

Fig. 7-19: Selecting a Restore Operation
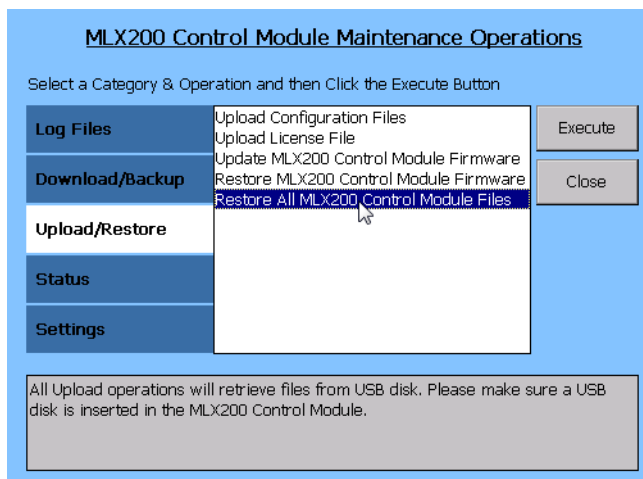
Click on the [Execute] button after selecting the appropriate restore operation. A new pop-up dialog will be shown as shown in *Fig.7-20 "Selecting the Folder Containing Backed Up Files to Restore"* that allows the user the select the folder in the USB disk that contains the backed up files to be used for the restore operation.

Fig. 7-20: Selecting the Folder Containing Backed Up Files to Restore
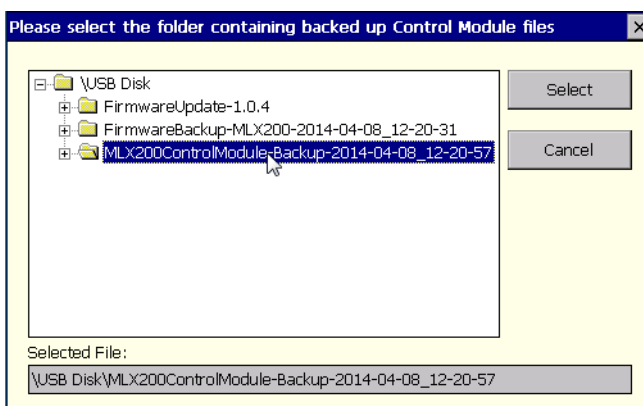
Select the folder that contains the backed up files to be restored and click the [Select] button. The Maintenance Tool will first perform verification of the integrity of the backed up files and then prepare the Control Module for the restore operation. Once this preparation is complete, the actual restore task will be performed after a reboot of the Control Module. Follow the procedure outlined in *Section 7.2.4 "Rebooting the MLX200 Control Module"* in order to reboot and complete the restore operation.

> NOTE
>
> The files and the folders created by the backup procedure described earlier in this section should not be modified in any manner. If any file or folder is modified, the integrity check performed during a restore operation will fail and the restore will not be performed.

After the MLX200 Control Module is rebooted to perform the restore operation, the {Status Display} screen will show the results of the restore operation as shown in *Fig.7-21 "Restore Complete Status Display"*.

Fig. 7-21: Restore Complete Status Display
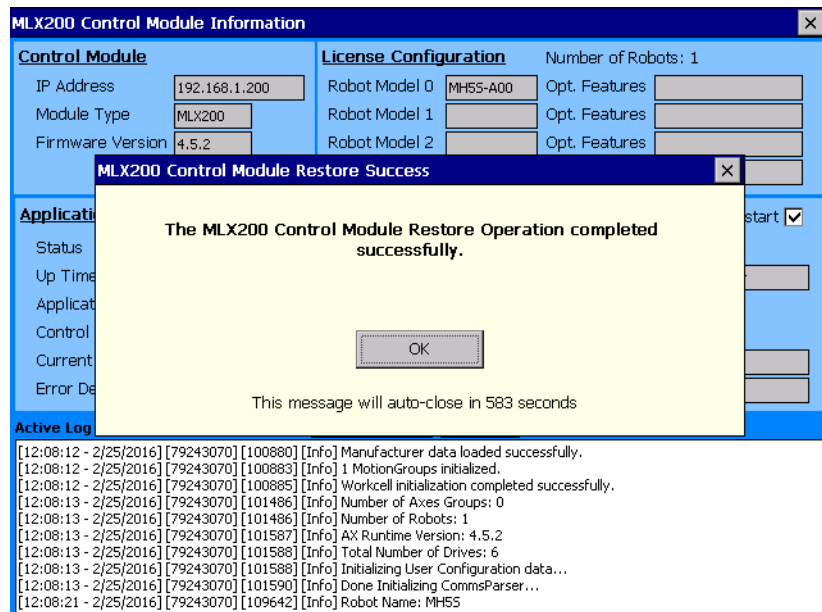
7.2.8   Performing Firmware Update

The manufacturer may occasionally issue a software update package to the MLX200 Control Module Firmware which includes the MLX200 PLC Interface Application and its components and dependencies. The procedure to update the firmware on the MLX200 Control Module using such a firmware update package is described below.

The first step is to copy the entire folder supplied with the firmware update package to a USB Disk. Then insert this USB Disk into an available USB port on the MLX200 Control Module. From the {Maintenance Operations} screen on the Control module, click on the "Upload/Restore" category on the left and select the "Update MLX200 Control Module Firmware" operation as shown in *Fig.7-22 "Selecting the Firmware Update Operation"*.

Fig. 7-22: Selecting the Firmware Update Operation
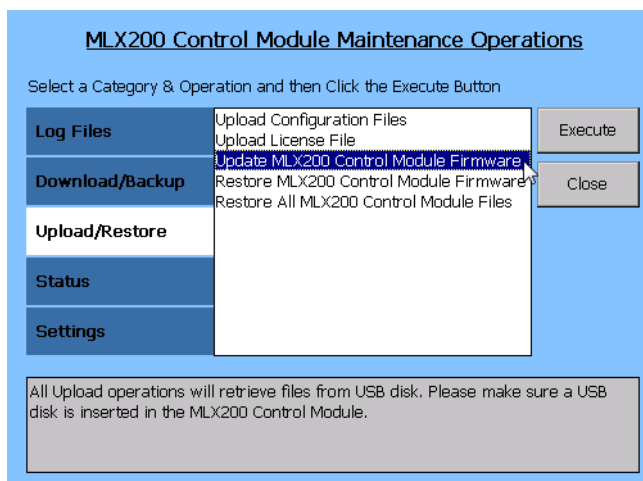
Click on the [Execute] button after selecting the update operation. A new pop-up dialog will be shown as shown in *Fig.7-23 "Selecting the Folder Containing Firmware Update Files"* that allows the user the select the folder in the USB disk that contains the firmware update files.

Fig. 7-23: Selecting the Folder Containing Firmware Update Files
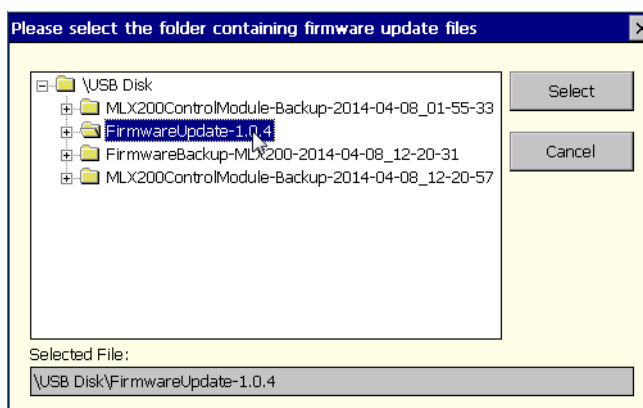
Select the folder that contains the firmware update files and click the [Select] button. The Maintenance Tool will first perform verification of the integrity of the update files and then prepare the Control Module for the update operation. Once this preparation is complete, the actual firmware update task will be performed after a reboot of the Control Module. Follow the procedure outlined in *Section 7.2.4 "Rebooting the MLX200 Control Module"* in order to reboot and complete the firmware update operation.

After the MLX200 Control Module is rebooted to perform the update operation, the {Status display} screen will show the results of the firmware update operation as shown in *Fig.7-24 "Status Display Showing Firmware Update Result"*.

Fig. 7-24: Status Display Showing Firmware Update Result
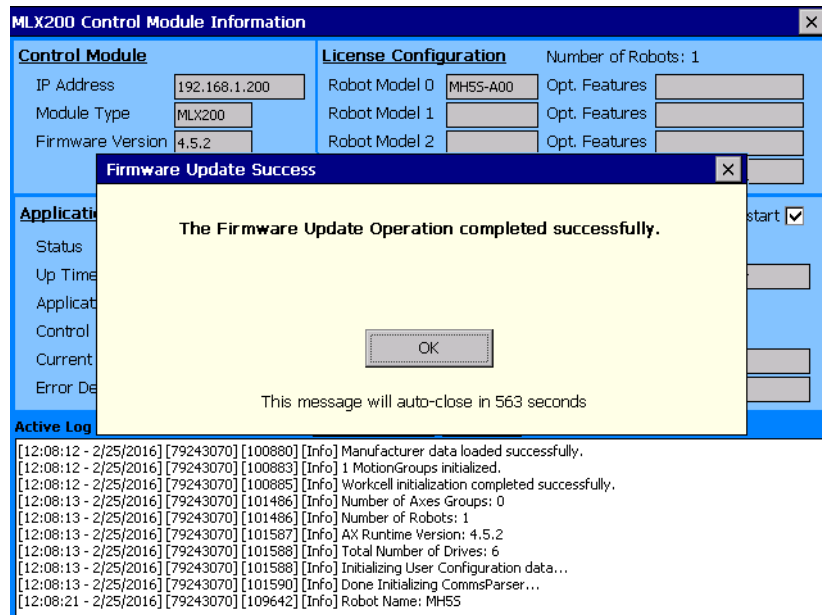
7.2.9   Advanced Operations to Assist with Maintenance and Troubleshooting

The Maintenance Tool enables the user to perform certain additional operations that may be needed for advanced trouble shooting purposes. These include disabling the automatic startup of MLX200 PLC Interface Application (MLX-R.exe) and manually starting the MLX-R.exe application once its automatic startup is disabled. By default, MLX-R.exe is set to start automatically at power up and after it is commanded to restart from the PLC or HMI. However, in certain situations, it may be necessary to disable the automatic restart of MLX-R.exe. For example, when performing a backup of firmware files or backup of all Control Module files, it is recommended to first disable the automatic startup of MLX-R.exe.

7.2.9.1   Disabling Automatic Restart of MLX-R.exe

> ⚠ **CAUTION**
>
> Changing the automatic startup and restart behavior of MLX200 PLC Interface Application (MLX-R.exe) should be done only in consultation with a customer support person. Incorrect settings may cause the MLX200 system to not function properly.

If the need arises to disable the automatic restart of MLX-R.exe, for example to perform a backup operation, click on the "Settings" category on the left side of the {Maintenance Operations} screen and select "Modify MLX-R.exe Autostart Settings" operation as shown in *Fig.7-25 "Selecting the Operation to Modify MLX-R.exe Autostart Settings"*, and click the [Execute] button.

Fig. 7-25: Selecting the Operation to Modify MLX-R.exe Autostart Settings
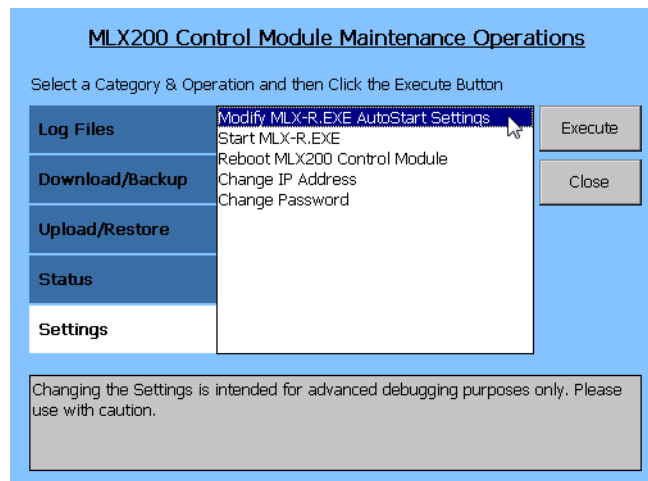
⚠ **CAUTION**

Once the troubleshooting or backup operation is completed, it is important to re-enable the automatic re-start of MLX-R.exe. Failure to do so will cause the MLX200 system to not function as expected. To re-enable the automatic restart of MLX-R.exe, follow the same procedure outlined in this section, with the exception that in the "AutoStart Settings" pop-up dialog (*Fig.7-26 "Auto-start Settings for MLX-R.exe"*), make sure the check box next to "Automatically re-launch MLX-R.EXE after it shuts down" is checked and click on the [Set] button.

A pop-up dialog showing the auto-start settings for MLX-R.exe will be displayed, as shown in *Fig.7-26 "Auto-start Settings for MLX-R.exe"*. Uncheck the check box that reads "Automatically re-launch MLX-R.EXE after it shuts down" and click on the [Set] button. If MLX-R.exe is already running at this point, it can be commanded to restart from the PLC program or through the MLX200 HMI. However, MLX-R.exe will not restart since the automatic restart was disabled from the {Maintenance Operations} screen. At this point the backup operation can be performed. Disabling the auto-start of MLX-R.exe may also be useful to troubleshoot persistent connectivity problems or initialization problems.

Fig. 7-26: Auto-start Settings for MLX-R.exe

168542-1CD
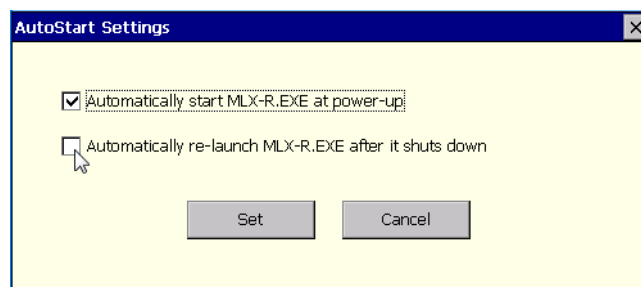
7.2.9.2   Manually Starting MLX-R.exe After Auto-start is Disabled

In situations where the automatic restart of MLX-R.exe is disabled for troubleshooting purposes, it may be necessary manually start MLX-R.exe. This can be done by clicking on the "Settings" category on the left side of the {Maintenance Operations} screen, selecting "Start MLX-R.exe" operation and clicking on the [Execute] button as shown in *Fig.7-27 "Manually Starting MLX-R.exe After its Automatic Restart is Disabled"*.

Fig. 7-27: Manually Starting MLX-R.exe After its Automatic Restart is Disabled
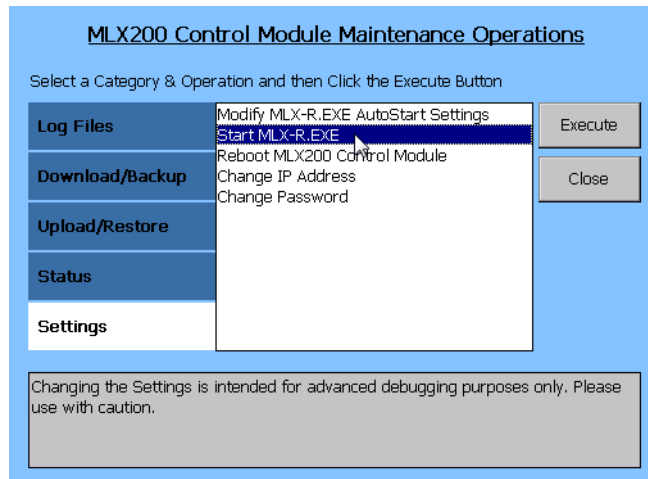
Appendix A

A.1 MLX200 Add-on Instructions

A.1.1 MLxAbort

The MLxAbort instruction is used to command a controlled stop on all servo drives, disable the drives, and place the system into ServosOffAborted state.

Fig. A-1: MLxAbort Instruction
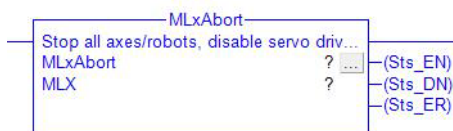
Table A-1: MLxAbort Instruction

Name	Data Type	Usage	Description
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag.

A.1.2 MLxEnable

The MLxEnable instruction is used to enable the servos on all axes/robots and transition the system into Idle state. This must be called before motions can be commanded on the system.

Fig. A-2: MLxEnable Instruction
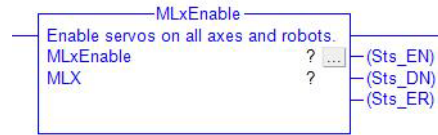
Table A-2: MLxEnable Instruction

Name	Data Type	Usage	Description
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag.

A.1.3 MLxHold

The MLxHold instruction is used to stop all axes/robots in the current path, while maintaining the queue of programmed motions. The system will transition to the Held state after this command. The queued motions can be restarted using the MLxRestart command.

Fig. A-3: MLxHold Instruction

Table A-3: MLxHold Instruction

Name	Data Type	Usage	Description
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.

A.1.4 MLxReset

The MLxReset instruction is used to reset all axes and robots in the system and transition the state of the system to ServosOffReady where the system will be ready to enable. The instruction can fail in some instances where the error state is not removed before resetting (e.g. leaving an emergency stopped pressed). The instruction will also remove all motions previously queued.

Fig. A-4: MLxReset Instruction

Table A-4: MLxReset Instruction

Name	Data Type	Usage	Description
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.5 MLxResetAndHold

The MLxResetAndHold instruction is used to reset the errors in the system. This instruction will not reset the queue of programmed motions unlike the MLxReset instruction. The system will transition to ServosOffHeld state. From the ServosOffHeld state, the programmed motions can be restarted by first transitioning to the Held state by issuing the MLxEnable command and then issuing a MLxRestart command.

Fig. A-5: MLxResetAndHold Instruction

Table A-5: MLxResetAndHold Instruction

Name	Data Type	Usage	Description
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.6 MLxRestart

The MLxRestart instruction is used to restart queued motions when the system is in the Held state.

Fig. A-6: MLxRestart Instruction

Table A-6: MLxRestart Instruction

Name	Data Type	Usage	Description
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.7 MLxStop

The MLxStop instruction is used to bring all Axes and Robots in the system to a controlled stop and then transition to the Idle state.

Fig. A-7: MLxStop Instruction

Table A-7: MLxStop Instruction

Name	Data Type	Usage	Description
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.8 MLxRobotMoveAxisAbsolute

The MLxRobotMoveAxisAbsolute instruction is used to move each axis of the robot to the final absolute commanded position as quickly as possible, with all axes starting and stopping at the same time. While an axis motion results in the shortest travel time, it does not follow a particular path for the TCP. An axis move is the least likely to cause errors when recovering from positions close to travel limits and singularities.The user can specify the target position as well as desired speed, accel/decel, and jerk parameters (when using a Jerk-Limited Velocity Profile). These parameters are specified as a percentage of the max (e.g. 50%).

Fig. A-8: MLxRobotMoveAxisAbsolute Instruction

Table A-8: MLxRobotMoveAxisAbsolute Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
TargetPosition	MLxAppData TeachPoint	InOut	MLxAppDataTeachPoint containing the target position. This position can be taught from the {Teach} Screen using the MLX 200 HMI.
TargetType	BOOL	Input	Axis/TCP Position. 0= Axis Position 1= TCP Position. This will define which data inside the TeachPoint structure to use. In most cases, these values will be the same and lead to the same motion. However, if the data inside the TeachPoint has been modified manually, this parameter can be used to point to the new data.
BlendFactor	DINT	Input	Valid values: 0-8. This will define how much this motion should be blended into the next motion. Note: an additional motion will need to be added to the queue for this parameter to work correctly. See User Guide for detailed instructions on using this parameter.
Speed	REAL	Input	Speed to move axis in% of maximum.
Acceleration	REAL	Input	Acceleration rate for axis in% of maximum.
Deceleration	REAL	Input	Deceleration rate for axis in% of maximum.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished

168542-1CD

Name	Data Type	Usage	Description
Sts_IP	BOOL	Output	In process bit. This Instruction is actively executing, but another instruction may be commanding the active movement.
Sts_AC	BOOL	Output	Active bit HIGH if this motion is currently executing
Sts_PC	BOOL	Output	Process complete bit. HIGH if this motion has reached the end of its commanded trajectory.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
PercentComplete	SINT	Output	Percentage of motion that is completed

A.1.9 MLxRobotMoveAxisRelative

The MLxRobotMoveAxisRelative instruction is used to move each axis of the robot to the final relative commanded position as quickly as possible, with all axes starting and stopping at the same time. While an axis motion results in the shortest travel time, it does not follow a particular path for the TCP. An axis move is the least likely to cause errors when recovering from positions close to travel limits and singularities. The user can specify the target position as well as desired speed, accel/decel, and jerk parameters (when using a Jerk-Limited Velocity Profile). These parameters are specified as a percentage of the max (e.g. 50%).

Fig. A-9: MLxRobotMoveAxisRelative Instruction

Table A-9: MLxRobotMoveAxisRelative Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
DeltaPosition	REAL[7]	InOut	Target Relative Axis Position. If Robot has less than 7 axes, only the first n are used where n is the number of axes for the robot.
BlendFactor	DINT	Input	Valid values: 0-8. This will define how much this motion should be blended into the next motion. Note: an additional motion will need to be added to the queue for this parameter to work correctly. See User Guide for detailed instructions on using this parameter.
Speed	REAL	Input	Speed to move axis in % of maximum.
Acceleration	REAL	Input	Acceleration rate for axis in % of maximum.
Deceleration	REAL	Input	Deceleration rate for axis in % of maximum.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_IP	BOOL	Output	In process bit. This Instruction is actively executing, but another instruction may be commanding the active movement.
Sts_AC	BOOL	Output	Active bit HIGH if this motion is currently executing

Name	Data Type	Usage	Description
Sts_PC	BOOL	Output	Process complete bit. HIGH if this motion has reached the end of its commanded trajectory.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
PercentComplete	SINT	Output	Percentage of motion that is completed

A.1.10   MLxRobotMoveLinearAbsolute

The MLxRobotMoveLinearAbsolute instruction is used to initiate a Linear motion of the robot TCP in Cartesian space to an absolute target position. The result is a straight line trajectory for the robot TCP. The user can specify the target position as well as desired speed, accel/decel, and jerk parameters (when using a Jerk-Limited Velocity Profile). These parameters can be specified as either absolute values (e.g. 750 mm/sec) or a percentage of the max speed (e.g. 50%). The max translational and rotational speeds can be found in the MLX[].Robot[]. ConfigurationData data structure.

Fig. A-10: MLxRobotMoveLinearAbsolute Instruction

Table A-10: MLxRobotMoveLinearAbsolute Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
TargetPosition	MLxAppData TeachPoint	InOut	MLxAppDataTeachPoint containing the target position. This position can be taught from the {Teach} Screen using the MLX200 HMI
TargetType	Bool	Input	Axis/TCP Position. 0= Axis Position 1= TCP Position. This will define which data inside the TeachPoint structure to use. In most cases, these values will be the same and lead to the same motion. However, if the data inside the TeachPoint has been modified manually, this parameter can be used to point to the new data.
BlendFactor	DINT	Input	Valid values: 0-8. This will define how much this motion should be blended into the next motion. Note: an additional motion will need to be added to the queue for this parameter to work correctly. See User Guide for detailed instructions on using this parameter.
Speed	REAL	Input	Speed to move axis in % of maximum.
UseRotationalSpeed	BOOL	Input	Units to use for Speed. 0 = (linear units)/sec, 1= deg/sec. When set to 1, the speed of a linear motion between two points is defined as the time is takes to rotate the product at the defined rotational speed. The rotation and linear translation will stop at the same time.

Name	Data Type	Usage	Description
SpeedUnits	DINT	Input	0 = % Maximum, 1 = Absolute Value in position units/sec
Acceleration	REAL	Input	Acceleration rate for axis in % of maximum.
Deceleration	REAL	Input	Deceleration rate for axis in % of maximum.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_IP	BOOL	Output	In process bit. This Instruction is actively executing, but another instruction may be commanding the active movement.
Sts_AC	BOOL	Output	Active bit HIGH if this motion is currently executing
Sts_PC	BOOL	Output	Process complete bit. HIGH if this motion has reached the end of its commanded trajectory.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
PercentComplete	SINT	Output	Percentage of motion that is completed

168542-1CD

A.1.11 MLxRobotMoveLinearRelative

The MLxRobotMoveLinearRelative instruction is used to initiate a Linear motion of the robot TCP in Cartesian space to a relative target position (e.g. a position 100 mm away in the Z direction). The result is a straight line trajectory for the robot TCP. The user can specify the delta position as well as desired speed, accel/decel, and jerk parameters (when using a Jerk-Limited Velocity Profile). These parameters can be specified as either absolute values (e.g. 750 mm/sec) or a percentage of the max speed (e.g. 50%). The max translational and rotational speeds can be found in the MLX[].Robot[].ConfigurationData data structure. In addition, the user can specify a Coordinate Frame to do the motion relative to:

- 0, World: System performs a linear motion relative to the world frame using the incremental values found in the DeltaPosition input parameter.

- 1, Tool: System performs a linear motion relative to the active tool pose along the tool coordinate system using the incremental values found in the DeltaPosition input parameter.

- 2, User: System will perform a linear motion relative to the active user frame using the incremental values found in the DeltaPosition input parameter.

Fig. A-11: MLxRobotMoveLinearRelative Instruction

Table A-11: MLxRobotMoveLinearRelative Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
DeltaPosition	MLxRobotPosition	InOut	A MLxRobotPosition structure containing the relative target TCP coordinates and closure information.
BlendFactor	DINT	Input	Valid values: 0-8. This will define how much this motion should be blended into the next motion. Note: an additional motion will need to be added to the queue for this parameter to work correctly. See User Guide for detailed instructions on using this parameter.
Speed	REAL	Input	Speed to move axis in % of maximum.

Name	Data Type	Usage	Description
UseRotationalSpeed	BOOL	Input	Units to use for Speed. 0 = (linear units)/sec, 1= deg/sec. When set to 1, the speed of a linear motion between two points is defined as the time is takes to rotate the product at the defined rotational speed. The rotation and linear translation will stop at the same time.
SpeedUnits	DINT	Input	0 = % Maximum, 1 = Absolute Value in position units/sec
Acceleration	REAL	Input	Acceleration rate for axis in % of maximum.
Deceleration	REAL	Input	Deceleration rate for axis in % of maximum.
CoordFrame	DINT	Input	0 = World, 1 = Tool, 2 = User
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_IP	BOOL	Output	In process bit. This Instruction is actively executing, but another instruction may be commanding the active movement.
Sts_AC	BOOL	Output	Active bit HIGH if this motion is currently executing
Sts_PC	BOOL	Output	Process complete bit. HIGH if this motion has reached the end of its commanded trajectory.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
PercentComplete	SINT	Output	Percentage of motion that is completed

A-15

A.1.12 MLxRobotMoveCircular

The MLxRobotMoveCircular instruction is used to initiate a Circular Arc motion of the robot TCP in Cartesian space through two target TCP Positions. The result is a circular arc trajectory for the robot TCP that uses the current position as well as the ViaPosition and FinalPosition parameters to calculate the circular path. The user can specify the target position as well as desired speed, accel/decel, and jerk parameters (when using a Jerk-Limited Velocity Profile). These parameters can be specified as either absolute values (e.g. 750 mm/sec) or a percentage of the max (e.g. 50%). The max translational and rotational speeds can be found in the MLX[].Robot[].ConfigurationData data structure.

The rotational motion of the TCP is defined by the RotationType parameter as follows:

- 0 = Use initial orientation (no rotational motion)

- 1 = Interpolate initial and final orientation (i.e. interior orientation is ignored)

- 2 = Variable Interpolation (interpolate interior orientation position)

> **NOTE**
>
> The case of Holding and Restarting a circular motion with RotationType = 2, the restarted motion will interpolate directly to the final orientation (i.e. similar to RotationType=1).

Fig. A-12: MLxRobotMoveCircular
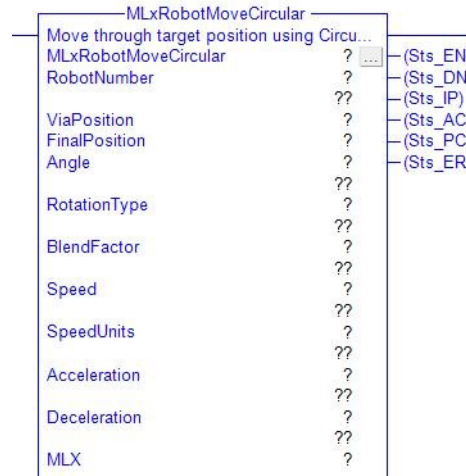
Table A-12: MLxRobotMoveCircular

Name	Data Type	Usage	Description
MLxRobotMoveCircular	MLxRobotMoveCircular	InOut	
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ViaPosition	MLxAppDataTeachPoint	Input	A MLxRobotPosition structure containing the target TCP coordinates and closure information for the circular arc Via Position.
FinalPosition	MLxAppDataTeachPoint	InOut	A MLxRobotPosition structure containing the target TCP coordinates and closure information for the circular arc Final Position.
Angle	REAL	Input	Angle of the circular arc to draw. For example, 360.0 will draw a full circle while 720.0 will draw two circles. If set to 0.0, the circular arc will interpolate the angle to the final position.
RotationType	REAL	Input	0 = Use initial orientation (no rotational motion), 1 = Interpolate initial and final orientation (i.e. middle orientation is ignored), 2 = Variable Interpolation (interpolate interior orientation position)
BlendFactor	DINT	Input	Valid values: -1-8. This will define how much this motion should be blended into the next motion. Note: an additional motion will need to be added to the queue for this parameter to work correctly. See User Guide for detailed instructions on using this parameter.
Speed	REAL	Input	Speed to move axis
SpeedUnits	DINT	Input	0 = % Maximum, 1 = Absolute Value in position units/sec
Acceleration	REAL	Input	Acceleration rate for axis in % of maximum.
Deceleration	REAL	Input	Deceleration rate for axis in % of maximum.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled.
Sts_DN	BOOL	Output	Done bit. This will turn HIGH when the command has been processed and added to the motion queue. The rung must stay enabled until this bit is active.
Sts_IP	BOOL	Output	In process bit. This Instruction is actively executing, but another instruction may be commanding the active movement.
Sts_AC	BOOL	Output	Active bit. HIGH if this motion is currently executing.
Sts_PC	BOOL	Output	Process complete bit. HIGH if this motion has reached the end of its commanded trajectory.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
PercentComplete	SINT	Output	Percentage of motion that is completed

A.1.13   MLxRobotJogAxes

The MLxRobotJogAxes instruction is used to manually jog the axes of the robot. The Directions parameter is an array the defines the direction to jog each axes with 1 being in the positive direction and -1 being in the negative direction. The Directions parameter is a DINT[7] array with only the first n parameters being used where n is the number of axes in the robot. The Speed is defined as a % of maximum speed.

If using the MLX200 HMI, the {Teach} Screen can be used instead of directly calling this AOI from application logic.

Fig. A-13: MLxRobotJogAxes Instruction
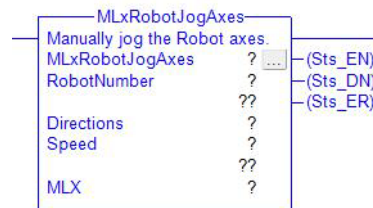
Table A-13: MLxRobotJogAxes Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
Directions	DINT[7]	InOut	Defines the job directions for each robot axis: 0 - Positive, 1 - Negative. If robot has less than 7 axes remaining values are ignored.
Speed	REAL	Input	Speed to move axis in % of maximum.
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.14 MLxRobotJogAxesToPoint

The MLxRobotJogAxesToPoint instruction is used to manually jog the robot to a target axes position in axis-interpolated motion. The TargetAxes parameter defines the target position, and the Speed is defined as a % of maximum speed.

If using the MLX 200 HMI, the {Teach} Screen can be used instead of directly calling this AOI from application logic.

Fig. A-14: MLxRobotJogAxesToPoint Instruction

Table A-14: MLxRobotJogAxesToPoint Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
TargetPoisition	MLxAppDataTeachPoint	InOut	Target Position to jog the robot to, in Axis jog mode
Speed	REAL	Input	Speed to move axis in % of maximum.
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.15 **MLXRobotJogTCP**

The MLxRobotJogTCP instruction is used to manually jog the TCP of the robot. The Directions parameter is an array the defines the direction to jog each coordinate direction (X,Y,Z,RX,RY,RZ). The Speed can be specified as either absolute values (e.g. 750 mm/sec) or a percentage of the max speed (e.g. 50%) depending on the SpeedUnits value. The CoordFrame parameter defines which Coordinate Frame to perform the jogging in:

- 0 - World Frame

- 1 - Tool Frame

- 2 - User Frame

If using the MLX 200 HMI, the {Teach} Screen can be used instead of directly calling this AOI from application logic.

Fig. A-15: MLxRobotJogTCP Instruction
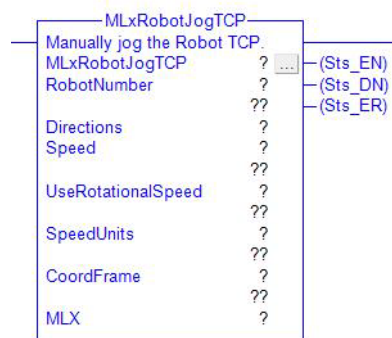
Table A-15: MLxRobotJogTCP Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
Directions	DINT[6]	InOut	Defines the job directions for each robot axis: 0 - Positive, 1 - Negative. If robot has less than 7 axes remaining values are ignored.
Speed	REAL	Input	Speed to move axis in % of maximum.
UseRotationalSpeed	BOOL	Input	Units to use for Speed 0 = (linear units)/sec, 1= deg/sec. When set to 1, the speed of a linear motion between two points is defined as the time is takes to rotate the product at the defined rotational speed. The rotation and linear translation will stop at the same time
SpeedUnits	DINT	Input	0 = % Maximum, 1 = Absolute Value in position units/sec
CoordFrame	DINT	Input	0 = World, 1 = Tool, 2 = User
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.16 MLxRobotJogTCPToPoint

The MLxRobotJogTCPToPoint instruction is used to manually jog the robot to a target TCP position. The TargetTCP parameter contains the (X,Y,Z,RX,RY,RZ) position of the target as well as closure information. The Speed can be specified as either absolute values (e.g. 750 mm/sec) or a percentage of the max speed (e.g. 50%) depending on the SpeedUnits value.

If using the MLX 200 HMI, the {Teach} Screen can be used instead of directly calling this AOI from application logic.

Fig. A-16: MLxRobotJogTCPToPoint Instruction

Table A-16: MLxRobotJogTCPToPoint Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
TargetPoisition	MLxAppDataTeachPoint	InOut	Target Position to jog the robot to, in Axis jog mode
Speed	REAL	Input	Speed to move axis in % of maximum.
UseRotationalSpeed	BOOL	Input	Units to use for Speed 0 = (linear units)/sec, 1= deg/sec. When set to 1, the speed of a linear motion between two points is defined as the time is takes to rotate the product at the defined rotational speed. The rotation and linear translation will stop at the same time
SpeedUnits	DINT	Input	0 = % Maximum, 1 = Absolute Value in position units/sec
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.17 MLxRobotCoordinateTransform

The MLxRobotCoordinateTransform instruction can be used to update the Axis/TCP position of a Teach Point or to convert a TCP position between World and User coordinates.

Fig. A-17: MLxRobotCoordinateTransform Instruction

Table A-17: MLxRobotCoordinateTransform Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
TeachPoint	MLxAppDataTeachPoint	InOut	MLxAppDataTeachPoint structure. Either the Axis or TCP position inside this structure will be updated depending on the TransformType value.
TransformType	DINT	Input	0 = Convert Axis Coordinates to TCP Coordinates, 1 = Convert TCP Coordinates to Axis Coordinates, 2 = Convert TCP in World Frame to Active User Frame, 3 = Convert TCP in Active User Frame to World Frame.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.

168542-1CD

A.1.18   MLxRobotSetBasePose

The MLxRobotSetBasePose instruction is used to set the position and orientation of the robot with respect to the World Frame. Changing a Robot's Base Pose will not cause any motion but will change the TCP position reported from MLxRobotCoordinateTransform. This is particularly useful in cells with multiple robots where you want TCP positions to be correctly reported in the work cell's World Frame.

Fig. A-18: MLxRobotSetBasePose Instruction
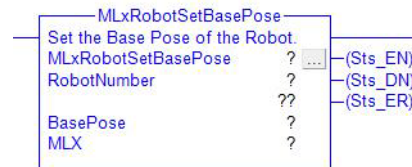
Table A-18: MLxRobotJogTCPToPoint Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
BasePose	REAL[6]	InOut	Base Pose Value (X, Y, Z, RX, RY, RZ)
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.19 MLxRobotSetCubicIZByCenterPoint

The MLxRobotSetCubicIZByCenterPoint instruction is used to define a Cubic IZ (Interference Zone) by defining a Center Point and the Dimension around it (length, width, height). The IZAction parameter is used to define the action of the IZ:

- 0 - When the IZ is entered, the bit in MLX[].Robot[].CubicIZStatus corresponding to the ZoneID will turn high. This can then be used in application logic to perform or prevent certain operations while the robot is inside the zone.

- 1 - When the TCP attempts to enter the IZ, the Robot will Abort and transition to ServosOffAborted state. This is useful when the Robot has delicate items in the workspace that should be avoided.

- 2 - Deactivate the current IZ.

Fig. A-19: MLxRobotSetCubicIZByCenterPoint Instruction

Table A-19: MLxRobotSetCubicIZByCenterPoint Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ZoneID	DINT	Input	Internal number identifier for IZ. This number will be used in the error messages
CenterPoint	REAL[6]	InOut	Defines position of coordinate frame to be the center of the IZ.
Dimensions	REAL[3]	InOut	Defines the distance along each of the Center Points axes to define the cube. See User Guide for more information.
IZAction	DINT	Input	0 = MonitorOnly, 1= StopMotion, 2 = Clear IZ
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.20 MLxRobotSetCubicIZByTwoCorners

The MLxRobotSetCubicIZByTwoCorners instruction is used to define a Cubic IZ (Interference Zone) by defining two corners of a cube in Cartesian space. The IZAction parameter is used to define the action of the IZ:

- 0 - When the IZ is entered, the bit in MLX[].Robot[].CubicIZStatus corresponding to the ZoneID will turn high. This can then be used in application logic to perform or prevent certain operations while the robot is inside the zone.

- 1 - When the TCP attempts to enter the IZ, the Robot will Abort and transition to ServosOffAborted state. This is useful when the Robot has delicate items in the workspace that should be avoided.

- 2 - Deactivate the current IZ.

Fig. A-20: MLxRobotSetCubicIZByTwoCorners Instruction

Table A-20: MLxRobotSetCubicIZByTwoCorners Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ZoneID	DINT	Input	Internal number identifier for IZ. This number will be used in the error messages
Corner1	REAL[3]	InOut	[X, Y, Z] position of first corner.
Corner2	REAL[3]	InOut	[X, Y, Z] position of second corner.
IZAction	DINT	Input	0 = MonitorOnly, 1= StopMotion, 2 = Clear IZ
CoordFrame	DINT	Input	0 = Robot 1 = Base, 2 = User
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.21 MLxRobotSetFrameShift

The MLxRobotSetFrameShift instruction allows a user to execute a Cartesian offset that is applied to subsequent moves. This is useful for defining programmatic patterns such as a pelletizing application where the first product is located, and the rest of the positions are offsets from that position. The CoordFrame parameter defines the frame to shift the motion relative to:

- 0 - World Frame

- 1 - [UNUSED]

- 2 - User Frame (note: by default, the active User Frame is the same as the World Frame. Use the MLxRobotSetUserFrame instruction to set a new User Frame).

Fig. A-21: MLxRobotSetFrameShift Instruction

Table A-21: MLxRobotSetFrameShift Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
FrameShift	REAL[6]	InOut	Frame Shift Value (X, Y, Z, RX, RY, RZ)
CoordFrame	DINT	Input	0 = World 1 = Unused, 2 = User
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.22 MLxRobotSetToolProperties

The MLxRobotSetToolProperties instruction is used to update the offset between the Tool Plate and TCP position of the Robot. The ToolPose parameter defines the (X,Y,Z,RX,RY,RZ) offsets for the new tool position. A new Tool Pose will change the TCP position of the Robot reported from MLxRobotCoordinateTransform and will also affect the TCP positions passed into Motion instructions as parameters. The Tool Pose is also used when doing Relative or Jogging motions in the Tool Frame.

Fig. A-22: MLxRobotSetToolProperties Instruction

Table A-22: MLxRobotSetToolProperties Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ToolNumber	DINT	Input	Index of the tool to use from the ApplicationDataTools[] array
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
ApplicationData	MLxApplicationData	InOut	The MLxApplicationData Control Module scope tag
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.23 MLxRobotSetUserFrame

The MLxRobotSetUserFrame instruction is used to set the Active User Frame for a Robot. The UserFrame parameter is a REAL[6] array containing the (X,Y,Z,RX,RY,RZ) coordinates of the User Frame in World Coordinates. Changing the active User Frame will affect Relative and Jogging Motions in the User Frame.

Fig. A-23: MLxRobotSetUserFramer Instruction

Table A-23: MLxRobotSetFrameShift Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
UserFrameNumber	DINT	Input	Index of the user frame to use, from the ApplicationDataUserFrames[] array
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Input	If the instruction fails, this parameter contain the error code. Call MLxGetErrorDetail to get detailed error information.
ApplicationData	MLxApplicationData	InOut	The MLxApplicationData Control Module scope tag
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.24 **MLxRobotCollisionDetection**

The MLxRobotCollisionDetection instruction is used to perform various actions for Collision Monitoring and Detection.

Valid actions are:

- 0 - Start Measurement Mode. This action will begin internally recording torque disturbance values. In this mode, the Torque Disturbance parameter is ignored and the system will not abort at detected collisions.

- 1 - Start Execution Mode. This action will turn on Collision Detection monitoring with the provided Allowable Torque Disturbance values. In this mode, the system will abort if a disturbance is measured larger than the Allowable Disturbance.

- 2 - Stop Collision Monitoring. This will turn off Collision Monitoring and return the largest disturbance value for each axis since Collision Monitoring was started (i.e. Action = 0 or 1).

- 3 - Get Maximum Torque Disturbance. This will return the current maximum disturbance value for each axis but will not affect the operation of Collision Monitoring.

- 4 - Reset Maximum Torque Disturbance. This will reset the current internal maximum disturbance values for each axis but will not affect the operation of Collision Monitoring.

Fig. A-24: MLxRobotCollisionDetection Instruction

Table A-24: MLxRobotCollisionDetection Instruction

Name	Data Type	Usage	Description
MLxRobotCollisionDetection	MLxRobotCollisionDetection	InOut	
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
CollisionFile	DINT	Input	The Collision File to use. Valid values are 0 to ApplicationData.NumberOfCollision Files-1
Action	DINT	Input	Use this to define the action taken by the instruction. Valid actions are: 0 - Start Measurement Mode. 1 - Start Execution Mode. 2 - Stop Collision Monitoring. 3 - Get Maximum Torque Disturbance. 4 - Reset Maximum Torque Disturbance.
ApplicationData	MLxApplicationData	InOut	The MLxApplicationData Control Module scope tag
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This will turn HIGH when the command has been processed. The rung must stay enabled until this bit is active.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.

168542-1CD

A.1.25 MLxRobotConvSyncStart

MLxRobotConvSyncStartt is used to initialize a conveyor tracking operation. This AOI will wait until a product has passed the ConveyorStartPosition and then sync with the conveyor. All commands performed after an MLxRobotConvSyncStartt command will track the conveyor until an MLxConvSyncStop is called

Fig. A-25: MLxRobotConvSyncStart Instruction

Table A-25: MLxRobotConvSyncStart Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ConveyorNumber	DINT	Input	The number of the conveyor corresponding to the data in ApplicationDataConveyorData Valid values are 0-3
UserFrameNumber	DINT	Input	The User Frame number that stores the X direction for the conveyor. Note: this User Frame must be defined in the ApplicationData, but does not need to be the active User Frame
ConveyorStartPosition	REAL	Input	This the position in mm at which point the system will initialize conveyor tracking operations.
ConveyorTeachPositon	REAL	Input	This is the position in mm where the teach points for the following motion commands were taught
ApplicationData	MLxApplicationData	InOut	The MLxApplicationData Control Module scope tag
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
Sts_OL	BOOL	OutPut	This bit turns high if the first object in the queue is past the MLX[] ConveyorData[], MaxStartPosition value

168542-1CD

A.1.26 MLxRobotConvSyncStop

MLxRobotConvSyncStop is used to stop conveyor tracking operations and update the object queue.

Fig. A-26: MLxRobotConvSyncStop Instruction

Table A-26: MLxRobotConvSyncStop Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ConveyorNumber	DINT	Input	The number of the conveyor corresponding to the data in ApplicationDataConveyorData Valid values are 0-3
ConveyorStartPosition	REAL	Input	This the position in mm at which point the system will initialize conveyor tracking operations.
KeepInQueue	BOOL	Input	If set to 1, this part will be moved to the robot specified in NewRobotQueue. If set to 0, the part will be removed from the queue and no longer tracked.
NewRobotQueue	DINT	Input	If KeepInQueue is set to 1, this parameter holds the robot to move the part to. If NewRobotQueue is the same as RobotNumber, the part is held in the current robot's queue.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.

A.1.27 MLxRobotConvSyncStopWithAxisMot

The MLxRobotConvSyncStopWithAxisMot command is used to stop
conveyor tracking operations and blend directly into an Absolute Axis
motion.

Fig. A-27: MLxRobotConvSyncStopWithAxisMot Instruction
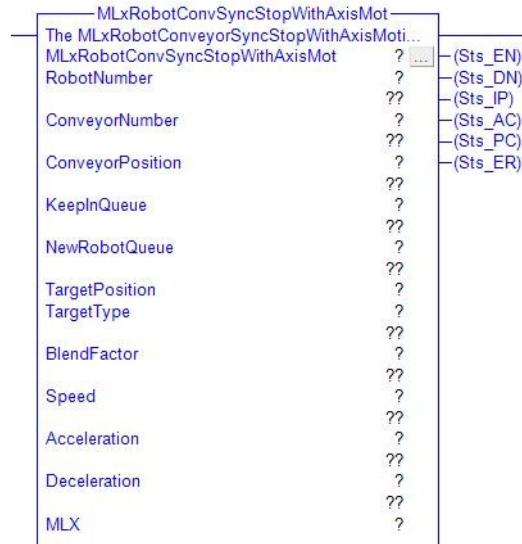
Table A-27: MLxRobotConvSyncStopWithAxisMot Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ConveyorNumber	DINT	Input	The number of the conveyor corresponding to the data in ApplicationDataConveyorData Valid values are 0-3
ConveyorPosition	DINT	Input	This value must be equal to the CurrentValue of the conveyor
KeepInQueue	BOOL	Input	If set to 1, this part will be moved to the robot specified in NewRobotQueue. If set to 0, the part will be removed from the queue and no longer tracked.
NewRobotQueue	DINT	Input	If KeepInQueue is set to 1, this parameter holds the robot to move the part to. If NewRobotQueue is the same as RobotNumber, the part is held in the current robot's queue.
TargetPosition	MLxAppDataTeachPoint	InOut	MLxAppDataTeachPoint containing the target position. This position can be taught from the {Teach} Screen using the MLX200 HMI
TargetType	BOOL	Input	Axis/TCP Position. 0= Axis Position 1= TCP Position. This will define which data inside the TeachPoint structure to use. In most cases, these values will be the same and lead to the same motion. However, if the data inside the TeachPoint has been modified manually, this parameter can be used to point to the new data.

Name	Data Type	Usage	Description
BlendFactor	DINT	Input	Valid values: 0-8. This will define how much this motion should be blended into the next motion. Note: an additional motion will need to be added to the queue for this parameter to work correctly. See User Guide for detailed instructions on using this parameter.
Speed	REAL	Input	Speed to move axis in % of maximum.
SpeedUnits	DINT	Input	0 = % Maximum, 1 = Absolute Value in position units/sec
Acceleration	REAL	Input	Acceleration rate for axis in % of maximum.
Deceleration	REAL	Input	Deceleration rate for axis in % of maximum.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_IP	BOOL	Output	In process bit. This Instruction is actively executing, but another instruction may be commanding the active movement.
Sts_AC	BOOL	Output	Active bit HIGH if this motion is currently executing
Sts_PC	BOOL	Output	Process complete bit. HIGH if this motion has reached the end of its commanded trajectory.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
PercentComplete	SINT	Output	Percentage of motion that is completed

A.1.28 MLxRobotConvSyncStopWithLinearMot

The MLxRobotConvSyncStopWithLinearMot command is used to stop conveyor tracking operations and blend directly into an Absolute Linear motion

Fig. A-28: MLxRobotConvSyncStopWithLinearMot Instruction
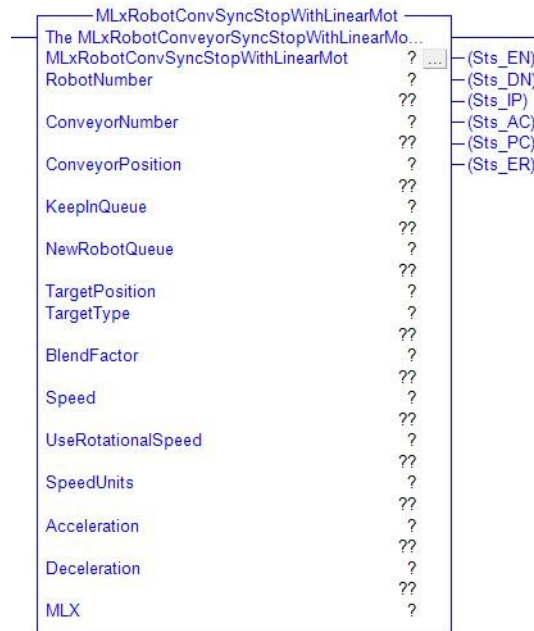
Table A-28: MLxRobotConvSyncStopWithLinearMot Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
ConveyorNumber	DINT	Input	The number of the conveyor corresponding to the data in ApplicationDataConveyorData Valid values are 0-3
ConveyorPosition	DINT	Input	This value must be equal to the CurrentValue of the conveyor
KeepInQueue	BOOL	Input	If set to 1, this part will be moved to the robot specified in NewRobotQueue. If set to 0, the part will be removed from the queue and no longer tracked.
NewRobotQueue	DINT	Input	If KeepInQueue is set to 1, this parameter holds the robot to move the part to. If NewRobotQueue is the same as RobotNumber, the part is held in the current robot's queue.
TargetPosition	MLxAppDataTeachPoint	InOut	MLxAppDataTeachPoint containing the target position. This position can be taught from the {Teach} Screen using the MLX200 HMI
TargetType	BOOL	Input	Axis/TCP Position. 0= Axis Position 1= TCP Position. This will define which data inside the TeachPoint structure to use. In most cases, these values will be the same and lead to the same motion. However, if the data inside the TeachPoint has been modified manually, this parameter can be used to point to the new data.

Name	Data Type	Usage	Description
BlendFactor	DINT	Input	Valid values: 0-8. This will define how much this motion should be blended into the next motion. Note: an additional motion will need to be added to the queue for this parameter to work correctly. See User Guide for detailed instructions on using this parameter.
Speed	REAL	Input	Speed to move axis in % of maximum.
UseRotationalSpeed	BOOL	Input	Units to use for Speed. 0 = (linear units)/sec, 1= deg/sec. When set to 1, the speed of a linear motion between two points is defined as the time is takes to rotate the product at the defined rotational speed. The rotation and linear translation will stop at the same time.
SpeedUnits	DINT	Input	0 = % Maximum, 1 = Absolute Value in position units/sec
Acceleration	REAL	Input	Acceleration rate for axis in % of maximum.
Deceleration	REAL	Input	Deceleration rate for axis in % of maximum.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_IP	BOOL	Output	In process bit. This Instruction is actively executing, but another instruction may be commanding the active movement.
Sts_AC	BOOL	Output	Active bit HIGH if this motion is currently executing
Sts_PC	BOOL	Output	Process complete bit. HIGH if this motion has reached the end of its commanded trajectory.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
PercentComplete	SINT	Output	Percentage of motion that is completed

A.1.29 MLxGetErrorDetail

The MLxGetErrorDetail instruction is used to retrieve detailed error information from the MLX200 Control Module. This error information will populate the passed in MLxErrorDetail structure that contains the detailed error message as well as fields describing the Type of error, Origin of error, Remedy, and others. Note: if using the MLX200 HMI, the error details should be automatically updated from inside the HMI Task.

Fig. A-29: MLxGetErrorDetail

Table A-29: MLxGetErrorDetail Instruction

Name	Data Type	Usage	Description
ErrorDetail	MLxErrorDetail	InOut	A MLxErrorDetail parameter that will contain the detailed error information
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.30 MLxGetModuleInfo

The MLxGetModuleInfo instruction is used to retrieved information about the configuration of the connected MLX200 Robot Control Module. This information includes the type of Control Module, the IP and MAC addresses, and the firmware version. This information will get automatically populated during initialization.

Fig. A-30: MLxGetModuleInfo

Table A-30: MLxGetModuleInfo Instruction

Name	Data Type	Usage	Description
ModuleInfo	MLxModuleInfo	InOut	A MLxModuleInfo data structure that will contain the MLX200 Control Module Info.
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.31 MLxReadDigitalInputs

The MLxReadDigitalInputs instruction is used to read digital inputs that have been wired into a servo drive. These digital inputs will need to be wired into the hardware panel to be used. When this AOI is called, the IO1-IO4 values will map to digital inputs defined for the servo drive type. On a Yaskawa SigmaV drive IO1=SI0, IO2=SI1, IO3=SI2 and IO4=SI3.

Fig. A-31: MLxReadDigitalInputs Instruction

Table A-31: MLxReadDigitalInputs Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	Robot to read digital inputs from. Valid values are 0 to MLX[].NumberOfRobots-1.
AxisNumber	DINT	Input	Robot axis to read digital inputs from. Range is 0-Robot[].ConfigurationData.NumberOfAxes-1
IO1	BOOL	Output	Mapped to SI0 on Yaskawa Sigma V-drives.
IO2	BOOL	Output	Mapped to SI1 on Yaskawa Sigma V-drives.
IO3	BOOL	Output	Mapped to SI2 on Yaskawa Sigma V-drives.
IO4	BOOL	Output	Mapped to SI3 on Yaskawa Sigma V-drives.
MLX	MLxData	InOut	Control Module-scope DataStructure Containing information for MLXsystem.

A.1.32 MLxWriteDigitalOutputs

The MLxWriteDigitalOutputs instruction is used to write digital outputs that have been wired into a servo drive. When this AOI is called, the IO1-IO4 values will map to digital outputs defined for the servo drive type. For example, on a Yaskawa SigmaV drive IO1=SO1, IO2=SO2, IO3=SO3 and IO4 is unused.

Fig. A-32: MLxWriteDigitalOutputs Instruction

```
┌─ MLxWriteDigitalOutputs ─┐
│ Write Digital Outputs to servo drives. │
│ MLxWriteDigitalOutputs      ?  ... │
│ RobotNumber                 ?     │
│                            ??     │
│ AxisNumber                  ?     │
│                            ??     │
│ IO1                         ?     │
│                            ??     │
│ IO2                         ?     │
│                            ??     │
│ IO3                         ?     │
│                            ??     │
│ IO4                         ?     │
│                            ??     │
│ MLX                         ?     │
└──────────────────────────┘
```
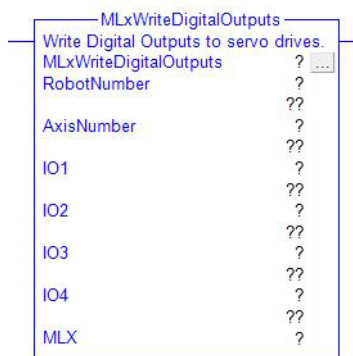
Table A-32: MLxWriteDigitalOutputs Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	Robot to write digital outputs on. Valid values are 0 to MLX[].NumberOfRobots-1.
AxisNumber	DINT	Input	Robot axis to write digital outputs to. Range is Robot[].ConfigurationData.NumberOfAxes-1
IO1	BOOL	Input	Mapped to SI0 on Yaskawa Sigma V-drives.
IO2	BOOL	Input	Mapped to SI1 on Yaskawa Sigma V-drives.
IO3	BOOL	Input	Mapped to SI2 on Yaskawa Sigma V-drives.
IO4	BOOL	Input	Unused
MLX	MLxData	InOut	Control Module-scope DataStructure Containing information for MLX system.

A.1.33 MLxRobotGetHomeOffsets

The MLxRobotGetHomeOffsets is used to retrieve the current home positions for each robot axis.

Fig. A-33: MLxRobotGetHomeOffsets Instruction

Table A-33: MLxRobotGetHomeOffsets Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
OffsetType	DINT	Input	Specifies the type of offset value to retrieve. 0 = Get the offsets measured at current robot location, 1 = Get the home offset values currently in use by the system
Offset	DINT[7]	InOut	Contains the retrieved offset values
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Output	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A.1.34 MLxRobotSetHomeOffsets

The MLxRobotSetHomeOffsets is used to update the Home Position of
the robot. In order for the new home values to take place, the system must
be restarted after calling this AOI.

Fig. A-34: MLxRobotGetHomeOffsets Instruction

Table A-34: MLxRobotGetHomeOffsets Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
OffsetType	DINT	Input	Specifies the type of offset operation to perform. 0 = Set home offsets as offsets measured at current robot location, 1 = Use the home offset values specified in the Offsets parameter.
Offset	DINT[7]	InOut	Offset values to set when OffsetType parameter value is 1.
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
ErrorCode	DINT	Inputs	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag

A-43

A.1.35 MLxRobotGetProperties

The MLxRobotGetProperties instruction is used to read the configuration parameters of a Robot. These properties will automatically populate the MLX[].Robot[].ConfigurationData as well as the individual axis configurations inside MLX[].Robot[].RobotAxes[]. This AOI will automatically run during system initialization to populate this data.

Fig. A-35: MLxRobotGetProperties Instruction

Table A-35: MLxRobotGetProperties Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
ErrorCode	DINT	Inputs	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.

A.1.36 MLxRobotSetProperties

The MLxRobotSetProperties instruction is used to update the Robot Configuration Data (e.g. TCP speed/acceleration limits, Axis position/speed/acceleration limits, etc). This data should be directly changed inside the MLX[].Robot[].ConfigurationData and MLX[].Robot[].RobotAxes[].ConfigurationData structure before calling the AOI. Then, when this AOI is called, the new configuration parameters will be stored; however, these parameters will not persist when the system is restarted. If different limits are desired, call this AOI during application initialization

Fig. A-36: MLxRobotSetProperties Instruction

Table A-36: MLxRobotSetProperties Instruction

Name	Data Type	Usage	Description
RobotNumber	DINT	Input	The robot commanded by this instruction instance. Valid values are 0 to MLX[].NumberOfRobots-1.
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. If using HMI, view detailed error message on {HMI} screen. Otherwise, call MLxGetErrorDetail for more information.
MLX	MLxData	InOut	The MLxData Control Module Scope tag
ErrorCode	DINT	Inputs	If the instruction fails, this parameter will contain the error code. Call MLxGetErrorDetail to get detailed error information.

A.1.37 MLxSetGlobalParameter

The MLxSetGlobalParamer is for setting an MLX Global Parameter. The type of parameter can be set using the ParameterType field. Available parameter types include:

- **0 - Speed Scale % (5-100)**. This sets the global speed scale for all motions. Default value is 100.

> NOTE
>
> - In order to maintain the same motion shape, this command will also scale the accelerations/decelerations of the motion profiles. Thus, a Speed Scale of 50% will not yield exactly twice the cycle time.
> - The MLxSetGlobalParameter only affects motions that are placed in cue after entering the instructions. To slow down motions immediately enter instruction MLxHold, MLxSetGlobalParameter, MlxRestart.
> - Contact the Add-On Instruction developer for questions or problems with this instruction

Table A-37: MLxSetGlobalPrameter Instructions

Name	Data Type	Usage	Description
MLxSetGlobalParameter	MLxSetGlobalParameter	InOut	
ParameterType	DINT	Input	Type of global parameter to set. Options: 0 - Speed Scale % (5-100)
ParameterValue	REAL	Input	Value of the global parameter.
Sts_EN	BOOL	Output	Enable bit. This bit will stay high as long as the ladder rung is enabled.
Sts_DN	BOOL	Output	Done bit. This bit will turn high when the instruction has finished.
Sts_ER	BOOL	Output	Error bit. Indicates an error during instruction execution. Use MLxGetSystemErrorDetail for a detailed error message.
MLX	MLxData	InOut	The MLxData Controller Scope tag.

Appendix B

B.1   MLX200 Control Module Performance Results and Memory Usage

The following tables contain basic performance results measured running MLX200 on a 1756-L75 ControlLogix Controller, an L35E CompactLogix Controller (old generation) and an L27ERM CompactLogix Controller (newer generation). These results will vary depending on the Controller model in use, but should provide a general understanding of the impact and trade-offs. The Motion Instruction Timing Results measure the latency between a MLX200 AOI scanning and the MLX200 Controller Module acknowledging receipt of the command (i.e. round-trip communication). The CPU Load is measured using the Logix5000 Task Monitor.

Table B-1: Performance Results on ControlLogix 1756-L75

Test Description	Average (ms)	Min. (ms)	Max. (ms)	Std. Dev. (ms)	CPU Load
1.0 *ms* RPI, 2.0 *ms* Task	7.541	3.84	9.92	1.119	23 - 24%
2.0 *ms* RPI, 2.0 *ms* Task	6.936	3.84	9.984	1.535	23 - 24%
1.0 *ms* RPI, 5.0 *ms* Task	11.204	9.728	14.976	2.198	9 -10%
2.0 *ms* RPI, 5.0 *ms* Task	11.042	9.742	14.972	2.098	9 -10%

Table B-2: Performance Results on ControlLogix L35E

Test Description	Average (ms)	Min. (ms)	Max. (ms)	Std. Dev. (ms)	CPU Load
1.0 *ms* RPI, 3.0 *ms* Task	9.579	8.512	12.288	1.297	54 - 55%
2.0 *ms* RPI, 3.0 *ms* Task	9.880	38.192	15.072	1.151	54 - 55%
1.0 *ms* RPI, 5.0 *ms* Task	14.383	9.296	19.856	1.513	23 - 24%
2.0 *ms* RPI, 5.0 *ms* Task	14.405	9.536	19.904	1.455	23 - 24%

> **NOTE**
> A 2ms MLX Task was causing Task Overruns on the L35E CompactLogix. This value was changed to 3 ms for these tests.

Table B-3: Performance Results on ControlLogix L27ERM

Test Description	Average (ms)	Min. (ms)	Max. (ms)	Std. Dev. (ms)	CPU Load
1.0 *ms* RPI, 2.0 *ms* Task	4.883	3.648	7.782	1.016	26 - 27%
2.0 *ms* RPI, 2.0 *ms* Task	3.940	3.648	5.920	0.294	26 - 27%
1.0 *ms* RPI, 5.0 *ms* Task	9.897	9.672	10.08	0.037	10 - 11%
2.0 *ms* RPI, 5.0 *ms* Task	9.900	9.728	10.064	0.034	10 - 11%

Appendix B
B.1 MLX200 Control Module Performance Results and Memory Usage

The below figure shows the memory used by an empty MLX200 project
(all needed AOIs, UDTs, Tasks included but not application data or tasks)

NOTE	The included project memory usage may be slightly higher as they will have the Example Applications already added. This memory shows the usage before any application data is included.

Fig. B-1: Memory Usage by an Empty MLX200 Control Module

Estimated Data and Logic Memory
Total: 33,554,432 bytes
Free: 32,261,784 bytes
Used: 1,292,648 bytes
Max Used: 1,292,648 bytes
Largest Block Free: 32,261,784 bytes
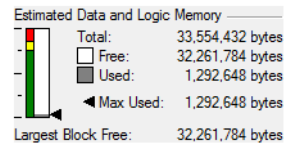
Appendix C

C.1   MLX200 Control Module Error Code List

Table C-1: Error Code List

MLx Error #	OEM Error #	Message	Description	Remedy
0	0	No Error		Nothing to remedy.
12	12	File I/O Error	Missing or Corrupt file.	Verify file location and existence.
13	13	Fatal Internal Error	Error details.	Restart system and contact customer support if error persists.
14	14	License Verification Failed	Error details.	Retrieve log files and verify features in Info.log.
15	15	Alarm File Error.	An error occurred loading the Alarm data file.	Restart system and contact customer support if error persists.
16	16	Configuration File Loading Failed	An error occurred loading the MLX configuration files.	Consult log files and contact customer support if problem persists.
17	17	Hardware Initialization Failed	An error occurred while initializing hardware.	Check all power and communications connections. Consult log files.
18	18	Control Module Initialization Failed	An operation was attempted, but the Control Module was not initialized.	Consult log files and restart MLX200 Control Module.
19	19	System Already Initialized	Initialize function was called on a Control Module that was already initialized.	Ensure that Initialize function of Control Module is not called multiple times.
20	20	Invalid Index passed into a command or instruction	An invalid index has been passed into a command or instruction.	Ensure that indices passed into the API are within limits.
21	21	Invalid Parameter passed into a command or instruction	An invalid parameter has been passed into a command or instruction.	Ensure that parameters passed into the API are valid.
23	23	Motion Group Initialization Failed	The Motion Group referenced is not initialized.	Ensure that the Motion Group is initialized before accessing it.
24	24	Vector or Matrix Size Mismatch	The provided Vector or Matrix is the incorrect size.	Call API again with correctly sized object.
25	25	Feature Not Supported	The requested feature is not supported or enabled.	Retrieve log files and verify features in Info.log.
26	26	Motion Aborted	The Motion was aborted due to error in a previous motion segment or safety signal event.	Consult alarm queue to determine what motion caused an error.
27	27	State Machine Initialization Failed	An error occurred while initializing state machine.	Review log files to identify what caused the error.
28	28	State Machine Internal Failure	The State Machine encountered an unexpected event.	Consult log files to identify what caused the error

MLx Error #	OEM Error #	Message	Description	Remedy
29	29	Command Not Allowed.	The requested command is not allowed in current state or mode.	Review the documentation to verify what commands are allowed in each state.
30	30	Command Failed	The requested command failed to execute in the current state.	Consult the log file and alarms queue to determine the cause of failure.
31	31	Invalid Control Module Mode	The requested operation is invalid in the current Control Module operation mode.	Hardware operations are allowed only when the Control Module operation mode is MLX200_HARDWARE_MODE.
32	32	Command Ignored	No action was required to complete requested state change.	No action required.
33	33	Data File Error	An error occurred while initializing the persisted servo data.	Restart system and contact customer support if problem persists.
34	34	Last Enabled Position Validation Error	A discrepancy with last enabled position and the current actual position was detected.	Verify home position or recalibrate home position.
35	35	Version Mismatch Detected	The versions of MLX-R and MLX-D do not match for this system.	Verify MLX-R and MLX-D version inside MLX[].MLxControllerInfo.
36	36	Error processing MLX Tuning Parameter Configuration	An error has occurred processing the MLxTuningParameters.dat file.	Verify validity of MLxTuningParameters.dat and contact customer support if problem persists.
100	100	Hardware Initialization Failed	Error connecting to hardware control panel.	Verify that the servo panel and servo drives are powered on, and that the EtherCAT cable is securely connected on both ends.
101	101	Hardware Initialization Failed	EtherCAT bus configuration does not match the configuration file.	Verify that all the servo drives and auxilliary devices on EtherCAT network are powered on, and that the hardware configuration (ENI) files are correct.
102	102	Hardware Initialization Failed	Hardware configuration (ENI) file not found.	Verify that the hardware configuration (ENI) files are present in the control module. Copy the configuration files from backup if available or contact
103	103	Hardware Initialization Failed	Hardware configuration (ENI) file format incorrect.	Verify that the hardware configuration (ENI) files in the control module are not corrupted. Copy the configuration files from backup if available or contact
104	104	Hardware Initialization Failed	Encoder connection error reported on one or more of the servo drives.	Verify that the encoder cable is plugged in securely at both ends. Verify that the 24V power supply and the associated fuses or circuit breakers are not tripped.
105	105	Hardware Initialization Failed	Servo Parameters on one of the servo drives appear to be not set correctly.	Contact support to reload drive parameters. This error typically occurs when a servo drive is replaced.
106	106	Hardware Initialization Failed	SetupDrive: Fault reset on failed on one of the servo drives.	Try power cycling the hardware control panel. If the problem persists, contact support with the error message and logs.

168542-1CD

MLx Error #	OEM Error #	Message	Description	Remedy
205	205	Communication Error PLC1 Ethernet Error: 205(0)	Fuji Teach Pendant is not able to communicate with the PLC **NOTE:** This error message appears on the Fuji Pendant screen.	Verify project setting match actual IP settings of Teach Pendant and the PLC in use. Verify power is applied to all units and Ethernet cabling is connected. Reference the "MLX200 Hardware Installation & Software Upgrade Manual" (168283-1CD) - Chapter 11 "Teach Pendant Software Upgrade Procedure" for project configuration and troubleshooting.
300	300	Motion Planner Error: Invalid Parameter	Invalid Parameter Passed to Motion Planner.	Retrigger command or instruction with valid parameter value.
301	301	Axis Position Travel Limit Reached	Axis Position Travel Limit Reached.	Change the commanded position to avoid violation of position travel limits.
302	302	Axis Speed Violation	Axis Speed Violation.	Reduce commanded speed or change commanded position.
303	303	Unable to Move as Requested	Unable to Move as Requested.	Change commanded position, or change type of move to PTP, or change move speed.
304	304	Interference Zone Violation	Interference Zone Violation.	Change commanded position to be compliant with defined interference zones.
305	305	Soft Travel Position Limit Violation	Soft Travel Position Limit Violation.	Move axis out of limit and restart.
306	306	Frame Shift Parameter Error	Frame Shift Parameter Error.	Enter valid parameters and call instruction again.
307	307	Unexpected internal error	Unexpected error.	Consult log file or contact customer support for more information.
308	308	Hardware following error	Hardware following error. Excessive following error reported from servo drives.	Reduce speed and/or payload.
309	309	Servo Defined Position Limit reached	Servo Defined Position Limit reached. Servos reporting position overtravel.	Release all limits or brakes and move axis away from limit.
311	311	Unknown servo error	Unknown servo error. Error code generated by servos does not exist.	Record alarm code on servos and contact customer support.
312	312	Axis Acceleration Violation	Axis Acceleration Violation.	Lower acceleration or deceleration values for offending motion.
313	313	Internal Exception Error	Internal Exception Error.	Restart system and contact customer support if error persists.
314	314	Servo Update Rate Error	Servos not being updated at desired rate.	Restart system and contact customer support if error persists.
315	315	Conveyor Tracking feature not enabled	Conveyor Tracking feature not enabled.	Contact customer support to enable conveyor tracking feature.

MLx Error #	OEM Error #	Message	Description	Remedy
316	316	External Exception Error	External Exception Error.	Restart system and contact customer support if error persists.
317	317	Servo torque limit reached	Servo torque limit reached.	Lower acceleration and/or speed values and reset system.
319	319	Axis and TCP positions do not match in target position	Axis and TCP positions do not match in target position. Check active tool or reteach point.	Reteach point.
320	320	Invalid Interference Zone Number	Invalid Interference Zone Number.	Enter valid Interference Zone Number.
321	321	Invalid Interference Zone Type	Invalid Interference Zone Type.	Enter valid Interference Zone Type.
322	322	Limits Setting Error	Soft Limits cannot be defined larger than Hard Limits.	Enter Soft limits that are smaller than Hard Limits.
323	323	Initialization of Motion Status tracking failed	Initialization of Motion Status tracking failed.	Restart system and contact customer support if error persists.
324	324	Error Planning Motion	Error Planning Motion. Maximum number of motions are currently queued.	Allow some motions to complete before adding more to queue.
500	500	System Monitoring Fault		See remedy details for a specific monitoring fault
501	501	System Monitoring Fault	Excessive TCP speed detected while Jogging. Consult Log file for more information.	Reduce jogging speed.
502	502	System Monitoring Fault	Actual robot position violated servo overtravel limits. Consult Log file for more information.	In Manual (Teach) Mode, jog the axis back so that it is within limits.
503	503	System Monitoring Fault	Actual robot axis speed violated axis maximum speed.	Change commanded position and/or speed.
504	504	System Monitoring Fault	Hardware E-stop. The hardware e-stop has been pressed.	Release E-stop and reset system.
505	505	System Monitoring Fault	Mode switch detected. Press reset to reinitialize system.	Reset system.
506	506	System Monitoring Fault	Timeout waiting for drives to enable.	Restart system and contact customer support if error persists.
507	507	System Monitoring Fault	Break detected in guard circuit.	Check guard circuit status and reset system.
508	508	System Monitoring Fault	A discrepancy with last enabled position was detected. Please verify hardware position before resetting.	Verify Home Position or recalibrate home position.
509	509	System Monitoring Fault	Excessive TCP speed detected during programmed motion. Consult Log file for more information.	Change commanded position, change move type, or reduce speed. Contact support.

MLx Error #	OEM Error #	Message	Description	Remedy
510	510	System Monitoring Fault	Brake release not allowed. Cannot enter brake release mode while in Running state.	Change system state before releasing brakes.
511	511	System Monitoring Fault	Interference zone violation detected.	Change commanded position to be away from interference zone. Contact support.
512	512	System Monitoring Fault	Excessive position increment in subsequent servo commands. Cycle power and restart system.	Restart system and contact customer support if error persists.
513	513	System Monitoring Fault	WatchDog timer failure. Cycle power and restart system.	Restart system and contact customer support if error persists.
514	514	System Monitoring Fault	Fatal Servo Fault detected. Cycle power and restart system.	Restart system and contact customer support if error persists.
515	515	System Monitoring Fault	Actual robot TCP speed violated maximum allowed TCP speed.	Change commanded position and/or speed.
516	516	System Monitoring Fault	Actual axis torque violated soft torque limits. Consult Log file for more information.	Remove potential obstacles from workspace. Lower acceleration or load if needed.
517	517	System Monitoring Fault	One or more drives reported Hardware Base Block status. Hardware Base Block status unexpected in current system state.	Check safety and E-Stop circuits and remove conditions causing drives to be in Hardware Base Block.
518	518	System Monitoring Fault	Mismatch detected between commanded and actual axis positions. Position mismatch unexpected in current system state.	Check safety and E-Stop circuits, review log files.
519	519	System Monitoring Fault	Non-zero commanded velocity detected by Monitor. Non-zero commanded velocity unexpected in current system state.	Consult log files for more information.
520	520	System Monitoring Fault	One or more drives in disabled state detected by Monitor. All drives are expected to be enabled in current system state.	Check safety and E-Stop circuits to determine why some drives are disabled.
521	521	System Monitoring Fault	One or more drives in enabled state detected by Monitor. All drives are expected to be disabled in current system state.	Verify software configuration and consult log files. Contact customer support if problem persists.
522	522	System Monitoring Fault	Unhandled Deadman Switch disengage in Teach Mode detected by Monitor. Deadman Switch disengage in Teach Mode did not result in a timely transition to error states.	Verify software configuration and consult log files. Contact customer support if problem persists.
523	523	System Monitoring Fault	Unhandled Guard circuit signal loss detected by Monitor. Guard signal loss did not result in a timely transition to error states.	Verify software configuration and consult log files. Contact customer support if problem persists.
524	524	System Monitoring Fault	Unhandled E-Stop detected by Monitor. E-Stop signal did not result in a timely transition to error states.	Verify software configuration and consult log files. Contact customer support if problem persists.

MLx Error #	OEM Error #	Message	Description	Remedy
525	525	System Monitoring Fault	Actual Position movement detected by Monitor. Actual servo position is expected to remain constant in the current state.	Verify software configuration and consult log files. Contact customer support if problem persists.
526	526	System Monitoring Fault	Stopping time violation detected by Monitor. The robot did not come to a controlled stop within the configured stopping time.	Verify configured maximum deceleration values against configured stopping time for the robot and adjust either parameter accordingly.
527	527	System Monitoring Fault	Actual robot position violated servo overtravel limits. Consult Log file for more information.	Jog the axis back within limits.
528	528	Collision Detected	A collision has been detected on at least one axis.	Remove obstacles from workspace. Lower acceleration or load if needed.
600	3744	Servo Communication Error	Command-Option IF Servo Unit Initial Error. The initial sequence between the EtherCAT (CoE) Network Module and the SERVOPACK was not completed within 10s.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
601	3745	Servo Communication Error	Command-Option IF Memory Check Error. The communication memory of the EtherCAT (CoE) Network Module and the SERVOPACK is broken.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
602	3746	Servo Communication Error	Command-Option IF Servo Synchronization Error. The data exchange between the EtherCAT (CoE) Network Module and the SERVOPACK was not synchronized.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
603	3747	Servo Communication Error	Command-Option IF Servo Data Error. The communication data between the EtherCAT (CoE) Network Module and the SERVOPACK was inappropriate.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
604	2576	Servo Communication Error	EtherCAT DC Synchronization Error. The Sync0 event and the SERVOPACK cannot be synchronized.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
605	2577	Servo Communication Error	EtherCAT State Error. The EtherCAT AL state became not Operational while the DS402 drive state is in Operation enabled.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
606	2578	Servo Communication Error	EtherCAT Outputs Data Synchronization Error."The events, receive process data and sync0, do not synchronize. (Failed to receive the process data.)"	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
607	2592	Servo Communication Error	Parameter Setting Error. The parameter setting is out of range.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

168542-1CD

MLx Error #	OEM Error #	Message	Description	Remedy
608	2624	Servo Communication Error	System Initialization Error. The initialization at power on sequence was failed.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
609	2625	Servo Communication Error	Communication Device Initialization Error. The ESC initialization was failed.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
610	2631	Servo Communication Error	Loading Servo Information Error. The loading of SERVOPACK information was failed.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
611	2632	Servo Communication Error	EEPROM Parameter Data Error. The checksum in EEPROM is broken.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
700	32	Servo Drive Fault	Parameter Checksum Error. The data of the parameter in the SERVOPACK is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
701	33	Servo Drive Fault	Parameter Format Error. The data format of the parameter in the SERVOPACK is incorrect	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
702	34	Servo Drive Fault	System Checksum Error. The data of the parameter in the SERVOPACK is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
703	48	Servo Drive Fault	Main Circuit Detector Error. Detection data for power circuit is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
704	64	Servo Drive Fault	Parameter Setting Error. The parameter setting is outside the allowable setting range.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
705	65	Servo Drive Fault	Encoder Output Pulse Setting Error. The encoder output pulse setting (pulse unit) (Pn212) is outside the allowable setting range or does not satisfy the setting conditions.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
706	66	Servo Drive Fault	Parameter Combination Error. Combination of some parameters exceeds the setting range.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

MLx Error #	OEM Error #	Message	Description	Remedy
707	68	Servo Drive Fault	Fully-closed Loop Control Parameter Setting Error. "The settings of the fully-closed option module and Pn00B.3, Pn002.3 do not match. "	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
708	74	Servo Drive Fault	Parameter Setting Error 2. There is an error in settings of parameters reserved by the system.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
709	80	Servo Drive Fault	Combination Error. The SERVOPACK and the servomotor capacities do not match each other.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
710	81	Servo Drive Fault	Unsupported Device Alarm. The unsupported device unit was connected.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
711	176	Servo Drive Fault	Canceled Servo ON Command Alarm. The host Control Module reference was sent to turn the Servo ON after the Servo ON function was used with the utility function.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
712	256	Servo Drive Fault	Overcurrent or Heat Sink Overheated. An overcurrent flowed through the IGBT. Heat sink of the SERVOPACK was overheated.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
713	768	Servo Drive Fault	Regeneration Error. Regenerative circuit or regenerative resistor is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
714	800	Servo Drive Fault	Regenerative Overload. Regenerative energy exceeds regenerative resistor capacity.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
715	816	Servo Drive Fault	Main Circuit Power Supply Wiring Error. Setting of AC input/DC input is incorrect. Power supply wiring is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
716	1024	Servo Drive Fault	Overvoltage. Main circuit DC voltage is excessively high.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
717	1040	Servo Drive Fault	Undervoltage. Main circuit DC voltage is excessively low.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

MLx Error #	OEM Error #	Message	Description	Remedy
718	1104	Servo Drive Fault	Main-Circuit Capacitor Overvoltage. The capacitor of the main circuit has deteriorated or is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
719	1296	Servo Drive Fault	Overspeed. The servomotor speed is over the maximum allowable speed.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
720	1297	Servo Drive Fault	Overspeed of Encoder Output Pulse Rate. The set value of the encoder output pulse (Pn212) exceeds the speed limit.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
721	1312	Servo Drive Fault	Vibration Alarm. Vibration at the motor speed was detected.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
722	1313	Servo Drive Fault	Autotuning Alarm. Vibration was detected while performing tuning-less function.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
723	1808	Servo Drive Fault	Overload: High Load. The motor was operating for several seconds to several tens of seconds under a torque largely exceeding ratings.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
724	1824	Servo Drive Fault	Overload: Low Load. The motor was operating continuously under a torque largely exceeding ratings.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
725	1840	Servo Drive Fault	Dynamic Brake Overload. "When the dynamic brake was applied, rotational energy exceeded the capacity of dynamic brake resistor. "	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
726	1841	Servo Drive Fault	Dynamic Brake Overload. "When the dynamic brake was applied, rotational energy exceeded the capacity of dynamic brake resistor. "	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
727	1856	Servo Drive Fault	Overload of Surge Current Limit Resistor. The main circuit power was frequently turned ON and OFF.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
728	1952	Servo Drive Fault	Heat Sink Overheated. The temperature of the SERVOPACK heat sink exceeded 100 deg C.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

MLx Error #	OEM Error #	Message	Description	Remedy
729	1963	Servo Drive Fault	Built-in Fan in SERVOPACK Stopped. The fan inside the SERVOPACK stopped.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
730	2064	Servo Drive Fault	Encoder Backup Error. All the power supplies for the absolute encoder have failed and position data was cleared.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
731	2080	Servo Drive Fault	Encoder Checksum Error. The checksum results of encoder memory is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
732	2096	Servo Drive Fault	Absolute Encoder Battery Error. The battery voltage is lower than the specified value after the control power supply is turned ON.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
733	2112	Servo Drive Fault	Encoder Data Error. Data in the encoder is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
734	2128	Servo Drive Fault	Encoder Overspeed. The encoder was rotating at high speed when the power was turned ON.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
735	2144	Servo Drive Fault	Encoder Overheated. The internal temperature of encoder is too high.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
736	2208	Servo Drive Fault	External Encoder Error. External encoder is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
737	2209	Servo Drive Fault	External Encoder Error of Module. Serial converter unit is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
738	2210	Servo Drive Fault	External Encoder Error of Sensor (Incremental). External encoder is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
739	2211	Servo Drive Fault	External Encoder Error of Position (Absolute). The external encoder position data is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

MLx Error #	OEM Error #	Message	Description	Remedy
740	2213	Servo Drive Fault	Encoder Overspeed. The overspeed from the external encoder occurred.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
741	2214	Servo Drive Fault	Encoder Overheated. The overheat from the external encoder occurred.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
742	2865	Servo Drive Fault	Current Detection Error1 (Phase-U). The current detection circuit for phase-U is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
743	2866	Servo Drive Fault	Current Detection Error 2 (Phase-V). The current detection circuit for phase-V is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
744	2867	Servo Drive Fault	Current Detection Error 3 (Current detector). The detection circuit for the current is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
745	3056	Servo Drive Fault	System Alarm 0. "Internal program error 0" "occurred in the SERVOPACK. "	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
746	3057	Servo Drive Fault	System Alarm 1. "Internal program error 1" "occurred in the SERVOPACK."	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
747	3058	Servo Drive Fault	System Alarm 2. "Internal program error 2" "occurred in the SERVOPACK."	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
748	3059	Servo Drive Fault	System Alarm 3. "Internal program error 3" "occurred in the SERVOPACK."	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
749	3060	Servo Drive Fault	System Alarm 4. "Internal program error 4" "occurred in the SERVOPACK."	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
750	3088	Servo Drive Fault	Servo Overrun Detected. The servomotor ran out of control.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

MLx Error #	OEM Error #	Message	Description	Remedy
751	3200	Servo Drive Fault	Absolute Encoder Clear Error and Multi-turn Limit Setting Error. The multi-turn for the absolute encoder was not properly cleared or set.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
752	3216	Servo Drive Fault	Encoder Communications Error. Communications between the SERVOPACK and the encoder is not possible.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
753	3217	Servo Drive Fault	Encoder Communications Position Data Error. An encoder position data calculation error occurred.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
754	3218	Servo Drive Fault	Encoder Communications Timer Error. An error occurs in the communications timer between the encoder and the SERVOPACK.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
755	3232	Servo Drive Fault	Encoder Parameter Error. Encoder parameters are faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
756	3248	Servo Drive Fault	Encoder Echoback Error. >Contents of communications with encoder is incorrect.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
757	3264	Servo Drive Fault	Multi-turn Limit Disagreement. Different multi-turn limits have been set in the encoder and the SERVOPACK.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
758	3313	Servo Drive Fault	Feedback Option Module Communications Error (Reception error). Reception from the feedback option module is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
759	3314	Servo Drive Fault	Feedback Option Module Communications Error (Timer stop). Timer for communications with the feedback option module is faulty.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
760	3328	Servo Drive Fault	Position Error Pulse Overflow. Position error pulses exceeded the value set for parameter (Pn520) (Excessive Position Error Alarm Level).	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
761	3329	Servo Drive Fault	Position Error Pulse Overflow Alarm at Servo ON. Position error pulses accumulated too much.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

MLx Error #	OEM Error #	Message	Description	Remedy
762	3330	Servo Drive Fault	Position Error Pulse Overflow Alarm by Speed Limit at Servo ON. After a position error pulse has been input, Pn529 limits the speed if the servo ON command is received. If Pn529 limits the speed in such a state, this alarm occurs when the position references are input and the number of position error pulses exceeds the value set for parameter Pn520 (Excessive Position Error Alarm Level).	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
763	3344	Servo Drive Fault	Motor-load Position Error Pulse Overflow. Position error between motor and load is excessive when fully-closed position control is used.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
764	3584	Servo Drive Fault	Command Option Module IF Initialization Timeout Error. Communications initialization failed between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
765	3586	Servo Drive Fault	Command Option Module IF Synchronization Error 1. A synchronization error occurred between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
766	3587	Servo Drive Fault	Command Option Module IF Communications Data Error. An error occurred in the data of communications between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
767	3648	Servo Drive Fault	Command Option Module IF Communications Setting Error. An error occurred in establishing communications (settings) between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
768	3664	Servo Drive Fault	Command Option Module IF Synchronization Error 2. A error occurred in synchronization between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
769	3665	Servo Drive Fault	Command Option Module IF Synchronization Establishment Error. A error occurred in establishing communications between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
780	3680	Servo Drive Fault	Command Option Module IF Data Communications Error. A error occurred in communications between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"

MLx Error #	OEM Error #	Message	Description	Remedy
781	3681	Servo Drive Fault	Command Option Module IF Synchronization Error 3. There was a change in timing of synchronization between the SERVOPACK and the command option module.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
782	3696	Servo Drive Fault	Command Option Module Detection Failure. Detection of the command option module failed.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
783	3697	Servo Drive Fault	Safety Option Module Detection Failure. Detection of the safety option module failed.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
784	3698	Servo Drive Fault	Feedback Option Module Detection Failure. Detection of the feedback option module failed.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
785	3699	Servo Drive Fault	Unsupported Command Option Module. An unsupported command option module was connected.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
786	3700	Servo Drive Fault	Unsupported Safety Option Module. An unsupported safety option module was connected.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
787	3701	Servo Drive Fault	Unsupported Feedback Option Module. An unsupported feedback option module was connected.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
788	3712	Servo Drive Fault	Command Option Module Unmatched Error. The command option module was replaced with a different model.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
789	2352	Servo Drive Warning	Absolute Encoder Battery Error. This warning occurs when the absolute encoder battery voltage is lowered.	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
790	2417	Servo Drive Warning	Undervoltage. "This warning occurs before Undervoltage (410h) alarm occurs. If the warning is ignored and operation continues, an under voltage alarm may occur."	"See Chapter 9 of Sigma-V Series, AC Servo Drives, USER'S MANUAL. Model: SGDV-OCA01A EtherCAT (CoE) Network Module. MANUAL NO. SIEP C720829 04A"
800	800	EtherCAT Software Error	Error details.	Consult MLxServo.log and MLxServoError.log for more information.

MLx Error #	OEM Error #	Message	Description	Remedy
801	801	Servo Drive Fault	Hardware Fault occurred in the SanMotionR Servo Drive.	See Chapter 11, section 3 of SanMotionR Advanced Model EtherCAT I/F Manual.
802	802	Servo Drive Fault	Hardware Fault occurred in the Servo Drive.	Refer to the Servo Drive reference manual for list of alarm codes and remedy information.
803	803	Digital IO Error	An error occurred setting or reading Digital IO.	Check and Verify Digital IO configuration.

Appendix D

D.1   3rd Party Software Licenses Usage

<div align="center">
SOFTWARE DISTRIBUTION LICENSE FOR THE
ETHERNET/IP(TM) COMMUNICATION STACK
(ADAPTED BSD STYLE LICENSE)
</div>

Copyright (c) 2009, Rockwell Automation, Inc. ALL RIGHTS RESERVED.
EtherNet/IP is a trademark of ODVA, Inc.

Redistribution of the Communications Stack Software for EtherNet/IP and use
in source and binary forms, with or without modification, are permitted
provided that the following conditions are met:

Redistributions of source code must retain the above copyright and trademark
notices, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

Redistributions in binary form must reproduce the above copyright notice, this
list of conditions and the following disclaimer in the documentation and/or
other materials provided with the distribution.

Neither the name of Rockwell Automation, ODVA, nor the names of its
contributors may be used to endorse or promote products derived from this
software without specific prior written permission from the respective owners.

The Communications Stack Software for EtherNet/IP, or any portion thereof,
with or without modifications, may be incorporated into products for sale.
However, the software does not, by itself, convey any right to make, have
made, use, import, offer to sell, sell, lease, market, or otherwise distribute or
dispose of any products that implement this software, which products might be
covered by valid patents or copyrights of ODVA, Inc., its members or other
licensors nor does this software result in any license to use the EtherNet/IP
mark owned by ODVA. To make, have made, use, import, offer to sell, sell,
lease, market, or otherwise distribute or dispose of any products that
implement this software, and to use the EtherNet/IP mark, one must obtain the
necessary license from ODVA through its Terms of Usage Agreement for the
EtherNet/IP technology, available through the ODVA web site at
www.odva.org. This license requirement applies equally (a) to devices that
completely implement ODVA's Final Specification for EtherNet/IP (?Network
Devices?), (b) to components of such Network Devices to the extent they
implement portions of the Final Specification for EtherNet/IP, and (c) to
enabling technology products, such as any other EtherNet/IP or other network
protocol stack designed for use in Network Devices to the extent they
implement portions of the Final Specification for EtherNet/IP. Persons or
entities who are not already licensed for the EtherNet/IP technology must
contact ODVA for a Terms of Usage Agreement.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE

MLX200
SOFTWARE AND OPERATION
USERS MANUAL

HEAD OFFICE
2-1 Kurosakishiroishi, Yahatanishi-ku, Kitakyushu 806-0004, Japan
Phone   +81-93-645-7703   Fax   +81-93-645-7802

YASKAWA America Inc. (Motoman Robotics Division)
100 Automation Way, Miamisburg, OH 45342, U.S.A.
Phone   +1-937-847-6200   Fax   +1-937-847-6277

YASKAWA Europe GmbH（Robotics Divsion）
Yaskawastrasse 1, 85391 Allershausen, Germany
Phone   +49-8166-90-100   Fax   +49-8166-90-103

YASKAWA Nordic AB
Bredbandet 1 vån. 3 varvsholmen 392 30 Kalmar, Sweden
Phone   +46-480-417-800   Fax   +46-480-417-999

YASKAWA Electric (China) Co., Ltd.
22/F One Corporate Avenue No.222, Hubin Road, Huangpu District, Shanghai 200021, China
Phone +86-21-5385-2200   Fax ＋86-21-5385-3299

YASKAWA SHOUGANG ROBOT Co. Ltd.
No7 Yongchang North Road, Beijing E&T Development Area, China 100176
Phone   +86-10-6788-2858   Fax   +86-10-6788-2878

YASKAWA India Private Ltd. (Robotics Division）
#426, Udyog Vihar, Phase- IV, Gurgaon, Haryana, India
Phone   +91-124-475-8500   Fax   +91-124-475-8542

YASKAWA Electric Korea Co., Ltd
9F, Kyobo Securities Bldg., 26-4, Yeouido-dong, Yeongdeungpo-gu, Seoul 150-737, Korea
Phone   +82-2-784-7844   Fax   +82-2-784-8495

YASKAWA Electric Taiwan Corporation
12F, No.207, Sec. 3, Beishin Rd., Shindian District, New Taipei City 23143, Taiwan
Phone   +886-2-8913-1333   Fax   +886-2-8913-1513

YASKAWA Electric (Singapore) PTE Ltd.
151 Lorong Chuan, #04-02A, New Tech Park, Singapore 556741
Phone   +65-6282-3003   Fax   +65-6289-3003

YASKAWA Electric (Thailand) Co., Ltd.
252/125-126 27th Floor, Tower B Muang Thai-Phatra Complex Building,
Rachadaphisek Road，Huaykwang, Bangkok 10320, Thailand
Phone   +66-2693-2200   Fax   +66-2693-4200

PT. YASKAWA Electric Indonesia
Secure Building-Gedung B Lantai Dasar & Lantai 1 JI. Raya Protokol Halim Perdanakusuma,
Jakarta 13610, Indonesia
Phone   +62-21-2982-6470   Fax   +62-21-2982-6741

Specifications are subject to change without notice
for ongoing product modifications and improvements.