Rockwell Logix5000 AOI for MAC and MIS –motors.

Description

The JVL AOI's will simplify the integration into the Rockwell Logix5000 environment.

The AOI's are designed to emulate the functionality as close to the existing motion commands in the Rockwell Kinetix world as possible.

For the Rockwell motion programmer the JVL AOI's will help the integration of the JVL motors into a Rockwell application.

This document along with the example project should give a good understanding on how to setup and control a JVL motor.

Downloading the packages

2 different packages are required to control the JVL motors, one MIS and one for the MAC –motors.

The packages can be downloaded following the link below:

https://www.jvl.dk/1189/addon-instructions

For MIS –Stepper motors

For MAC –Servo motors

Please observe that an EDS –file is used for the MAC –Servo motors, however this is not the case with the MIS –motors.

The EDS file must be imported into the Studio5000 Logix Designer environment.

Content

AOI for the MAC servo motors

Importing the EDS file into the Logix Designer (OBS! Only for MAC –motors)

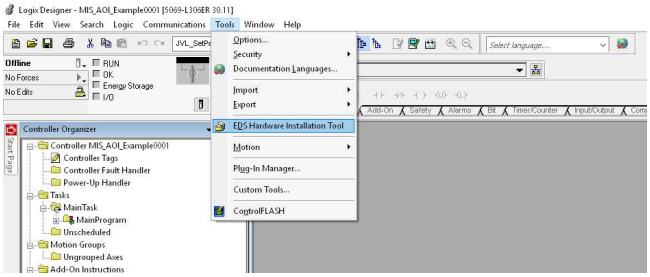
When AOI's for the MAC-motors are used, the EDS file is required.

The .eds file is included in the package with the AOI –files and the example project.

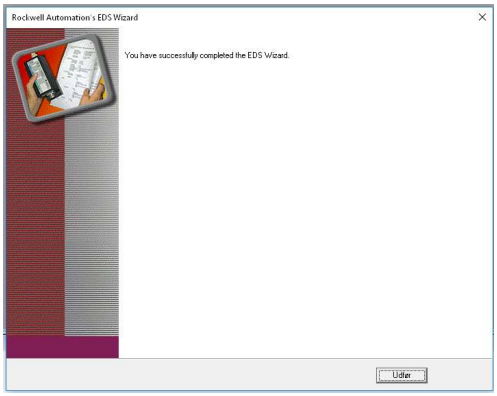
The EDS files holds a description of the different registers in the motor allowing a far more easy integration and predefined connection properties, the only parameter required for the connection is the IP address information.

Follow these steps to import the EDS -file into the Logix Designer.

1. Locate the "EDS Hardware Installation Tool"



2. Follow the guidelines and find the **JVL_MAC_ServoMotors.eds** –file.
3. Finally the EDS file should be successfully installed.

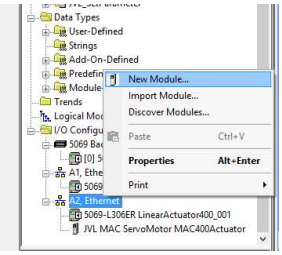


Adding the MAC-motor as a new module to the project

Now that the EDS file has been installed, the MAC motor will be recognized by the Logix Designer as a "JVL MAC ServoMotor".

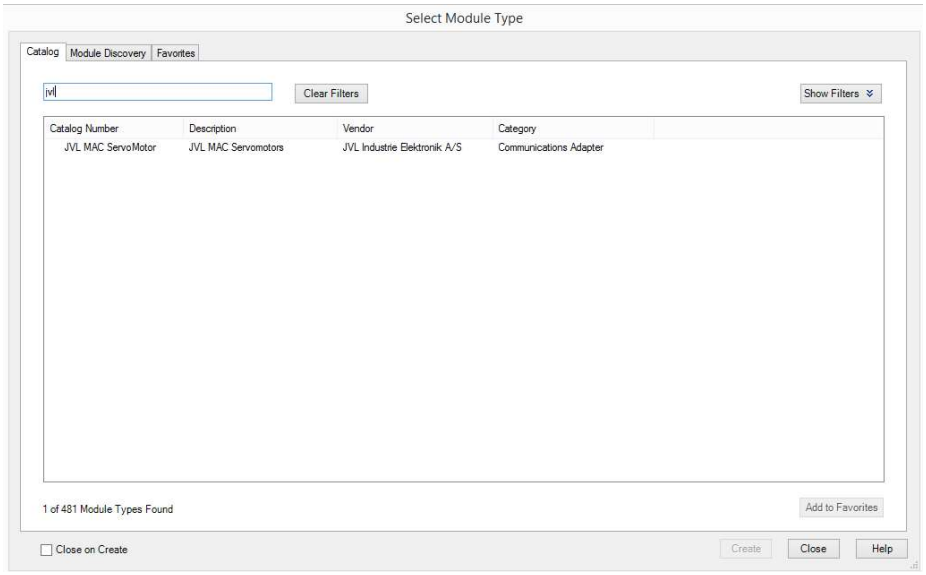
To add a motor to the project follow these steps.

1. Right click on the Ethernet Controller and select "New module"

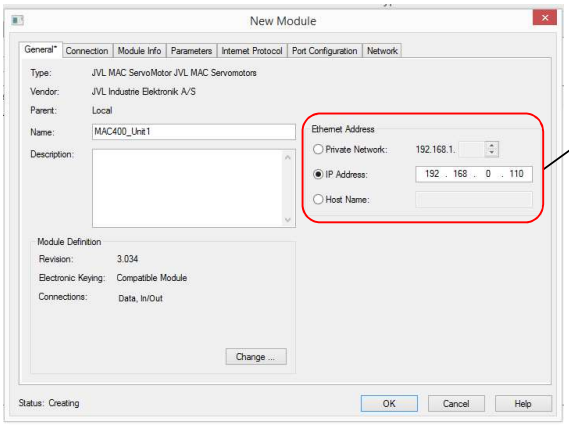


2. Type in "JVL" in the search field and the "JVL MAC ServoMotor" appears.

Select the motor type and press "Create"



3. Name the motor for the project and enter the IP address.



The IP address will follow the motor settings.

The default IP address is always “192.168.0.XXX” where XX is the decimal number of the last byte from the MAC-address. The Mac address is visible from a label placed on the motor.

Exc.

If an MAC address has the number:

54:E3:B0:00:51:**B5**

Then the last byte in the MAC address is HEX B5 (0xB5) which is equivalent to 181 in decimal.

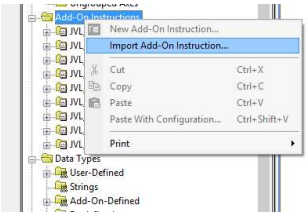
So the stock default IP address of the motor will be **192.168.0.181**

Now the motor has been added to the project.

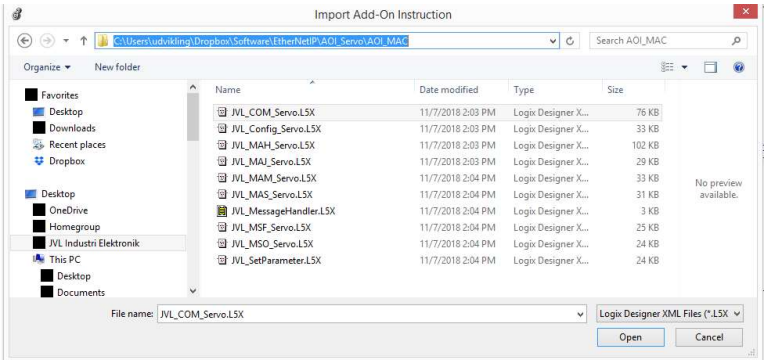
Please note that the motor needs configuration before the I/O connection will be ok. This configuration of the motor is done from the JVL_Comms –AOI described later.

Importing the JVL AOI's into the project.

1. Import the JVL AOI -set into the project by right click on the "Add on instructions"

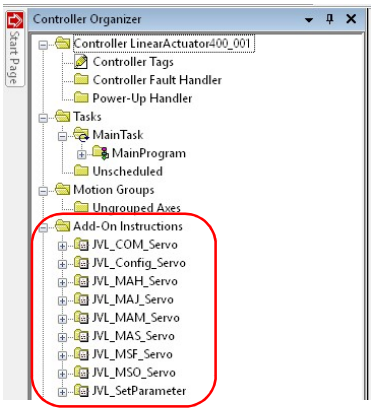


4. Select the JVL AOI from the list and press "Open".



Unfortunately it is not possible to select more than one AOI in the dialog, so each AOI needs to be loaded one at a time.

When all AOI's are loaded the tree structure of the "Add on instructions" should appear as follows:

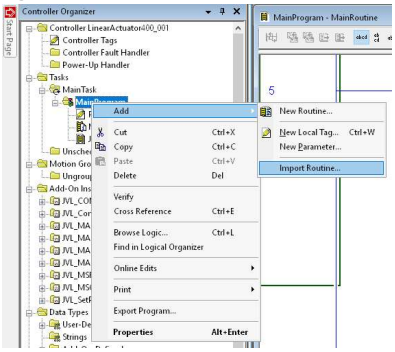


Finally the JVL_MessageHandler routine needs to be added to the project.

This routine handles all explicit messages that are used for configuration and special functions.

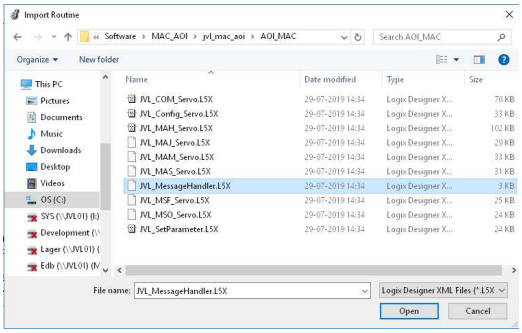
The message handler routine is also included in the package and it is imported into the project following these guidelines:

1. Right click on the **Task->MainTask->MainRoutine**



Select **Add->Import Routine**

Select the "JVL_MessageHandler.L5X" –file from the list and press open:



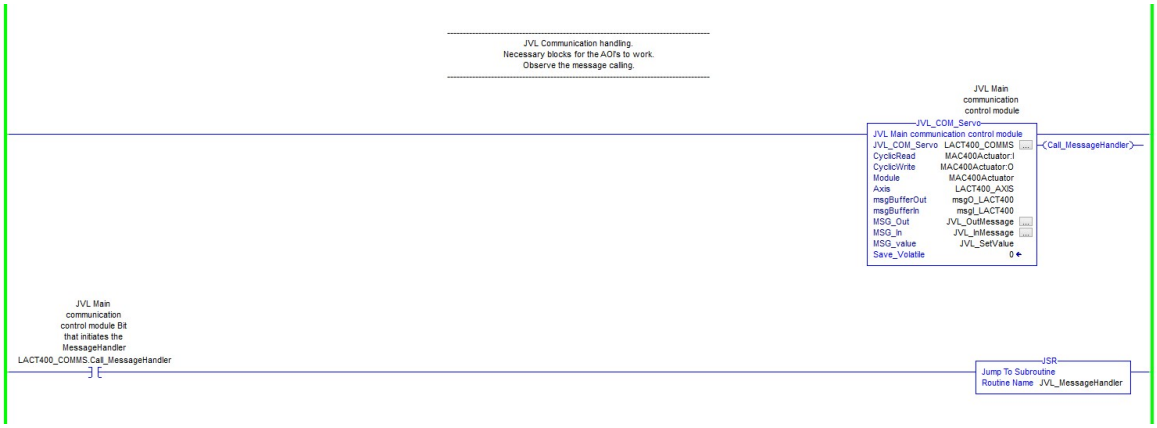
AOI Description

JVL_COM_Servo

Since the AOI's works by using a combination of cyclic data exchange and explicit messages, the JVL_COM_Servo instruction must be called continuously. This AOI handles scaling and basic operation of the motor and ensures that the right data is setup and copied into the cyclic data exchange handled by the I/O.

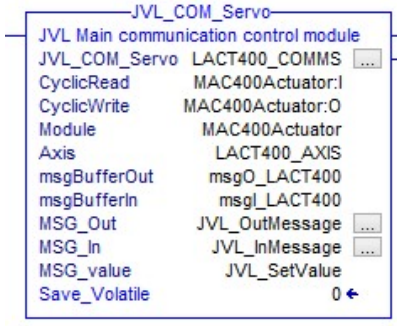
The separate routine JVL_MessageHandler previously included in the project, is called on request from the CALL_MessageHandler -bit in the JVL_COM_Servo -instance.

The basic concept is displayed below in the 2 rungs taken from the JVL example.



Whenever the "Call_MessageHandler" –bit controlled from the JVL_Com_Servo instance goes high the JVL_MessageHandler is called.

Parameters:



JVL_COM_Servo:	The instance name, exc. "LACT400_COMMS"
CyclicRead:	Pointer to the Cyclic read data of the MAC-motor. Exc. MAC400Actuator is the name of motor in the project, so we point on the **MAC400Actuator:I**
CyclicWrite:	Pointer to the Cyclic read data of the MAC-motor. Similar to CyclicRead, but points to the Output –data, exc. **MAC400Actuator:O**
Module:	Motor exc. MAC400Actuator.
Axis:	Axis –data object, Create an object with a name and finalize with CTRL + W. This object will be used in other AOI's ass well.
msgBufferOut:	Buffer that holds message data. Create an instance with CTRL+W.
msgBufferIn:	Buffer that holds message data. Create an instance with CTRL+W.
MSG_Out:	MSG instance create with CTRL+W.
MSG_In:	MSG instance create with CTRL+W.
MSG_Value:	MSG value -instance create with CTRL+W.
Save_Volatile:	Input bit required to configure the motor and save the settings in volatile memory in the motor. This is only required once with a factory configured motor.

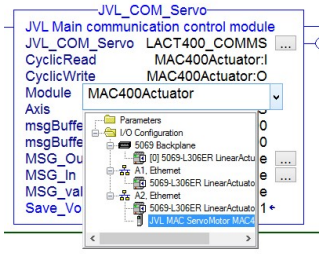
Tip:

Since most of the input operands are structures, for convenience they can be created using

CTRL + W for automatically adding the tag to the controller tag list.

Invent logical names to the tags, some of them will be used elsewhere.

For the “Module” -operand select the motor added to the ethernet controller from the list.



In case the motor holds factory default settings the PLC will flag an I/O Connection error and I/O LED will blink. This is completely normal. Simply run the JVL_CONFIG_Servo AOI and set the JVL_COM_Servo.Initialize –bit = 1 as demonstrated below. Since the “Save_Volatile” bit is set In the JVL_COM_SERVO instance, settings will be saved in flash memory. After the motor resets the PLC should automatically establish cyclic communication with the motor and the I/O status should recover from the fault condition.

JVL_Config_Servo

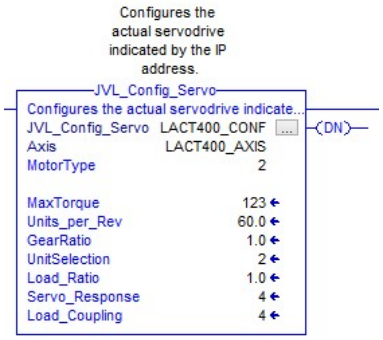
The JVL_Config_Servo is used to setup the application regarding scaling, servo filter settings and motor type used.

Using this function will configure the Axis -tag with the appropriate values that is transferred to the motor and used under special circumstances like during a homing sequence etc.

When the Initialize -bit is set in the JVL_COM_Servo instance, it will transfer the necessary data to the motor and setup the basic EthernetIP -settings.

This should only be necessary to do once, then the EthernetIP -settings and the settings in the motor as well as the Axis setting should be in place and after the next power cycle / reset all should be configured.

Parameters:



JVL_Config_Servo:	The instance name, exc. “LACT400_CONF”
Axis:	Axis –data object, Create an object with a name and finalize with CTRL + W. This object will be used in other AOI’s ass well.
MotorType:	Select which motortype used. 0: No Selction 1: MAC50-141 2: MAC050G – MAC141G (New generation) MAC400 - MAC4500, MAC800 w. abs. encoder (8192/rev). 3: MAC800 w. incremental encoder (8000/rev)
MaxTorque:	Max. allowed torque 0-300 [%]
Units_per_rev:	Units per. Revolution. This operand is used for scaling purposes. Exc. An actuator has a linear movement of 60mm / rev. then 60.0 is entered.
GearRatio:	If a gearbox is mounted the ratio is entered here n:1
UnitSelection:	With this selection all motion parameters will be scaled according to these settings. 0: None. 1: [um] and [um/s] 2: [mm] and [mm/s] 3: [m] and [m/s]

Servo internal filter settings

Load_Ratio:	Inertia ratio between the motor and load. N:1
Servo_Response:	Filter response 0=slow, 4=fast.
Load_Coupling:	Rigidity of the servo filter, 0=Flexible, 4=Rigid

JVL_SetParameter

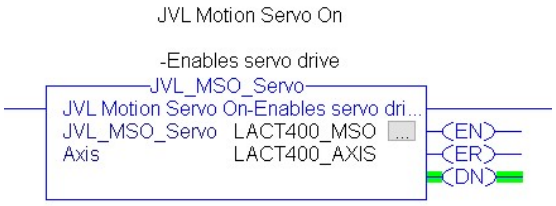
This AOI is mainly supporting the AOI's when a specific register need to be configured. It is not intended for custom use.

JVL_MSO (MotionServoOn)

This instruction is based on the Rockwell command MSO, which basically energizes the servo drive. In JVL terms it will put the motor into Position mode. This instruction is required to be called with success prior to other motion instructions to be issued.

Executes on rising edge.

Parameters:



JVL_MSO_Servo:	The instance name, exc. "LACT400_MSO"
Axis:	The Axis object of the motor. The same as the Axis object used in the JVL_COM_Servo instance.
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 1 = Motor has a fault 4 = Motor is already enabled.
Output DN:	Instuction has completed successfully

JVL_MSF (MotionServoOff)

This instruction is based on the Rockwell command MSF, which disables the servo drive and makes it passive. Prior of calling the instruction the JVL_MSO must successfully have been called otherwise an error will be flagged.

Executes on rising edge.

Parameters:

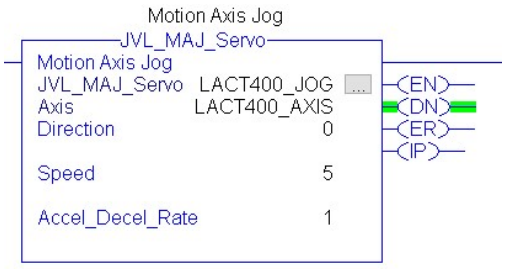
JVL_MSF_Servo:	The instance name, exc. "LACT400_MSF"
Axis:	The Axis object of the motor. The same as the Axis object used in the JVL_COM_Servo instance.
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled.
Output DN:	Instuction has completed successfully

JVL_MAJ (MotionAxisJog)

This instruction is based on the Rockwell command MAJ [MotionAxisJog], which starts jogging the servo drive in a certain direction with a certain speed. The jogging is stopped with the command JVL_MAS. This instruction basically puts the motor in velocity mode using JVL terms.

Executes on rising edge.

Parameters:



JVL_MAJ_Servo:	The instance name, exc. "LACT400_MAJ"
Axis:	The Axis object of the motor. The same as the Axis object used in the JVL_COM_Servo instance.
Direction:	Direction of motion: 0: CW 1: CCW
Speed:	Velocity in the selected unit.
Accell_Decel_rate:	Acceleration and deceleration rate in the selected unit.
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. 13 = an extended errorcode is issued EXERR: 2= Direction is not 0 or 1 3= Speed < 0 (negative value entered) 4= Accel_Decel_Rate < 0 (negative value)
Output DN:	Instuction has completed successfully
Output IP:	Motion is in progress. This will be held high until a MAS instruction is issued.

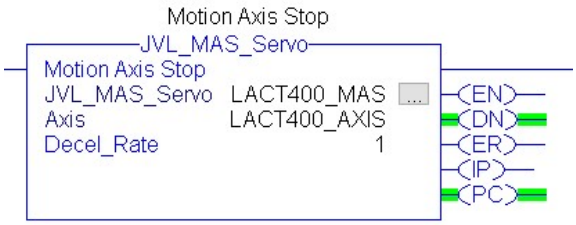
JVL_MAS (MotionAxisStop)

This instruction is based on the Rockwell command MAS [MotionAxisStop], which stops any pending motion.

In case a JVL_MAS instruction has been issued, the motors runs until the JVL_MAS instruction is issued.

Executes on rising edge.

Parameters:



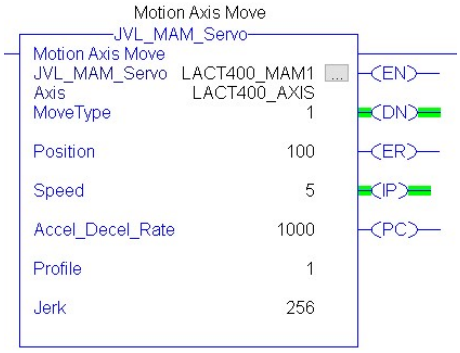
JVL_MAS_Servo:	The instance name, exc. "LACT400_MAS"
Axis:	The Axis object of the motor. The same as the Axis object used in the JVL_COM_Servo instance.
Accell_Decel_rate:	Acceleration and deceleration rate in the selected unit.
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. 13 = an extended errorcode is issued EXERR: 2= Accel_Decel_Rate < 0 (negative value)
Output DN:	Instuction has completed successfully
Output IP:	In progress.
Output PC:	Process completed.

JVL_MAM (MotionAxisMove)

This instruction is based on the Rockwell command MAM [MotionAxisMove], which starts a move to a certain position either absolute or incremental with a

Executes on rising edge.

Parameters:



JVL_MAM_Servo:	The instance name, exc. "LACT400_MAM"
Axis:	The Axis object of the motor. The same as the Axis object used in the JVL_COM_Servo instance.
MoveType:	Absolute (=0) or Relative (=1)
Position:	Position in the selected unit.
Speed:	Speed used in the selected unit
Accell_Decel_rate:	Acceleration and deceleration rate in the selected unit.
Profile:	0=Trapezoid, 1=S-Curve
Jerk:	S-Curve value in case Profile = 1. Values: [0-256]
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. 13 = an extended errorcode is issued EXERR: 2= MoveType out of range 3= Position out of range (max/min signed 32bit) 4= Speed < 0 (negative value) 5= Accell_Decel_Rate < 0 (negative value)
Output DN:	Instuction has completed successfully
Output IP:	In progress.
Output PC:	Process completed.

In the example above the servo will move 100 units relative to the current position. It accelerate to speed of 5 units with acceleration setting 1000 unit/s. The setting has enabled S-Curve acceleration and the jerk is set to the max. value of 256, which is the max. S-Curve value for a JVL servo motor.

High settings of accelerations or decelerations can be hard on the mechanic construction and lead to premature failure and wearing.

The S-Curve acceleration and deceleration method is often used in applications where less wear and tear is desirable due to the limitation in the dV/dT factor of the acceleration.

JVL_MAH (MotionAxisHome)

This instruction is based on the Rockwell command MAH [MotionAxisHoming], which starts a homing sequence.

The basic ideer is that the motor either runs a sensor, Torque based or Encoder index -based homing sequence.

A sensor based homing triggers on the sensor input (AIN, see module specs for details) when it transitions either from high to low or low to high. The position is then preset to the value indicated in the “Position” –parameter.

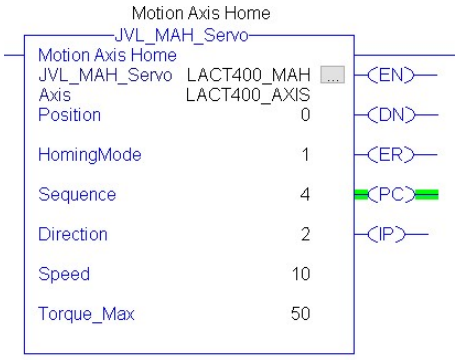
The Torque homing method is less accurate due to tolerances in the mechanical construction and changes due to tear and wear. However when the motor detects an applied torque setting above the indicated in the Torque_Max –parameter it will trigger and set the position to the indicated.

The last method triggers in the internal encoder index pulse and will thus be accurate within one revolution of the motor shaft. This method is very accurate however only usable within one shaft revolution.

Both the torque and the index –method eliminates the need for an external sensor.

Executes on rising edge.

Parameters:



JVL_MAH_Servo:	The instance name, exc. “LACT400_MAH”
Axis:	The Axis object of the motor. The same as the Axis object used in the JVL_COM_Servo instance.
Position:	The reference position to set in the homing position.
HomingMode:	0= Passive, 1= Active. Only Active (=1) supported.
Sequence:	Selects the method used for the homing process along with the Direction parameter. See the table for further details on which homing method to use and how to setup the Sequence and Direction parameters.
Direction:	With the Sequence parameter, Direction selects the homing method used.
Speed:	Speed used during the homing process.
Torque_Max:	Trip point at which Torque homing is triggered.

Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. 13 = an extended errorcode is issued EXERR: 3= Position out of range (max/min signed 32bit) 5= Sequence is beyond range [1..4] 7= For sequence = 1 or 2, legal range for the direction is [1..4]. 8= Speed must be > 0. 10= For Sequence=4 (Torque) the Torque_Max must be > 0. 100= Error state is present in motor.
Output DN:	Instuction has completed successfully
Output IP:	In progress.
Output PC:	Process completed.

Table of homing possibilities

Sequence	Direction	Description
1	0	Forward homing to switch, Uni direction. Stops on switch rising edge.
1	1	Forward homing to switch, Bi direction. Detects switch on rising edge and returns to find falling edge. This is a more accurate method than the Uni direction –method.
1	2	Reverse homing to switch, Uni direction. Stops on switch rising edge.
1	3	Reverse homing to switch, Bi direction. Detects switch on rising edge and returns to find falling edge. This is a more accurate method than the Uni direction –method.
2	0	Slow Forward home to marker (internal index pulse). Very accurate. Only works within one revolution of the motor shaft.
2	1	Fast Forward home to marker (internal index pulse). Less accurate. Only works within one revolution of the motor shaft.
2	2	Slow Reverse home to marker (internal index pulse). Very accurate. Only works within one revolution of the motor shaft.
2	3	Fast Reverse home to marker (internal index pulse). Less accurate. Only works within one revolution of the motor shaft.
4	0	Forward Torque homing. Trips when the actual torque > Torque_Max [%]
4	2	Reverse Torque homing. Trips when the actual torque > Torque_Max [%]

Please observe that for MAC50-141 –servo motors the Home to marker methods are not available.

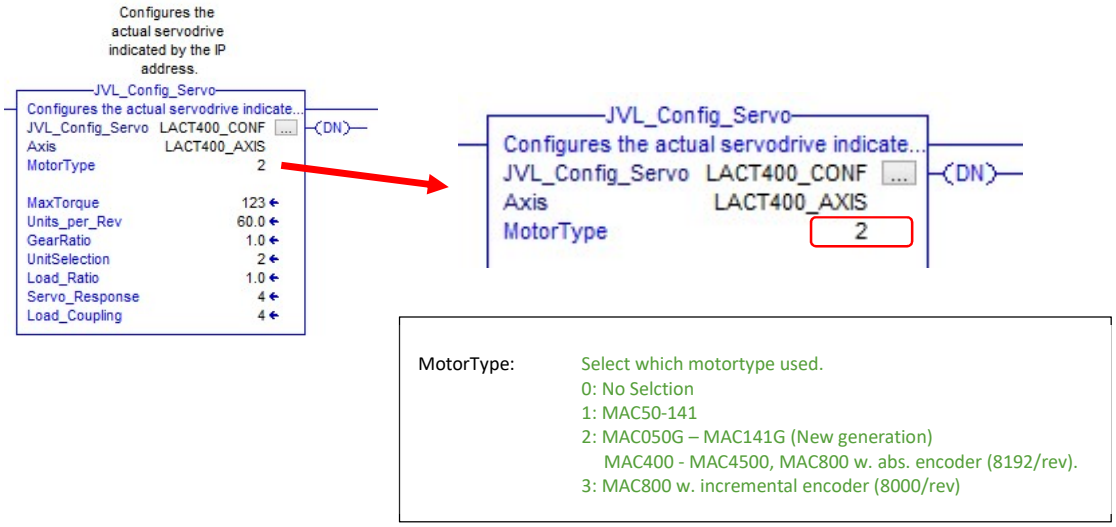
Supplied MAC AOI -Example

The example supplied shows a demo of all the JVL AOI's in a typical application using a linear actuator. When the example is loaded, all JVL AOI's are loaded automatically so there is no need for importing all the AOI's prior to loading the example.

In the example we use the metric units of [mm] and [mm/s] and limits the Max. Torque allowed to 123%.

Observe that the motor is capable of delivering 300% of the rated torque for a short period.

The example can be used to any JVL servomotor, just change the **LACT400_CONF.Motortype** to an appropriate value for the motor type used.



The Config is called with the following parameters:

MotorType = 2 (MAC400)

MaxTorque = 123% (Peak is 300%)

units_per_Rev = 60.0 which is this case means 60.0mm/rev of the motorshaft. This is the linear spindle ratio.

GearRatio = 1.0, means 1:1 ratio, we dont have any gearbox attached.

UnitSelection = 2 means we use [mm] and [mm/s] in this applicaiton, Then all motion instruction will use these units in the motion profile settings.

Internal filter settings in the motor, these will vary with the application.

Load_Ratio = 1.0, The motor will face an intertia load ratio 1:1, this value is used internally in the motor for the motion filter settings.

Servo_Response = 4 Means we request a fast acting servo. The value goes from 1 to 4, where 4 = fast.

Load_Coupling = 4, we request a rigid load coupling, value goes from 1 to 4.

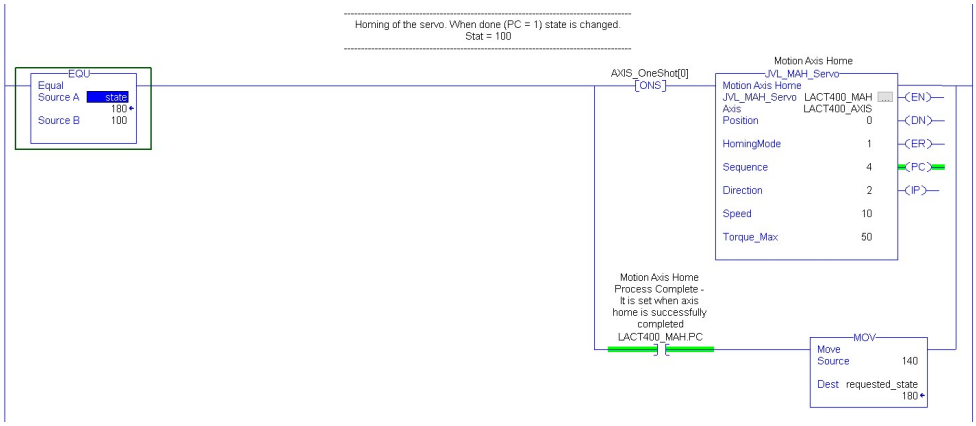
Please observe that a combination of high Load ratio and high servo response setting with high load coupling setting, could result in an unstable servo motor, use with care.

In the example the JVL_Config_Servo will be called once at the first PLC cycle. For a factory defaulted motor, please set the Save_Volatile (from the JVL_Com_Servo -object) –flag to 1 and run the process again.

This should only be necessary the first time, then the motor should have been configured and all settings saved in volatile memory.

The program is controlled through a statemachine entering different states along with the process of the different AOI has completed the motion.

When the **START** –bit is set, the servo is energized and enabled through the JVL_MSO –instruction and starts a homing procedure JVL_MAH in the requested_state = 100.

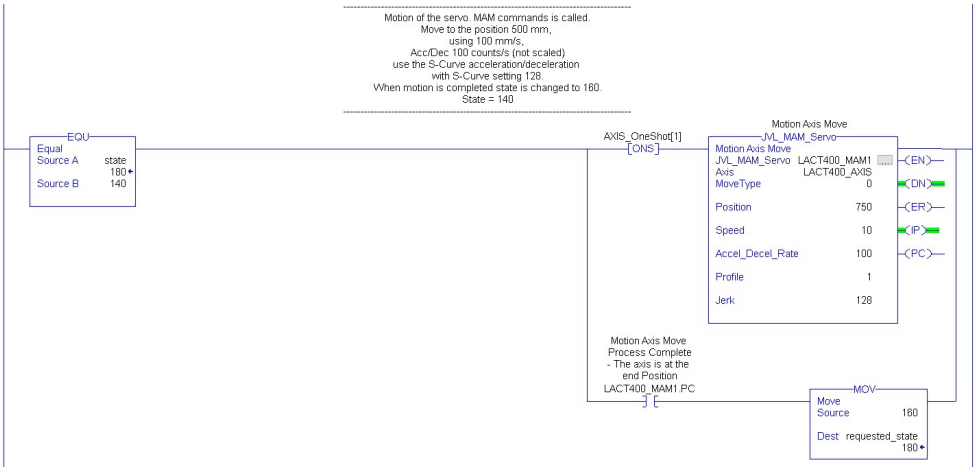


The homing procedure is configured for a torque reverse homing (Sequence = 4, Direction = 2) with a trip point of 50% torque. In other words the motor rotates CCW with the speed of 10mm/s to the mechanical end stop where it will trip on an applied torque exceeding 50%. This position is defined as Position = 0.

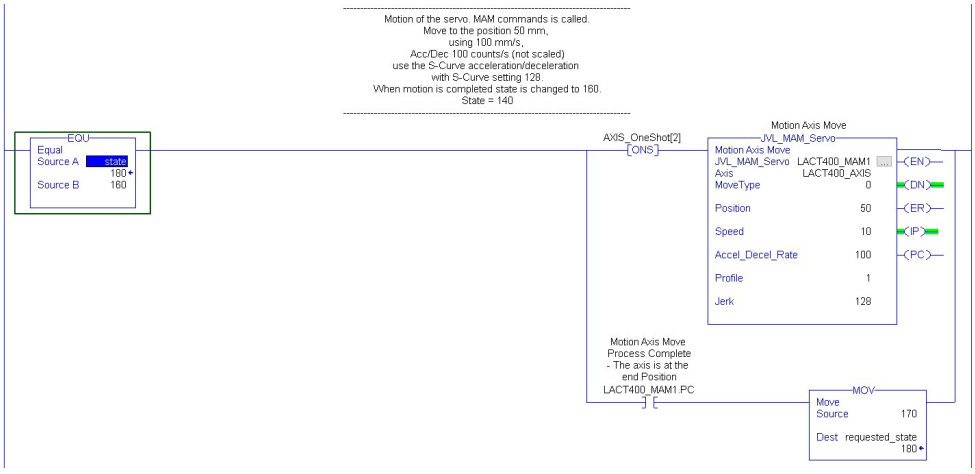
Note: We assume the motor is a basic MAC400 D2/5 –CAGM, supplied with a standard incremental encoder.

When the homing is done the requested_state changes to 140 where the JVL_MAM instruction is called.

The MAM instruction moves the motor to the absolute position 750mm, running 10mm/s, acceleration is 100 mm/s/s. The Profile used (Profile = 1) means the it will control the Acceleration and deceleration with S-Curve settings, using the Jerk value of 128 (Max. is 256).

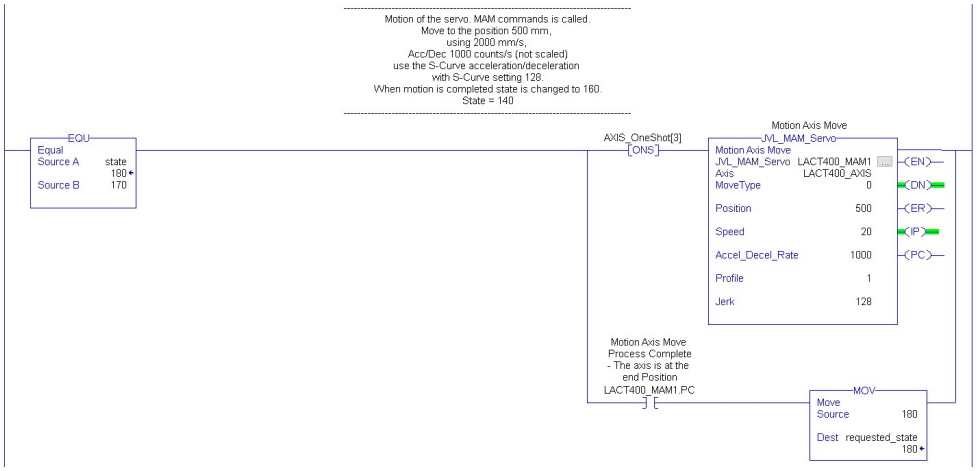


When the motor reaches the position 750mm the MAM instruction flags the .PC –bit and the requested_state = 160.

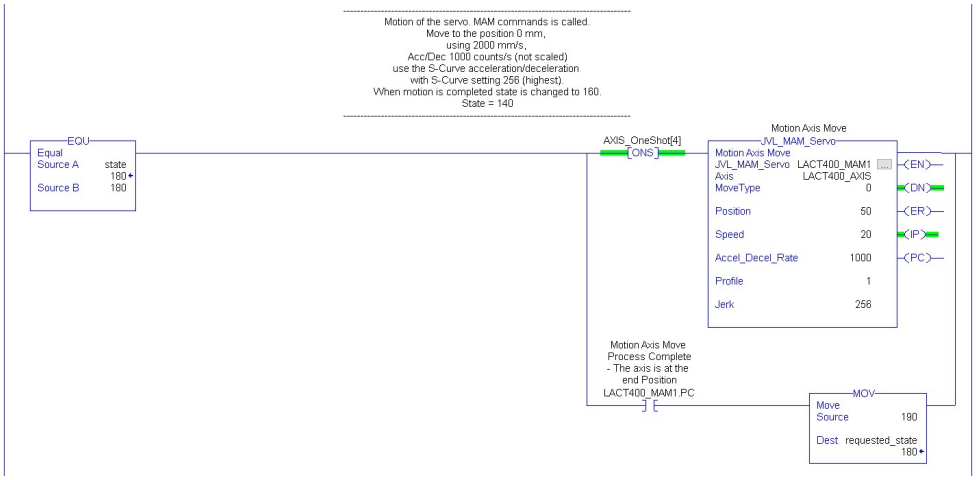


In this state the motor moves to absolute position 50mm using the same parameters as in the previous state.

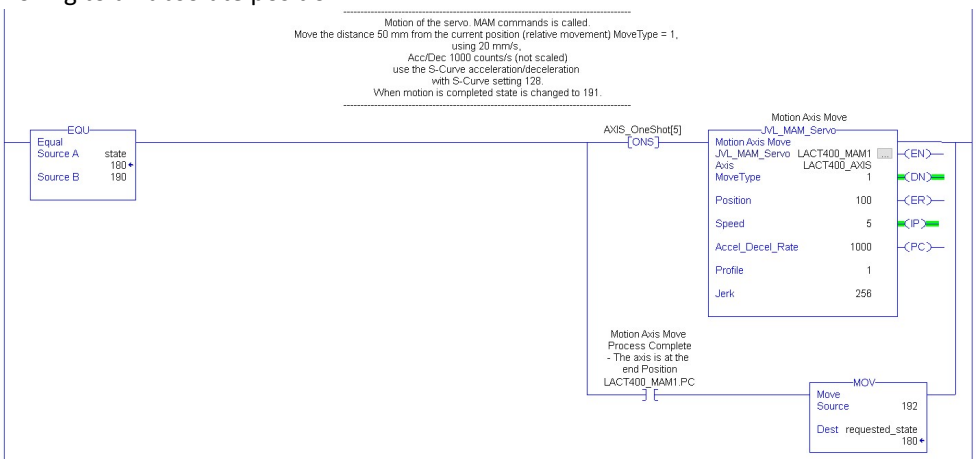
Next state, requested_state = 180, the motor moves to the position 500mm, speed = 20 mm/s, acceleration = 1000, still using S-Curve,



Next state is similar, the position is set to 50mm

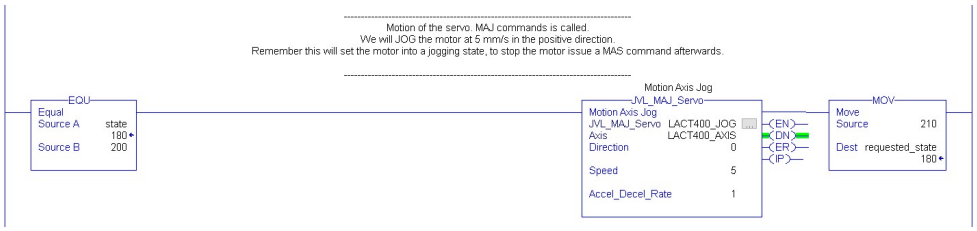


In the next state the MoveType has changed, so the motor will move relative to the current position instead of moving to an absolute position.



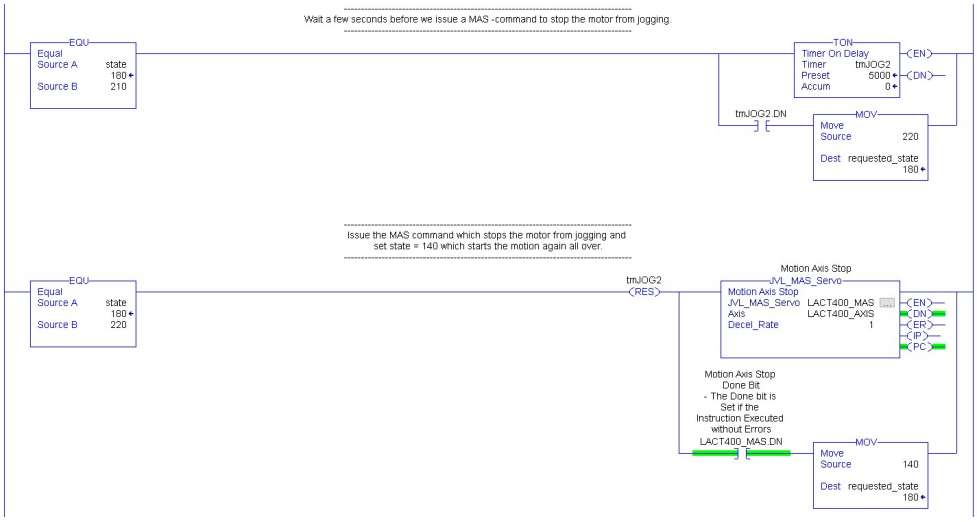
The motor will in this case move 100mm relative to the current position. In case the position was set to -100mm the motor would move 100mm in the opposite direction.

The next state, requested_state = 200, the JVL_MAJ AOI is demonstrated.



The motor will JOG forward, Speed is 5mm/s, Acceleration = 1. Observe that the MAJ (MotionAxisJog) needs to be stopped by issuing a MAS instruction.

In the example a 5000ms timer controls when the MAS instruction is issued. Afterwards it will return to the first motion state again requested_state = 140.



AOI for the MIS stepper motors

The AOI's for the MIS stepper series doesn't require an .EDS file in contrary to the MAC motors.

Since the MIS AOI's isn't based on an .EDS description there are some additional configuration AOI's that needs be called for configuration of a factory defaulted motor.

The supplied example goes through all of the necessary steps to get everything working, but a small step by step procedure is also supplies later in this manual, covering configuration from MacTalk.

The AOI's are very similar to the AOI's used in the MAC motors, however due to the differences in motor technology there are deviations between the AOI's.

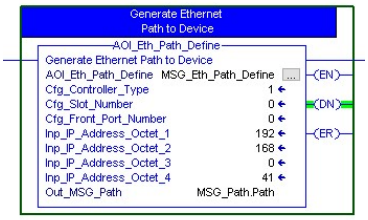
Setting up the project, the first steps

Message paths

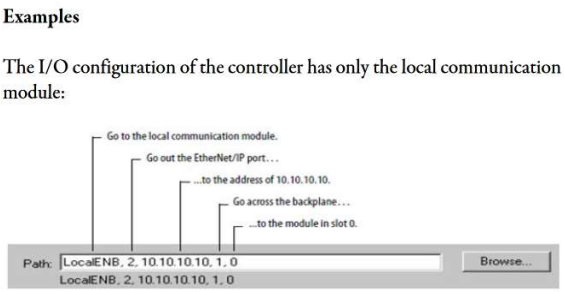
When a motor is added to the project the IP address needs to be added to the **AOI_Eth_Path_Define** –instruction in the **_02_Define_MSG_Path**.

This is essential for the configuration of a new motor.

The JVL_MAH homing instruction also needs a separate message object passed and the message path needs to be defined.



This configuration is for a CompactLogix 5380, Configured with A1/A2 Dual IP.



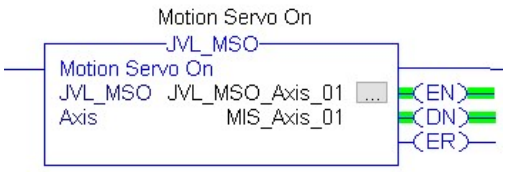
AOI Description

JVL_MSO (MotionServoOn)

This instruction is based on the Rockwell command MSO, which basically energizes the stepper drive. In JVL terms it will put the motor into Position mode. This instruction is required to be called with success prior to other motion instructions to be called.

Executes on rising edge.

Parameters:



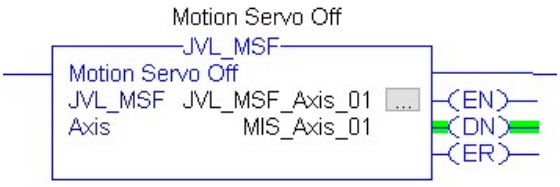
JVL_MSO:	The instance name, exc. "JVL_MSO_Axis_01"
Axis:	The Axis object of the motor.
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 4 = Motor is already enabled.
Output DN:	Instuction has completed successfully

JVL_MSF (MotionServoOff)

This instruction is based on the Rockwell command MSF, which disables the stepper drive and makes it passive. Prior of calling this instruction the JVL_MSO must successfully have been called otherwise an error will be flagged.

Executes on rising edge.

Parameters:



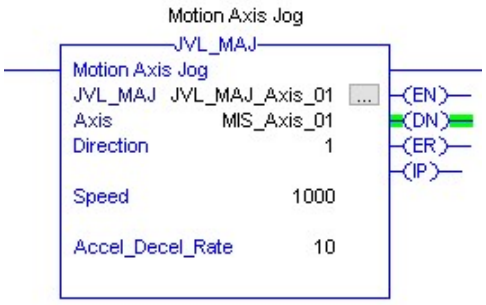
JVL_MSF:	The instance name, exc. "JVL_MSF_Axis_01"
Axis:	The Axis object of the motor.
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. Issue MSO first.
Output DN:	Instuction has completed successfully

JVL_MAJ (MotionAxisJog)

This instruction is based on the Rockwell command MAJ [MotionAxisJog], which starts jogging the servo drive in a certain direction with a certain speed. The jogging is stopped with the command JVL_MAS. This instruction basically puts the motor in velocity mode using JVL terms.

Executes on rising edge.

Parameters:



JVL_MAJ:	The instance name, exc. "JVL_MAJ_Axis01"
Axis:	The Axis object of the motor.
Direction:	Direction of motion: 0: CW (Forward) 1: CCW
Speed:	Velocity in the selected unit.
Accell_Decel_rate:	Acceleration and Deceleration in native motor units [Cnt's/Sample²]
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled (JVL_MSO). 13 = an extended errorcode is issued EXERR: 2= Direction is not 0 or 1 3= Speed < 0 (negative value entered) 4= Accel_Decel_Rate < 0 (negative value)
Output DN:	Instuction has completed successfully
Output IP:	Motion is in progress. This will be held high until a MAS instruction is issued.

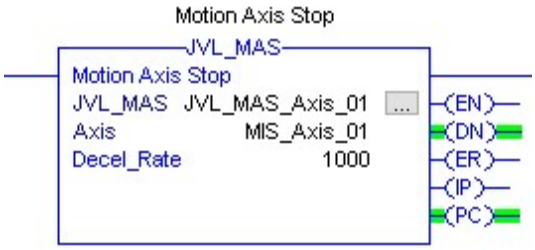
JVL_MAS (MotionAxisStop)

This instruction is based on the Rockwell command MAS [MotionAxisStop], which stops any pending motion.

In case a JVL_MAS instruction has been issued, the motors runs until the JVL_MAS instruction is issued.

Executes on rising edge.

Parameters:



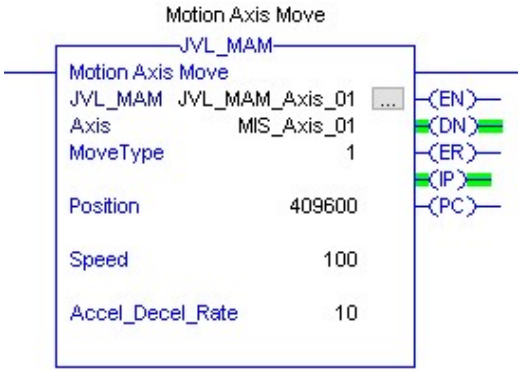
JVL_MAS:	The instance name, exc. "JVL_MAS_Axis_01"
Axis:	The Axis object of the motor.
Accell_Decel_rate:	Acceleration and deceleration rate in the selected unit.
Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. 13 = an extended errorcode is issued EXERR: 2= Accel_Decel_Rate < 0 (negative value)
Output DN:	Instuction has completed successfully
Output IP:	In progress.
Output PC:	Process completed.

JVL_MAM (MotionAxisMove)

This instruction is based on the Rockwell command MAM [MotionAxisMove], which starts a move to a certain position either absolute or incremental with a

Executes on rising edge.

Parameters:



JVL_MAM_Servo:	The instance name, exc. "JVL_MAM_Axis_01"
Axis:	The Axis object of the motor.
MoveType:	Absolute (=0) or Relative (=1)
Position:	Position in the selected unit.
Speed:	Speed used in the selected unit
Accell_Decel_rate:	Acceleration and deceleration rate in the selected unit.

Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. 13 = an extended errorcode is issued EXERR: 2= MoveType out of range 3= Position out of range (max/min signed 32bit) 4= Speed < 0 (negative value) 5= Accell_Decel_Rate < 0 (negative value)
Output DN:	Instuction has completed successfully
Output IP:	In progress.
Output PC:	Process completed.

In the example above the servo will move 409600 Units relative to the current position. It accelerate to speed of 100 units with acceleration setting 10 unit/s.

JVL_MAH (MotionAxisHome)

This instruction is based on the Rockwell command MAH [MotionAxisHoming], which starts a homing sequence.

The basic ideer is that the motor either runs a sensor, Torque based or Encoder index -based homing sequence.

A sensor based homing triggers on the sensor input (AIN, see module specs for details) when it transitions either from high to low or low to high. The position is then preset to the value indicated in the “Position” –parameter.

The Torque homing method is less accurate due to tolerances in the mechanical construction and changes due to tear and wear. However when the motor detects an applied torque setting above the indicated in the Torque_Max – parameter it will trigger and set the position to the indicated.

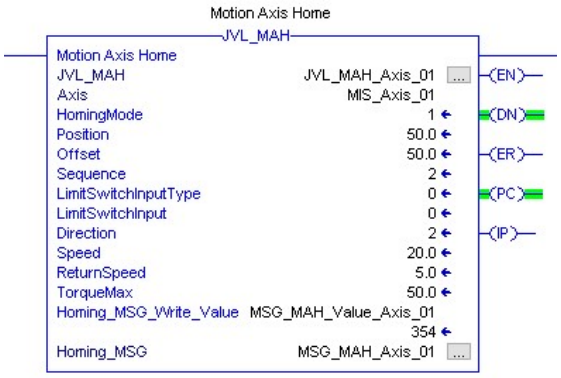
The last method triggers in the internal encoder index pulse and will thus be accurate within one revolution of the motor shaft. This method is very accurate however only usable within one shaft revolution.

Both the torque and the index –method eliminates the need for an external sensor.

The JVL_MAM instruction must be installed with the JVL_MAH instruction.

Executes on rising edge.

Parameters:



JVL_MAH_Servo:	The instance name, exc. "JVL_MAH_Axis_01"
Axis:	The Axis object of the motor
Position:	The reference position to set in the homing position.
HomingMode:	0= Passive, 1= Active.
Sequence:	Selects the method used for the homing process along with the Direction parameter. See the table for further details on which homing method to use and how to setup the Sequence and Direction parameters.
LimitSwitchInputType:	Normally Open = 0, Normally closed = 1.
LimitSwitchInput:	BOOL tag that must be set with the homing input. **Currently only a PLC input tag. Unless the motor inputs are mapped into an Cyclic write word.**
Direction:	With the Sequence parameter, Direction selects the homing method used.
Speed:	Speed used during the homing process.
Torque_Max:	Trip point at which Torque homing is triggered.
Homing_MSG_Write_Value:	DINT tag that holds a value for the Homing_MSG to send.
Homing_MSG:	Pointer to the homing message defined in the Controller tags –section.

Output EN:	Rung enable.
Output ER:	An error has occurred, read the instance parameter ERR for an errorcode. 5 = Motor is not enabled. 13 = an extended errorcode is issued EXERR: 3= Position out of range (max/min signed 32bit) 5= Sequence is beyond range [1..4] 7= For sequence = 1 or 2, legal range for the direction is [1..4]. 8= Speed must be > 0. 10= For Sequence=4 (Torque) the Torque_Max must be > 0. 100= Error state is present in motor.
Output DN:	Instuction has completed successfully
Output IP:	In progress.
Output PC:	Process completed.

Table of homing possibilities, Active = 1

Sequence	Direction	Description
0	X	Resets encoder position, without motor shaft turning, same as Active = 0.
1	0	Forward homing to switch, Uni direction. Stops on switch rising edge.
1	1	Forward homing to switch, Bi direction. Detects switch on rising edge and returns to find falling edge. This is a more accurate method than the Uni direction –method.
1	2	Reverse homing to switch, Uni direction. Stops on switch rising edge.
1	3	Reverse homing to switch, Bi direction. Detects switch on rising edge and returns to find falling edge. This is a more accurate method than the Uni direction –method.
2	0	Forward Torque homing. Stops on actual torque exceeding the Torque_Max –setting.
2	1	Forward Torque homing. Stops on actual torque exceeding the Torque_Max –setting.
2	2	Reverse Torque homing. Stops on actual torque exceeding the Torque_Max –setting.
2	3	Reverse Torque homing. Stops on actual torque exceeding the Torque_Max –setting.

X: Not relevant

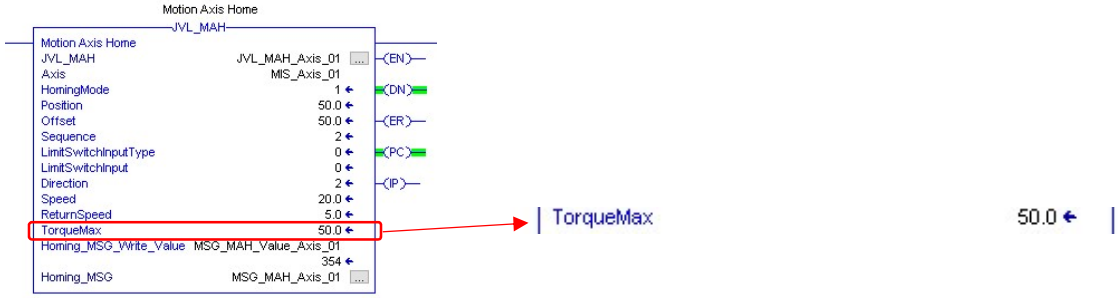
For Active = 0 the same applies as if Sequence = 0. The encoder position is reset and the a new reference point for the motor will be set.

Due to the nature of the Homing sequence the message paths needs to be configured for the motor. The COP –instruction needs to be executed prior to executing the MAH –instruction.

Torque homing

Torque homing is only possible for motors equipped with the encoder option H2 or H4. This homing method requires that the mechanics of the application is able to stall the motor at the desired homing position.

The torque threshold value is set in the *TorqueMax* –tag the unit is [%]



Sensor homing

Sensor homing requires a free input on the PLC that needs to activate the *LimitSwitchInput* –tag in the JVL_MAH –instruction.

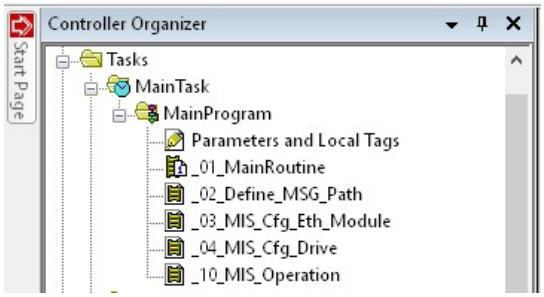
The example below shows how to setup a sensortype homing using Input 0 on the 5069-1B16 –input module on the PLC.

Supplied MIS AOI –Example

The supplied example for the MIS AOI usage is a demo case on how to configure and setup a project using the MIS AOI.

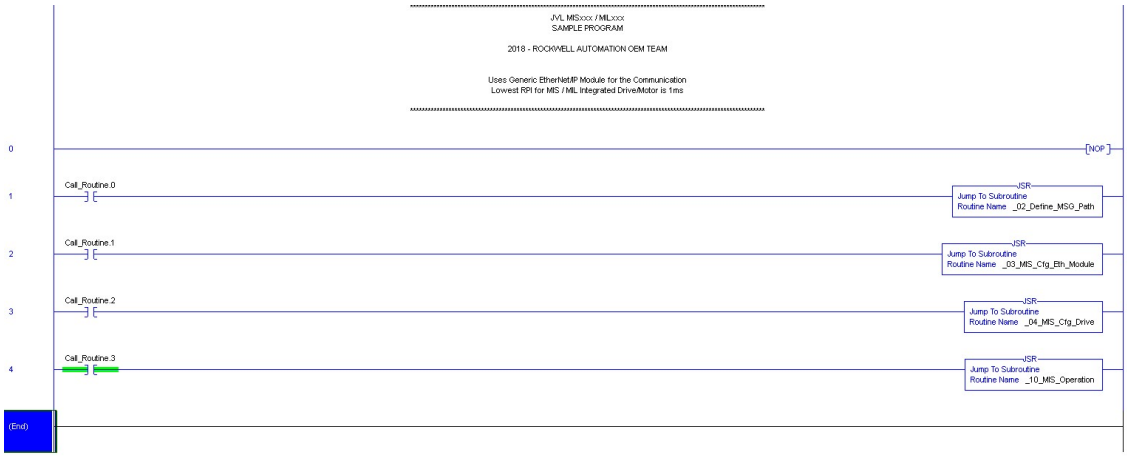
Loading the project will automatically include all AOI's and no real configuration is required besides the IP settings.

The Example is divided into 4 different routines besides the MainRoutine.



_01_MainRoutine

The routine handles calls to the underlying sub routines.



The CALL_Routine –tag holds a bit for each routine to call.

For a new motor not previously configured the following routines needs to be called:

1. _02_Define_MSG_Path
2. _03_MIS_Cfg_Eth_Module
3. _04_MIS_Cfg_Drive

When the routines has been called the settings are stored in volatile memory in the motor and the I/O should be established automatically. The I/O size doesn't match the motor before it is configured, so an I/O fault will be active.

_02_Define_MSG_Path

This routine converts the IP –address into a message path that can be used in the different message object used.

The motor is by factory default configured with the IP address 192.168.0.xx, where xx is the last byte from the MAC-address.

The resulting path definition is placed in the MSG_Path –tag.

_03_MIS_Cfg_Eth_Module_

This routine configures a factory defaulted module for with the right Cyclic setup and for 8x I/O usage.

The routine is using the MSG_Path –tag defined in the _02_Define_MSG_Path_ -routine.

_04_MIS_Cfg_Drive_

Only relevant for MIS motors with the H2 and H4 encoder options.

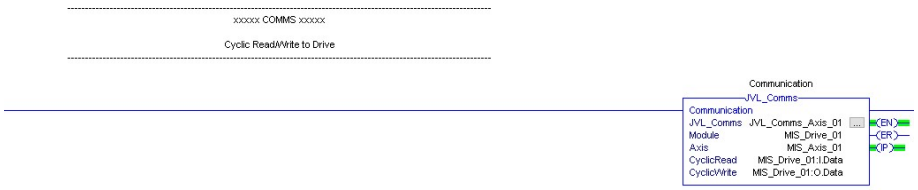
This routine configures a factory default MIS stepper motor.

Since the release of V4.04 firmware the Closed loop and current control are enabled by default and this routine is obsolete.

_10_MIS_Operation

This routine holds the calls for all the AOI motion instructions.

The JVL_Comms –instruction handles the Data conversion and needs to be called in each program cycle.



The JVL_Scaling instruction will setup the scaling parameters according to the application.

In this example the scaling has been setup for 78 units per revolution, no gearbox attached so gear ratio is 1:1 and the units used are [mm] and [mm/s].

So for a linear application this means that each motor revolution = 78 mm.

The homing procedure will be processed when the Axis_01.Home –bit is set high.

2 Rungs are executed the JVL_MAH and the message path are configured.

This page is left blank.