Getting Started With AKD Ethernet IP Rev C Mar 2016

Overview

Support documents that should help you get started are at the following link.

http://www.kollmorgen.com/en-us/products/drives/servo/akd/

Support files and documents related to AKD Ethernet IP:

1. Kollmorgen AKD Ethernet IP Communications Manual
2. Kollmorgen AKD Ethernet IP RSLogix Communications Manual
3. Add On Instruction Library for AKD Ethernet IP (contains Add On instructions and Data Types).
4. Change_Log.txt; A text file that documents the Add On Instruction library revisions, bugs, and fixes.
5. Sample Project for RSLogix5000 Users
6. Sample Project for Studio5000 Users
7. Sample Project for RSLogix500/Micrologix1400 users.
8. Sample Project for Registration application
9. Firmware and EDS file.

   Note: Some products/controllers may require the EDS file but we have not used it with Compactlogix or Contrologix. It is available if required (i.e. other Ethernet IP controllers). In general the AKD is setup as a generic Ethernet Module in the Allen Bradley software.

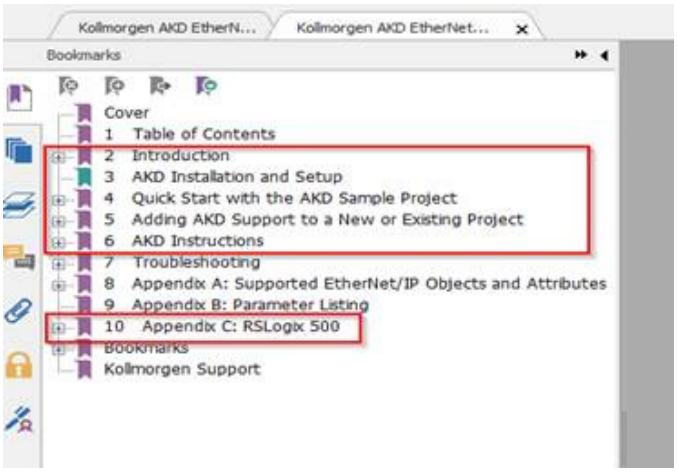
The AB products and processors we have tested the AKD with are:

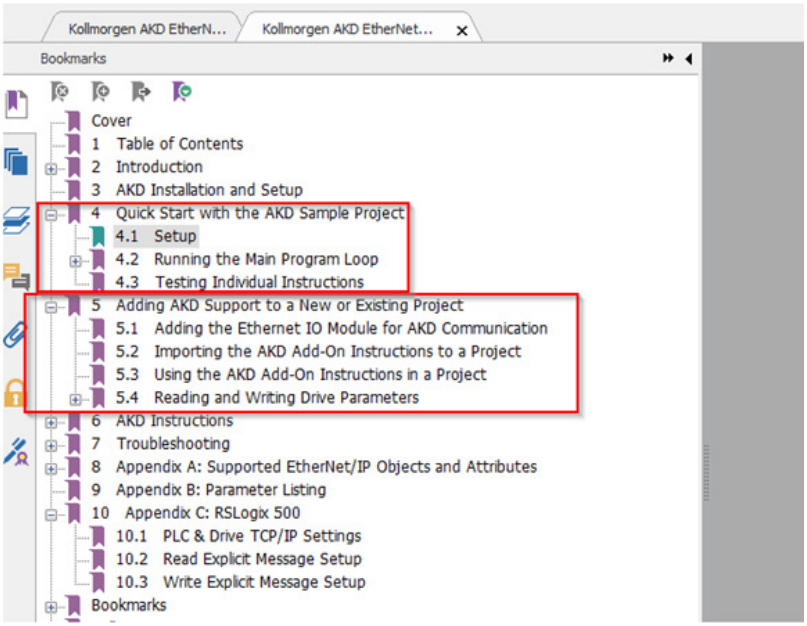
1. Micrologix 1400
2. Compactlogix
3. Contrologix

Note the Micrologix 1000 and SLC500 5/05 are NOT included in this list and will not work with the AKD Ethernet IP drive.

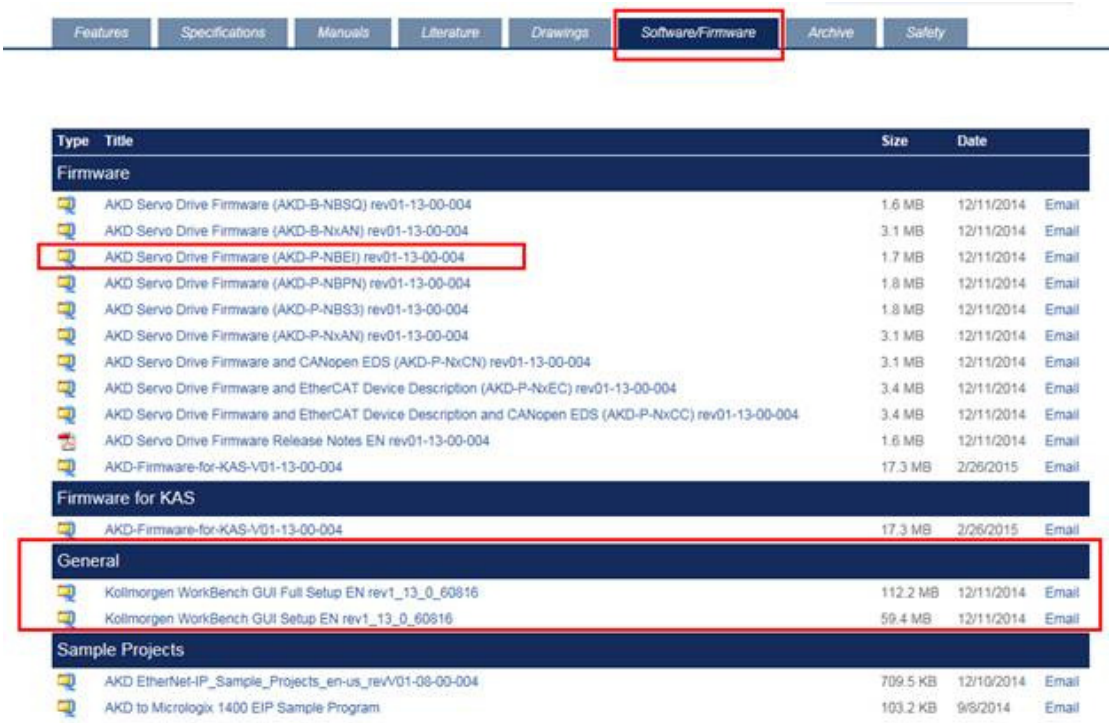
Quick Start and Preliminary Setup for Using Sample Project

The Kollmorgen AKD Ethernet IP RSLogix manual covers installation and setup, Quick Start with the sample projects and adding the AOI libraries (creating a new project from scratch or adding the functionality to an existing project). For RSLogix500 users, there is a section on that in Appendix C. Note RSLogix500 and the Micrologix 1400 controllers support Explicit messaging only via the MSG block and do not support the Add On Instructions which depend on cyclic messaging available only in RSLogix5000 and Compactlogix and Compactlogix processors.



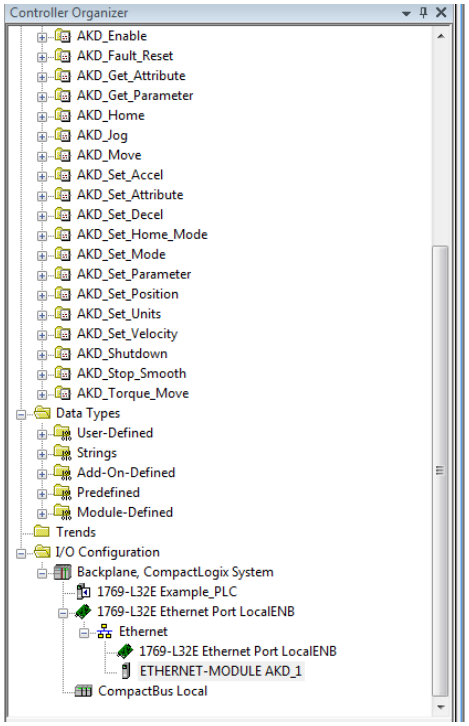


1. It is always best to start a new application with the latest version of the Workbench software and AKD drive firmware.

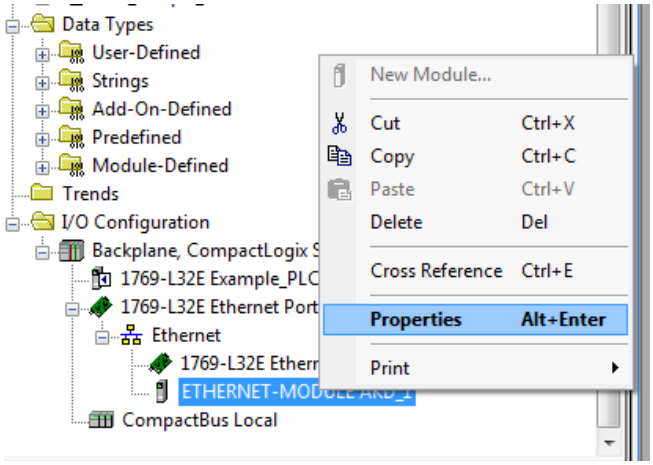


2. Although the manual covers how to start a project from scratch or to add to an existing project, the user is strongly encouraged to start with the sample project(s). This will allow the user to confirm the PLC is communicating with the AKD (start with 1 node/IP Address first before adding others). The sample projects also provide a way for the user to experiment/bench test manually triggering each AOI to perform all the basic functions (i.e. homing, jogging, indexing, etc.). The sample projects for RSLogix5000, Micrologix 1400, and Studio5000 also demonstrate proper timing/seal-in logic for the AKD Function Blocks to consistently execute when triggered.

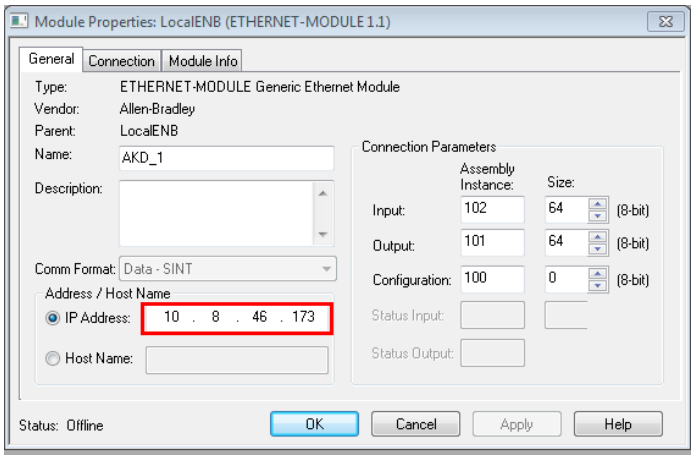
3. To use the Sample Project to test with 1 AKD Ethernet IP drive, open the project file and navigate to the Ethernet-Module AKD_1 under Ethernet in the Controller Organizer tree.



Right-click on the ETHERNET-MODULE AKD_1 and select properties.

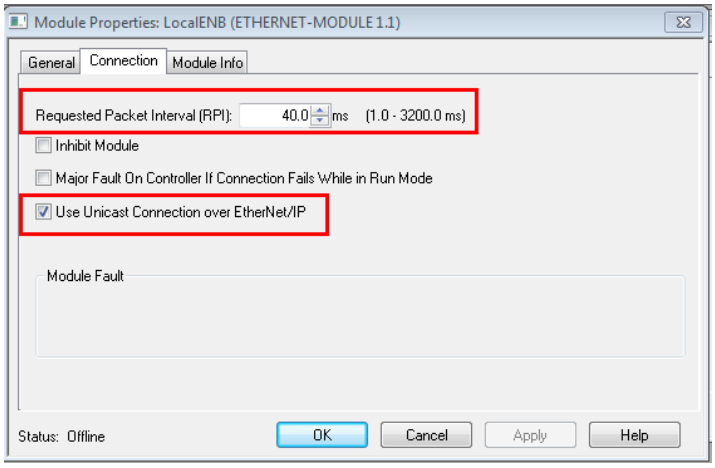


Under Change the IP Address of the AKD to match the target IP address of your AKD drive.



It is important to note on the Connection tab:

1. The Requested Packet Interval (RPI) is set to 40 ms in the Sample Projects.
2. The “Use Unicast Connection over EtherNet/IP” checkbox is checked.



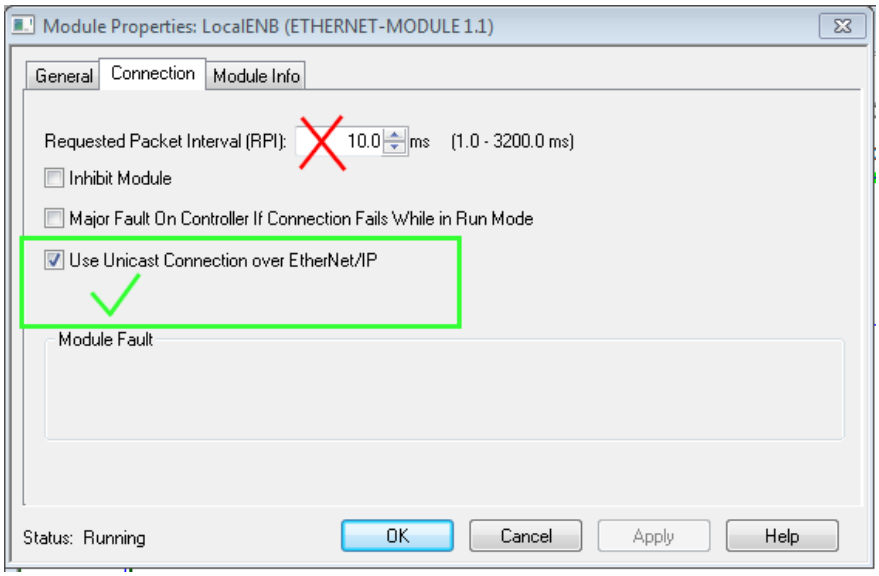
This convention should be used once you’ve finished the test with one node/IP address and are ready to add more AKD drive (Ethernet Modules) to your project.

The general rule based on experience is:

DO NOT: Set the RPI lower than 40msec when using both Workbench and RSLogix.

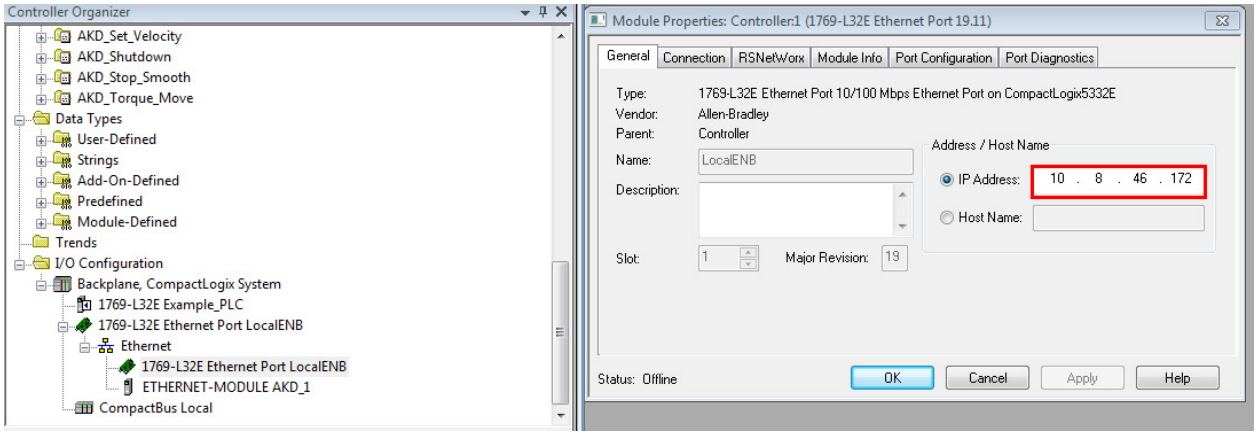
DO NOT: Set the RPI lower than 20msec when using RSLogix alone.

DO: Check the “Use Unicast Connection over EtherNet/IP” checkbox.



Attempts to set the RPI lower than 20msec will result in missed packets, timeouts, and over all communication issues.

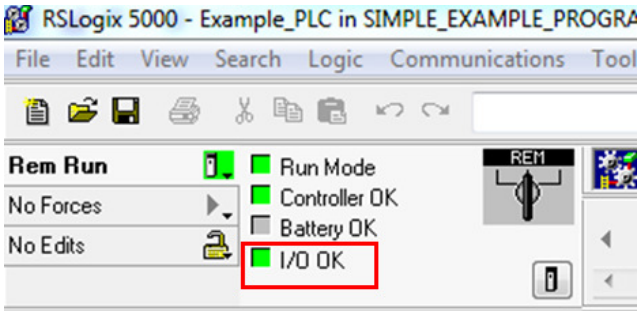
Now the AKD drive IP address is set, right click on your PLC’s Ethernet port to configure its IP Address.



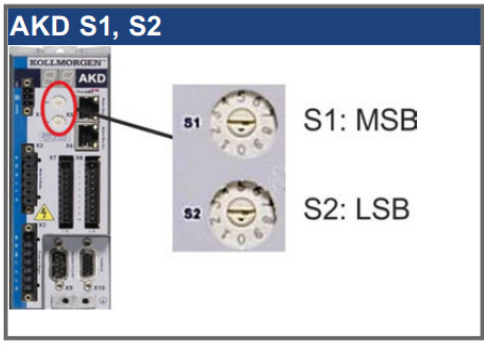
For a one PLC: one AKD drive network, these should be the only modifications required to the Sample Project to run an initial test.

Save your project and then download.

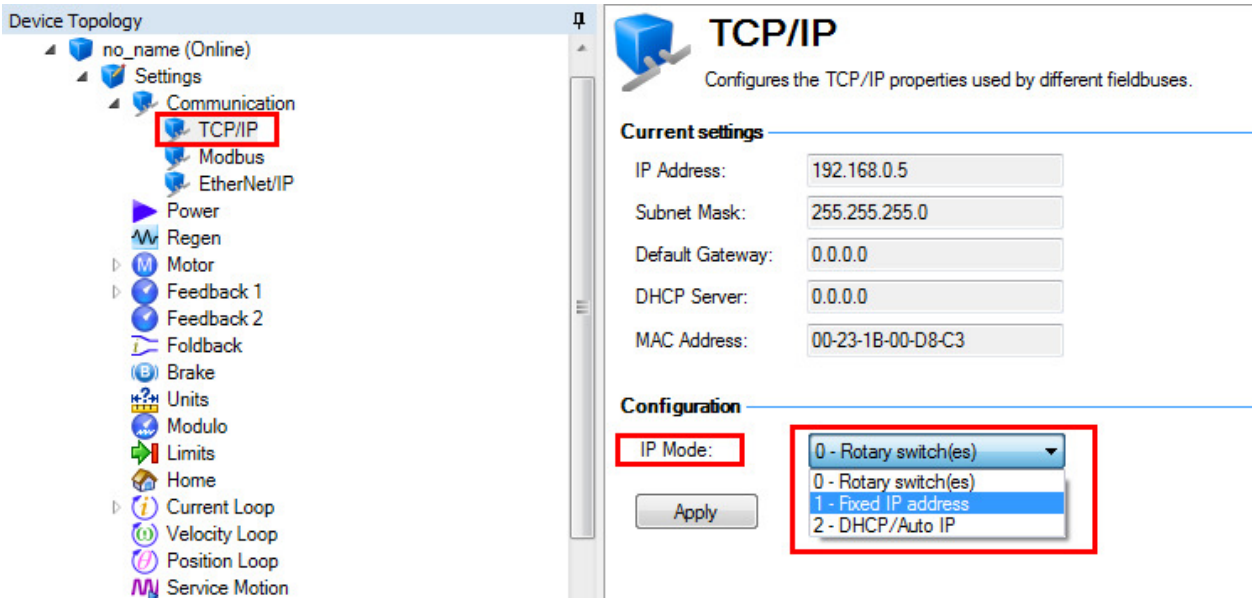
Once online, the I/O OK indicator lamp should be a solid green indicating the PLC is communicating to the AKD drive. Green and flashing indicates there is an issue with communications which can range from cabling, IP addressing conflicts, other hardware issues such as Ethernet switches, routers, etc. If you do not have communications, make sure you can ping both the PLC's and AKD's IP Address from your PC. There is a B1 button on the top of the AKD drive; when pressed the current IP address of the drive will be displayed in sequence.



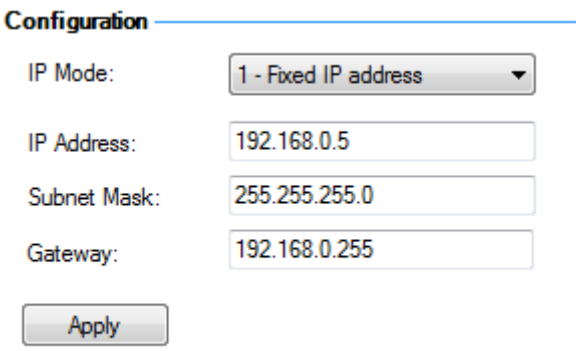
The AKD drive, as a default, uses the rotary switches S1 and S2 on the front of the drive.



The drive's IP address follows a 192.168.0.nn convention where nn is determined by the S1 and S2 rotary switches. For example, S1=0; S2=1 sets the drive's IP address to 192.168.0.1. S1=0;S2=2 sets the IP address to 192.168.0.2 and so forth. Many networks and other AB devices often have a different network convention than 192.168.0.nn. In this case it is possible using Workbench to set the drive's IP address by manual entry via changing the IP Mode to "1-Fixed IP Address".



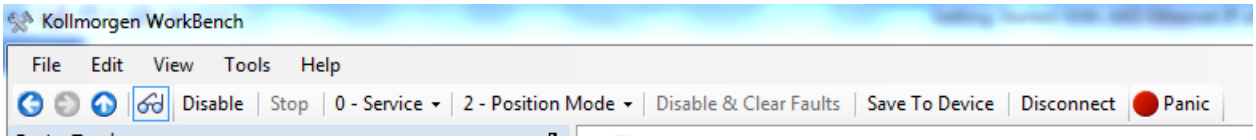
This will enable manual entry.



Assuming communications is established, for the test we will also open Workbench and connect to the AKD drive.

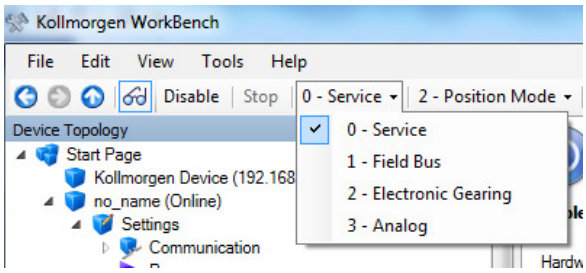
After connecting, the basic setup of the drive and motor should be performed first as with any AKD. See other documentation for details as this quick start's focus is Ethernet/IP.

The drive is set for Service and Position Mode.

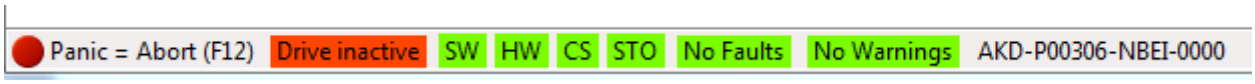


Note other command sources or operation modes may be used in the application and it is possible to switch between operation modes from the PLC but, in general, most applications that use the AOI such as AKD_HOME, AKD_JOG, AKD_MOVE operate in Service, Position Mode which will be assumed for this test.

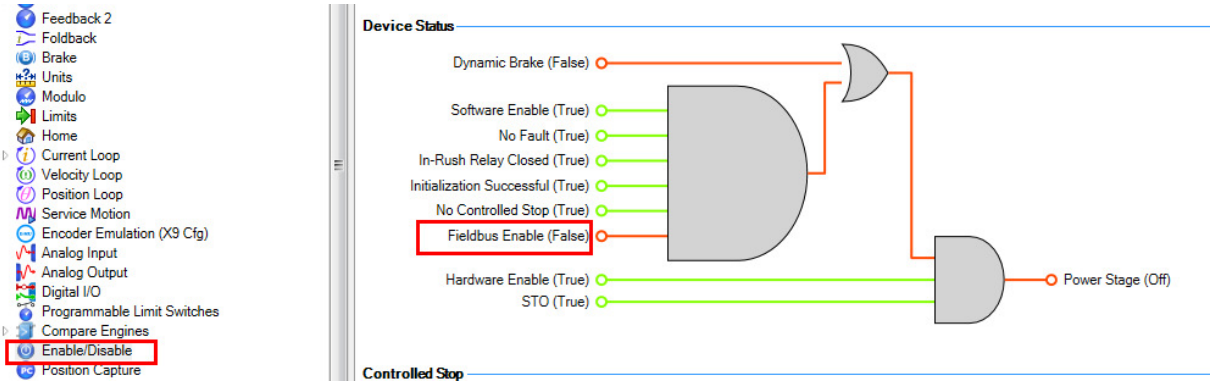
Also note command source 1-Field Bus does not apply to Ethernet/IP.



Assuming everything is ready for the drive/motor to be enabled (HW enable, STO, the basic drive setup is complete such as motor, feedback, etc), the status at the bottom of Workbench should appear as the following.



Note the drive is inactive. If you look at the Enable/Disable screen you will note everything is green except the Fieldbus Enable. This enable will come from the PLC using the AKD_Enable AOI.



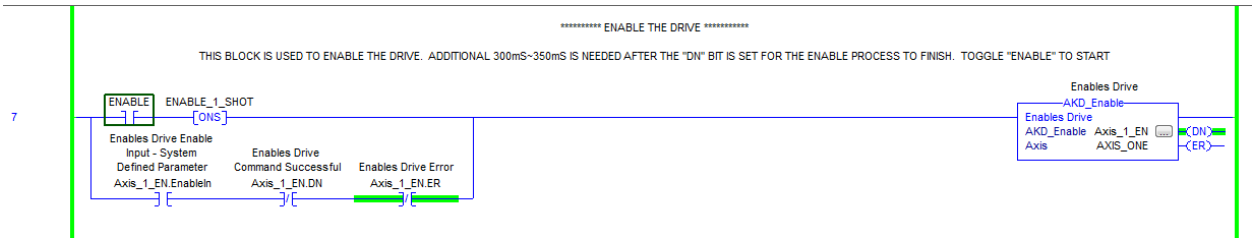
Understanding and using the Sample Project and programming methods:

From the Sample Project in RSLogix5000, the following rung shows the logic for triggering the AKD_Enable add-on-instruction. You will note that this convention follows through all the other rungs with AOIs in the Sample Project. In some cases the EnableIn, Done, and Error bits are used in the seal-in logic. In other cases the PC (process complete) bit is used in place of the done bit (see examples below).

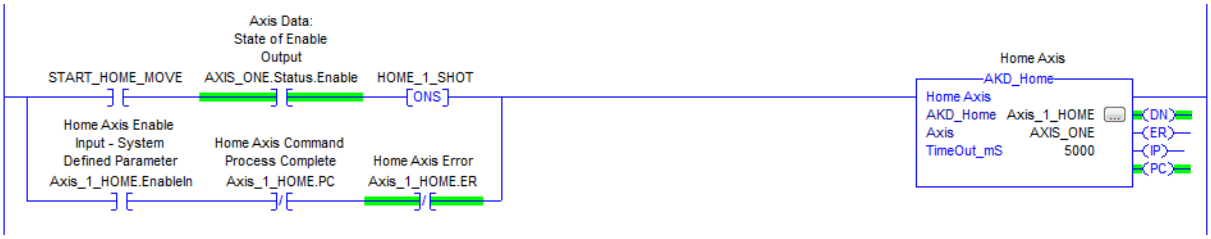
A normally open contact with the tag name “Enable” is provide so you may toggle it while online to run the initial test. Long term the user may conditionalize this tag (or substitue the tag with another tagname) to trigger the AOI based on the program logic.

It is important to note later if you add other subroutines and more axes, this methodology should be replicated for those axes as well.

Rung without a PC bit (process complete):



Rung with a PC bit (process complete):



Foreword on method and timing:

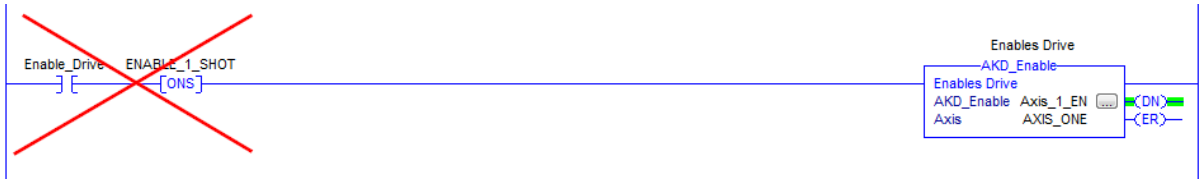
- Only one AOI can execute at a time PER axis. We have seen cases where the user’s program causes the AOIs to get into a state where they either don’t execute or lock up altogether. An example is the AKD_Enable block which as indicated in the sample program an additional 300-350 msec is needed AFTER the Done bit from the AOI turns on. The AOIs use the same area of the cyclic command and response assembly for a given Generic Ethernet Module (IP Address). Triggering another AOI while another one is still executing overwrites the commands in that assembly and causes a conflict for the control of the communication channel.

- Related, often programmers will attempt to use “Done” bit to move on to the next step which can cause lockups and other issues. The reason for this is “Done” indicates the AOI has been successfully enabled to execute, NOT that it has completed execution. AOIs with a PC (Process Complete) bit provides an indication that the AOI execution has actually completed. There are also status words and information in the response assembly that can be used to confirm completion, states, etc. (more on this topic later).

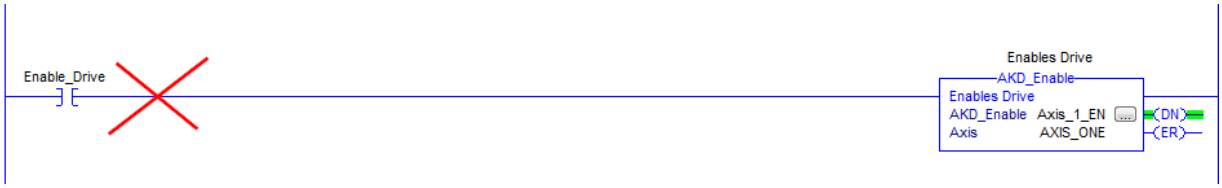
- We've seen user programs where one-shots (1 scan) only or maintained contacts are used in the logic which either doesn't hold the block enabled long enough or too long. The manual indicates only one AOI can execute at a time. This means PER axis. You can trigger multiple AOIs at the same time as long as each corresponds to its own axis. This should be intuitive as you may want to home or move multiple axes at the same time but it would not make sense to trigger axis one's Jog AOI and then while it the jog is executing trigger a Move for that axis (for example).

Common mistakes:

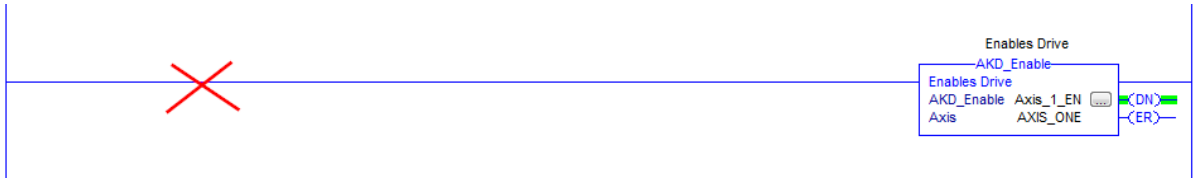
This rung only enables the AOI for one scan which is often not long enough to execute or finish executing.



This rung potentially keeps the AOI enabled forever if the N.O. contact stays on or is forced on; conflicting with other AOIs executing.

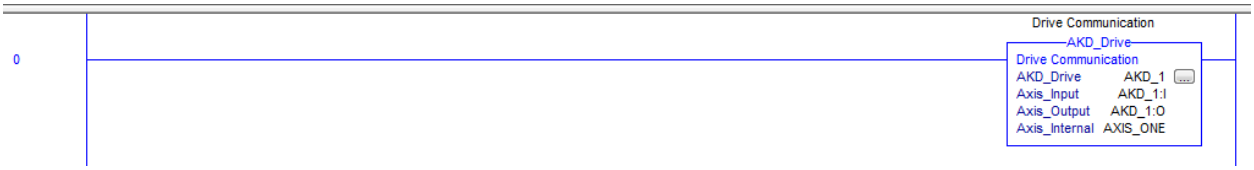


This rung unconditionally keeps the AOI enabled forever; conflicting with other AOIs executing.



Only the AKD_Drive (Communication) AOI can be unconditionally tied to the rail (at the top of the program or subroutine)

DO NOT: Put the AKD Drive Communication AOI in the ladder after AOIs related to that axis (i.e. AKD_Home, AKD_Jog). It should be at the top of the ladder or subroutine for that axis as shown.

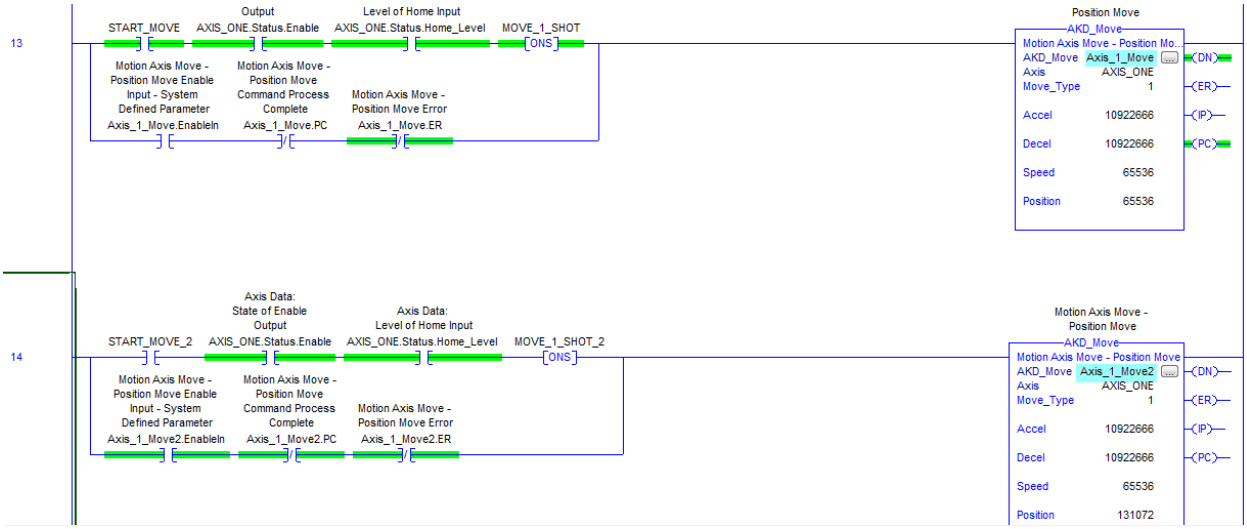


Note: The AKD_Drive (Drive Communication) AOI's configuration and functionality is tied to the Generic Ethernet Module and thus IP address as well. The handshake with cyclic data (command and response assemblies) are tied to the Axis_Input and Axis_Output) of this block when declared. The "Axis_Internal" is the name you want to give to the axis throughout your program. For example, "Vertical Lifter" or "Conveyor" ,etc. All other AOIs depend on this handshake and the axis name defined by "Axis Internal". Once declared at the top of your program or subroutine, all of the AOIs used in the ladder code that follows related to that axis must have the same name in those AOIs as well when configuring the AOI and inputting the Axis name into "Axis" for the AOI. For multiple axes this must be true for every axis. This is how the AOIs know which axis it is related to.

DO NOT: Rely on the DN (Done) bit to indicate the AOI's process is complete. The DN bit means the AOI was successfully triggered to execute; not that the AOI is finished processing the command. Use the PC (process complete) bit when possible or use diagnostics and status words to confirm completion or change of state.

DO NOT: Name AOI instances with the same tagname. Every AOI added to your project must have a unique name.

The following demonstrates where another move was added to the sample project. The original move was named "Axis_1_Move" and the new move was called "Axis_1_Move2". The toggle bit, one shot, and seal-in contacts in the new rung were also given new tags.



Using the Sample Project to control the drive and generate motion from the PLC

- Note: This procedure assumes the motor is unconnected to any mechanics and is free to turn independent of scaling and units.

OVERVIEW

The following steps will allow the user to experiment and get familiar with triggering AOIs and basic functionality the AOIs provide. Steps that are covered in the following procedure:

STEP 1: Enable Drive

STEP 2: Disable Drive

STEP 3: Enable Drive

STEP 4: Home

STEP 5: Jog and Stop

STEP 6: Home

STEP 7: Index

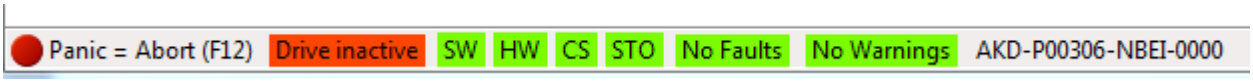
STEP 8: Scaling

STEP 9: Other AOIs in the Sample Project

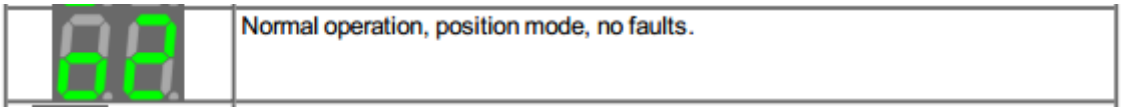
STEP 10: Diagnostics

STEP 1: Enable drive

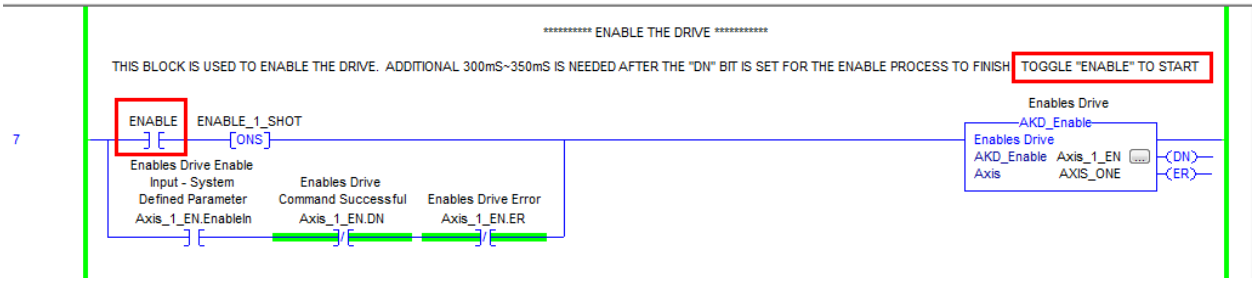
As stated before, assuming everything is ready for the drive/motor to be enabled (HW enable, STO, the basic drive setup is complete such as motor, feedback, etc), the status at the bottom of Workbench should appear as the following.



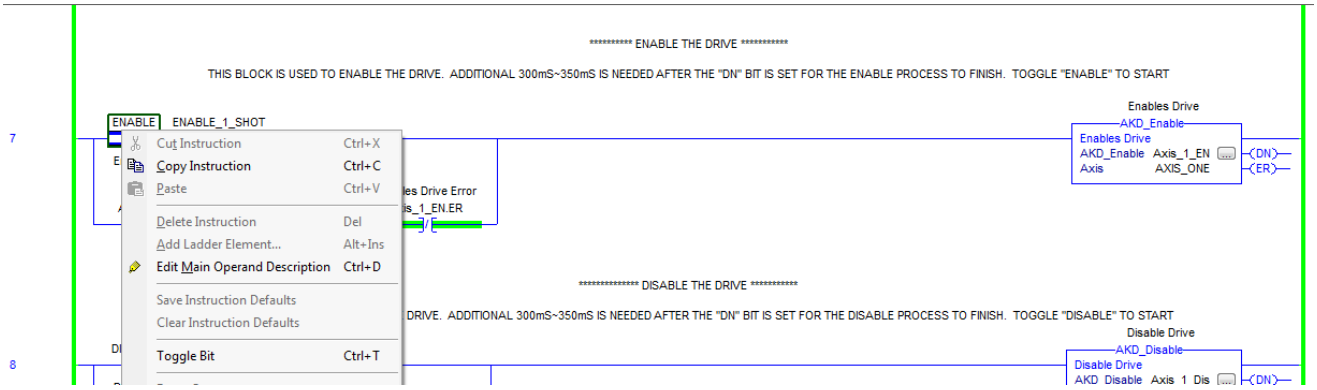
The drive's front display should look like the following:



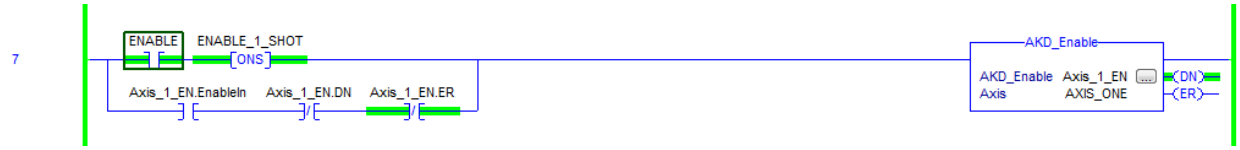
Note the AKD_Enable rung. There are comments related to using the logic and its functionality. There is a normally open contact called "ENABLE" before the one-shot. In the sample project this can be toggled when online to trigger the Add On Instruction (AKD_Enable) in this case. This convention is followed throughout the sample project.



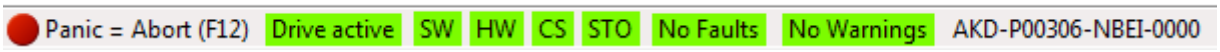
Right-click on the Enable normally open contact and select "Toggle Bit" to trigger the AKD_Enable AOI.



The DN (Done) bit turns on:

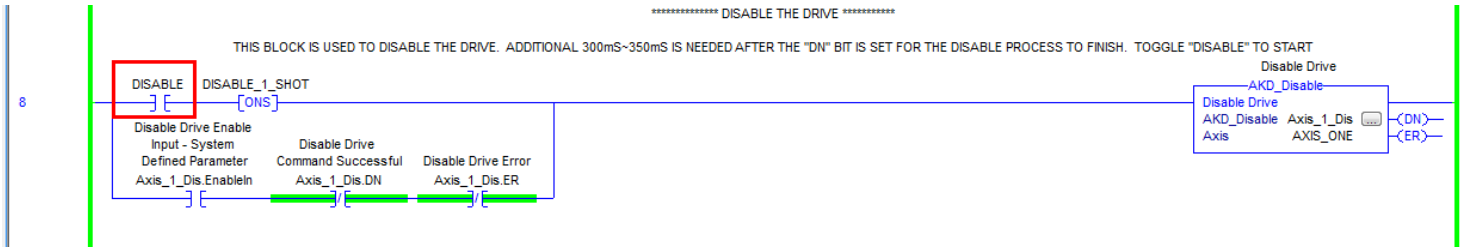


Switching back to Workbench, you can verify the drive was successfully enabled:



In addition to the “02” op mode (position mode) on the front display, the right bottom decimal point will should be illuminated to indicate the drive is enabled.

STEP 2: Disable Drive

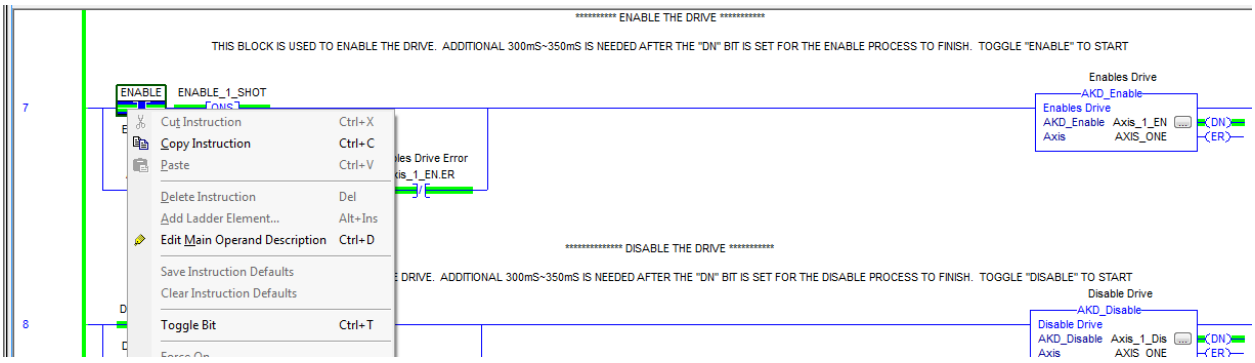


Right-click on the “Disable” normally open contact and select Toggle Bit to disable the drive.

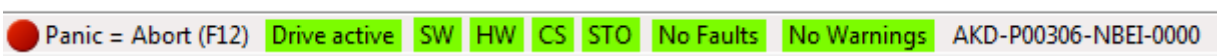
The AKD_Disable DN (Done) bit turns on and verifying in Workbench, the drive is inactive again.

STEP 3: Enable Drive (again)

To re-enable the drive, return to the AKD_Enable rung and right-click on the “Enable” normally open contact. The state is still high from STEP 1. Select Toggle bit to turn it back off.

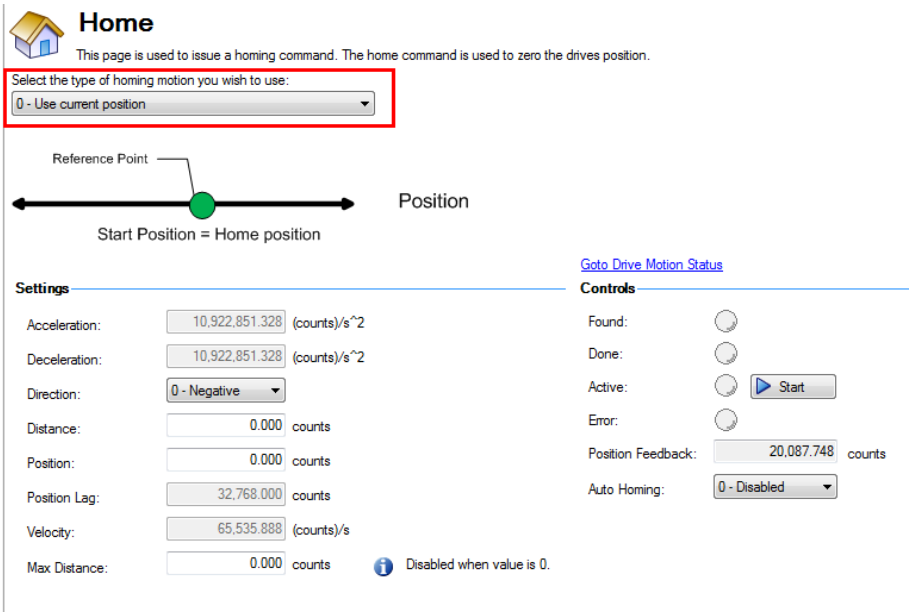


Right-click again on the Enable bit and Toggle it again (turn on) to re-enable the drive.

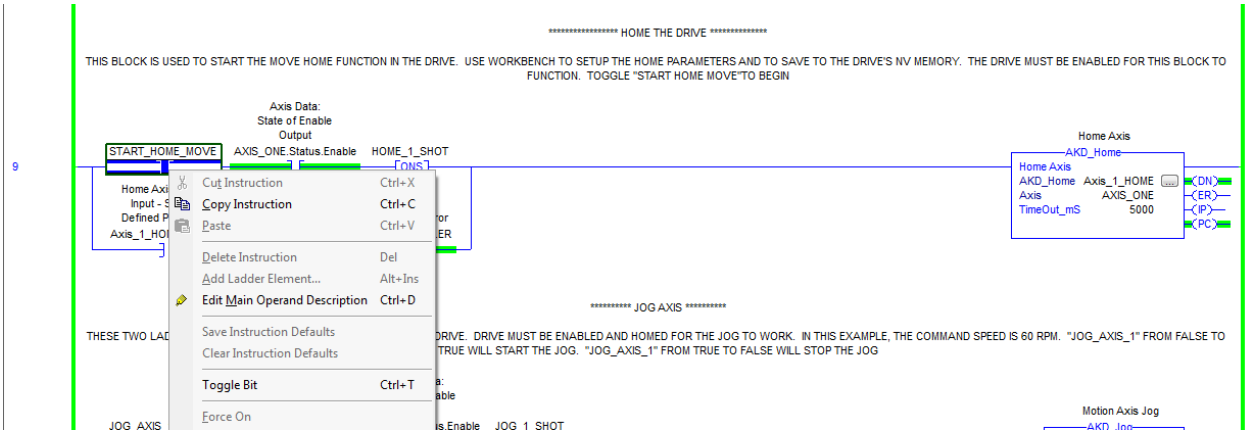


STEP 4: Home The Axis

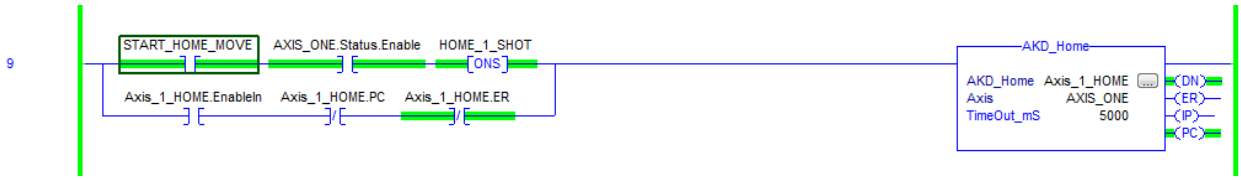
What method the drive uses to home the axis is dependent on the feedback type used on the motor and what Homing Type the drive is configured for. How the drive is configured is up to the user but in this case since this procedure is intended to establish communications between the PLC and AKD drive, get the user familiar with the logic and sample project, and assumes an empty motor, “Use Current Position” was selected.



Next the AKD_Home AOI is triggered in the PLC. Note the AOI essentially performs the same operation from the PLC as pressing the “Start” button in Workbench to start the home move. Right-click on the “Start_Home_Move” normally open contact and select Toggle Bit.

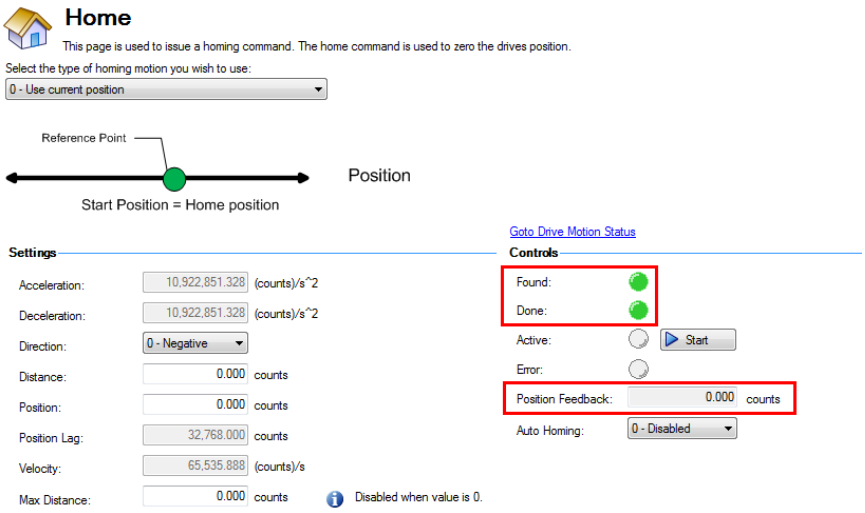


The DN (Done) and PC (Process Complete) bits turn on.



Switching back to Workbench to verify the drive is homed:

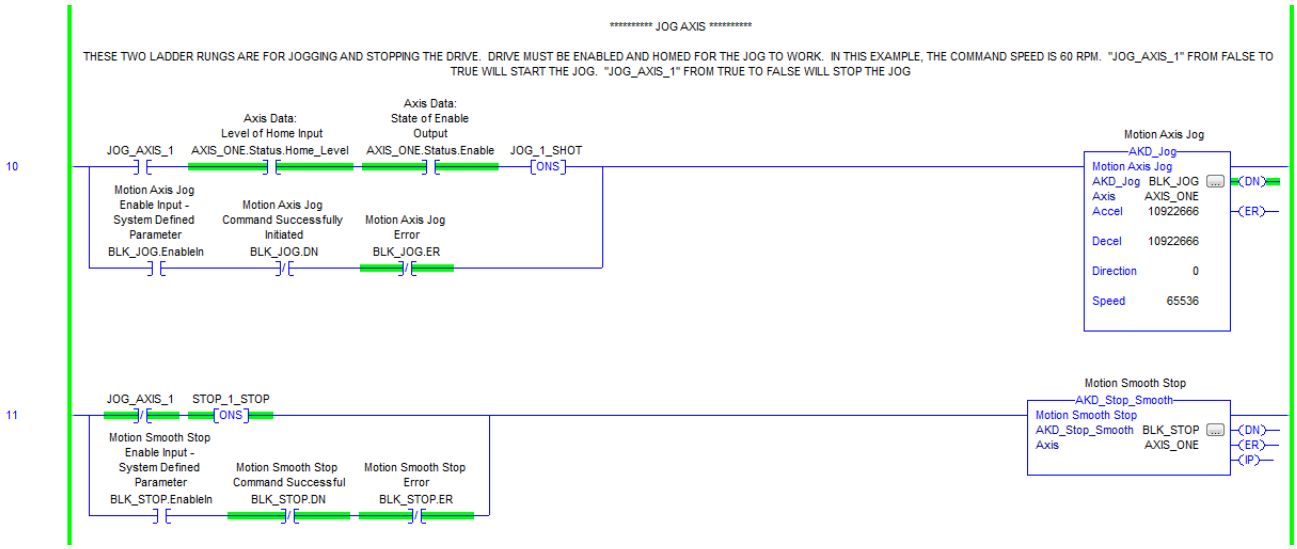
On the Home screen under Settings in the project tree the Found and Done lamps are illuminated and the Position Feedback indicates 0.



STEP 5: Jogging The Axis

Note the AKD_Jog AOI uses Motion Task 0 if the drive is in the Position Op Mode which requires that the AKD drive Home Found and Done bits are set before a Motion Task can be executed. This is why in this procedure the AKD_Home was executed before jogging (Note: the AKD_Jog AOI can also operate in with the drive in Velocity mode which does not require the home to be found and done). Looking at the logic below, you will notice the “AXIS_ONE.Status.Home_Level” contact interlocks the logic so that the AKD_Jog AOI can’t be triggered until the drive is successfully homed. The sample project only has provisions for jogging in one direction. Usually jogging in both directions is implemented by either changing the direction value in the AKD_Jog AOI using a tag instead of a constant or creating another instance of the AKD_Jog AOI in another rung with the direction value set for the opposite direction. Keep in mind every AOI declared in the PLC program should have a unique tag name (instance).

Sample Project Jog Logic:



There are a couple of things to note in the logic above.

1) The direction is 0 which is defined for the AKD_JOG AOI as:

Direction	DINT	Immediate	**For this jog direction:**	**Enter:**
			Forward	1
			Reverse	0

2) The speed is set to 65536 counts/sec. With default Ethernet/IP scaling this will result in a target velocity of 1 rev/sec or 60 rpm (more on scaling later).

3) Note the next rung uses the AKD_Stop_Smooth AOI and the trigger is a normally closed contact with the same tagname as the normally open contact in the AKD_JOG rung above. This is done in the sample project because the AKD_Jog AOI is edge triggered meaning the drive will continue to Jog even if that AOI's enable becomes false. Therefore in the next rung when the JOG_AXIS_1 has been toggled back OFF, the normally closed contact triggers the AKD_Stop_Smooth and the axis will decelerate to a stop.

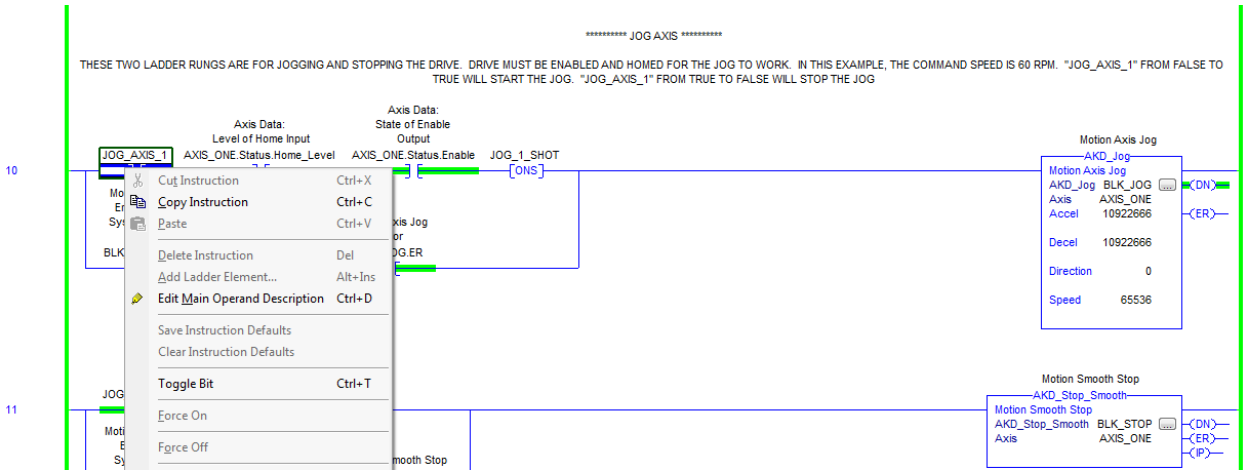
6.15.1 Description

Use the motion axis smooth stop (AKD_Stop_Smooth) instruction to end any controlled motion in process for the axis with a decelerated stop. The instruction stops the motion without disabling the servo loop. This command defaults to stop at the deceleration rate set for the current motion. Corresponds to the MAS instruction in Rockwell drives.

Use the instruction to:

- Stop a specific motion process such as jogging or moving
- Stop the axis completely

To start jogging, right-click on the JOG_AXIS_1 normally open contact and select Toggle Bit.

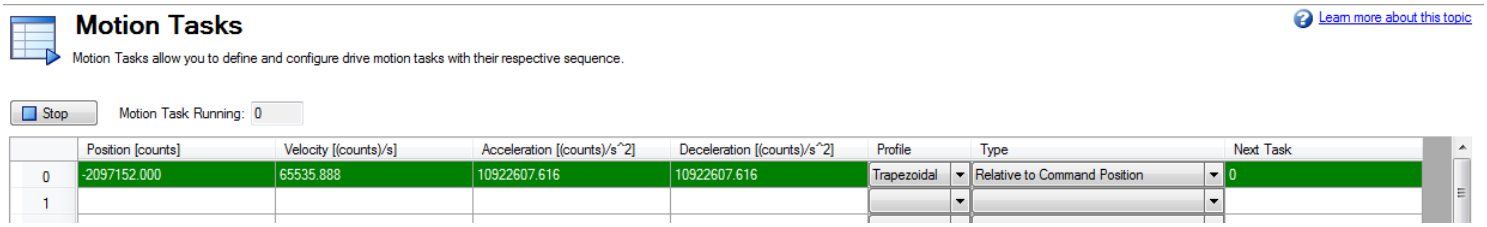


From Workbench and the Watch window you can see the Velocity Feedback is in the negative direction and the actual velocity is approximately the 65536 counts/sec

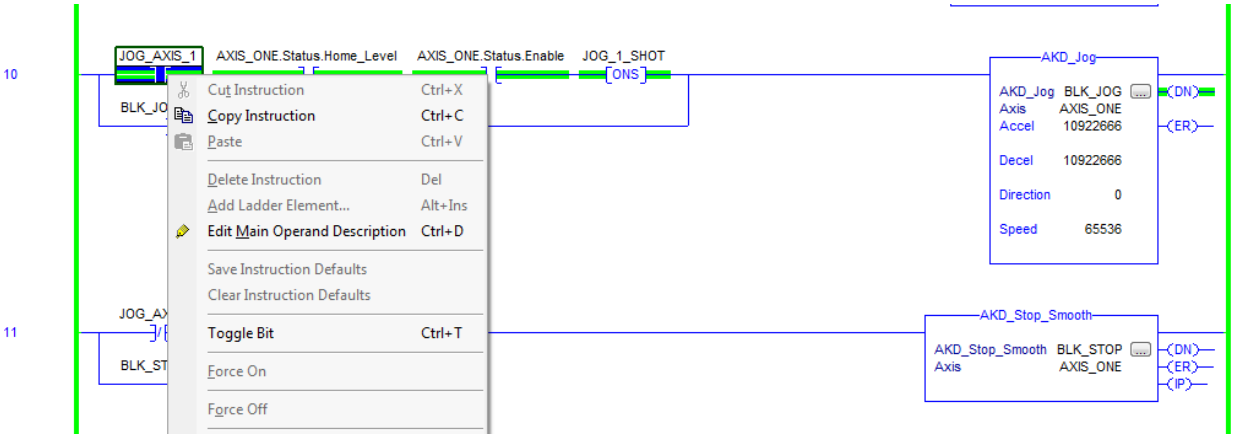
Watch

Enab...	Device	Parameter	Value	Units
☑	no_name (Online)	PL.FB - Position feedback	-48,782,994.000	counts
☑	no_name (Online)	IL.FB - Current feedback	-0.021	Arms
☑	no_name (Online)	VL.FB - Velocity feedback	-67,077.148	(counts)/s

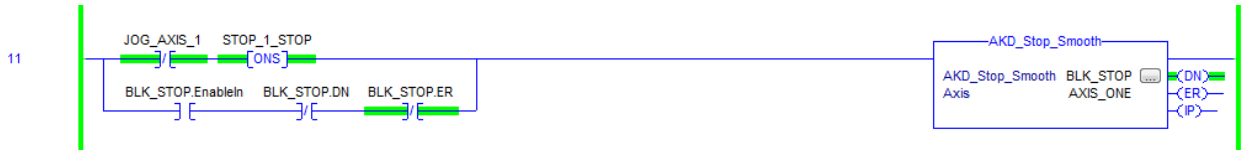
Behind the scenes, the AKD_JOG AOI setup and executed Motion Task 0.



To stop Jogging, right click on the JOG_AXIS_1 normally open contact and select toggle to turn it back off.



Notice now the AKD_Stop_Smooth was executed and the DN (Done) bit is on. The drive/motor decelerated to a stop.



STEP 6: Homing

Since we've been jogging the feedback position is now at some non-zero position. To experiment with the AKD_MOVE AOI, we're going to first re-home the drive. Follow the same instructions in STEP 4 (Home The Axis).

The feedback position should be approximately 0 counts.

Watch

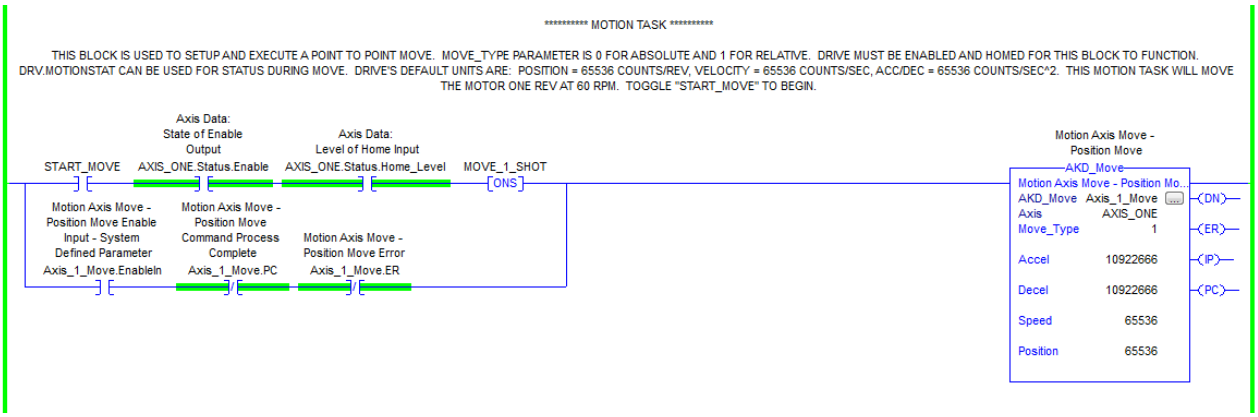
Enab...	Device	Parameter	Value	Units
☑	no_name (Online)*	PL.FB - Position feedback	0.268	counts

STEP 7: Indexing

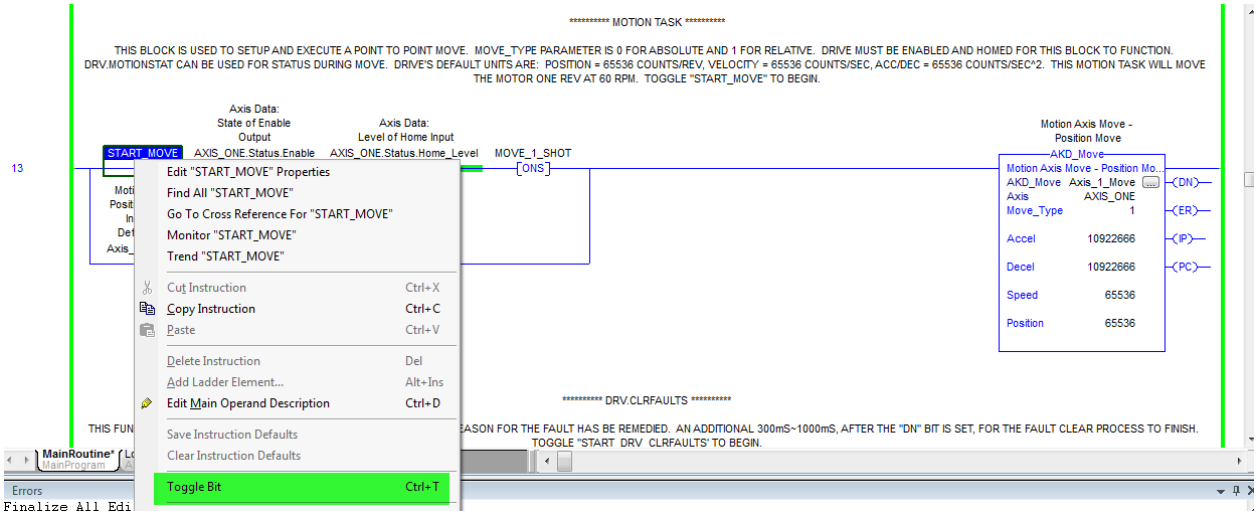
Next take a look at the AKD_MOVE instruction and rung in the sample project. The AKD_MOVE AOI is setup with constants in the sample project. It is possible to later to edit the speed and/or position and replace the constants with tags in order to make the speed and/or position variable. In the event the values are changed, the AKD_MOVE AOI must be retriggered in order for the change to take place. For the quick start and sample project, we're going to use the constants. Scaling will be explained later in this quick start guide but the speed constant of 65536 is counts/sec or 1 rps or 60 rpm (the same as the AKD_JOG speed previously covered). The position constant of 65536 is counts/rev or 1 revolution with the default Ethernet/IP scaling. Note the Move_Type is defined as follows:

Move Type	SINT	Immediate	**For this move mode**	**Enter:**
			Absolute	0
			Relative to Command Position	1

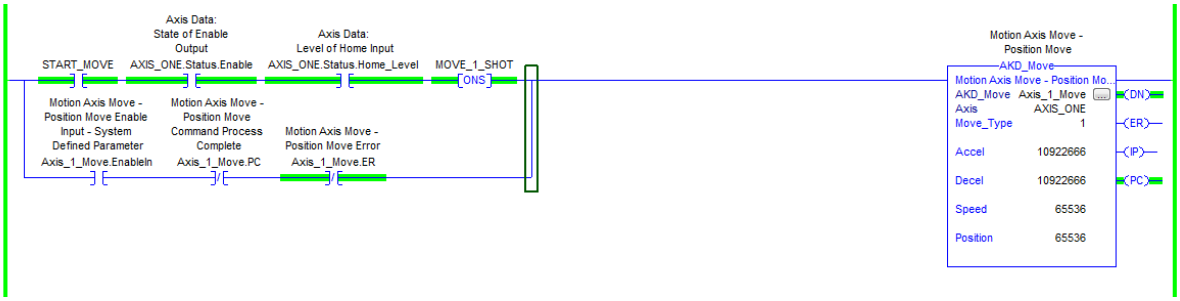
The sample project sets this to a 1 so the START_MOVE can be toggled on/off/on for as many times as desired and each off->on transition will cause the motor shaft to rotate (index) 1 revolution.



Toggle the “Start_Move” N.O. contact to start the index.

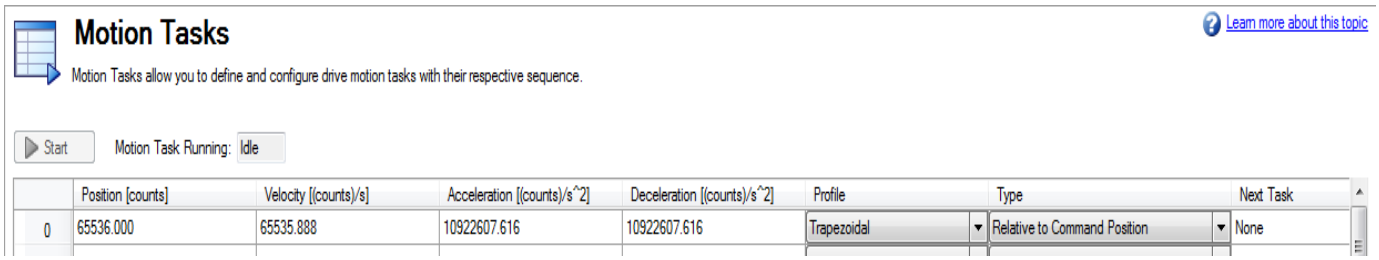


The process complete (PC) bit turns on. The motor rotated 1 revolution.



Verifying the position in Workbench from the Watch window, the target was 65536 counts and the feedback indicates the move was successful.

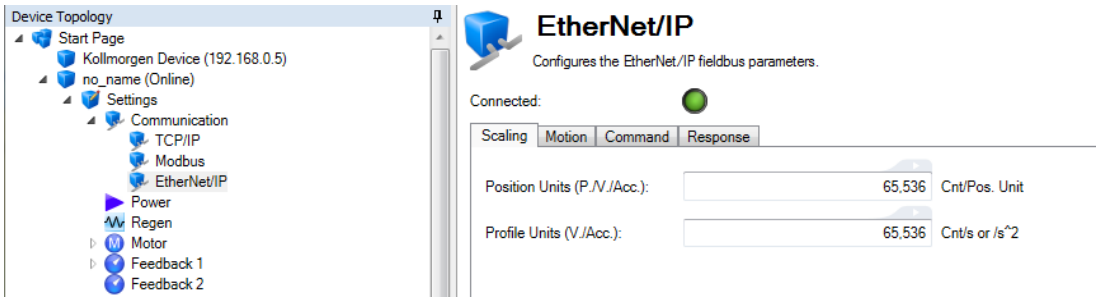
Like the AKD_Jog, behind the scene, the AKD_Move AOI uses Motion Task 0.
If you compare the current data in Motion Task 0 you will see it is consistent with the values/attributes for the AKD_Move block in the sample project.



STEP8: Scaling

It is important for first time users to understand how Ethernet/IP scaling works. The Ethernet/IP standard as managed by the ODVA dictates that the scaling is done in counts/position unit, counts/s or counts/s^2 for position, velocity, and acceleration respectively. It is also important to note that while Workbench provides a way to scale the drive in unit under the “Units” screen in the Workbench project tree, those units only pertain to how the units are set/displayed within Workbench; they have nothing to do with how the position, velocity, and acceleration are set/commanded/displayed from the controller (i.e. PLC, HMI, etc.) using Ethernet IP. From experience, the most intuitive approach is to have the Workbench units scaled the same as Ethernet/IP scaling so the counts in the PLC equal the counts set/read in the drive while monitoring with Workbench.

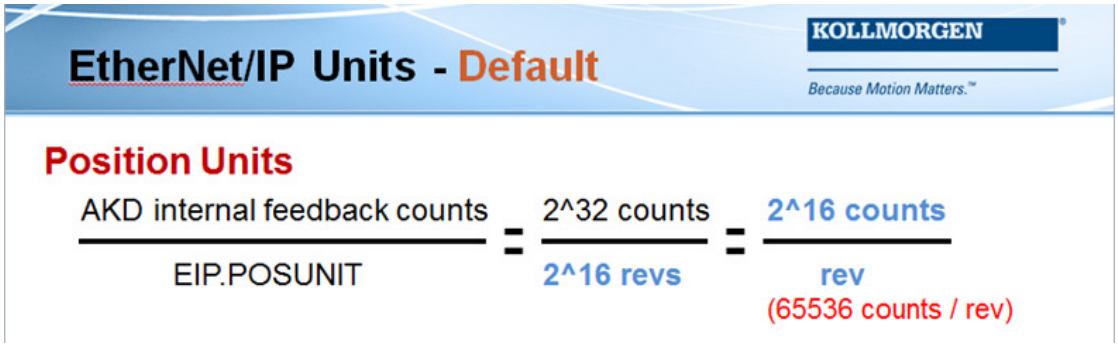
When you are online with the AKD under Settings->Communication->EtherNet/IP->Scaling tab, the default scaling is shown.



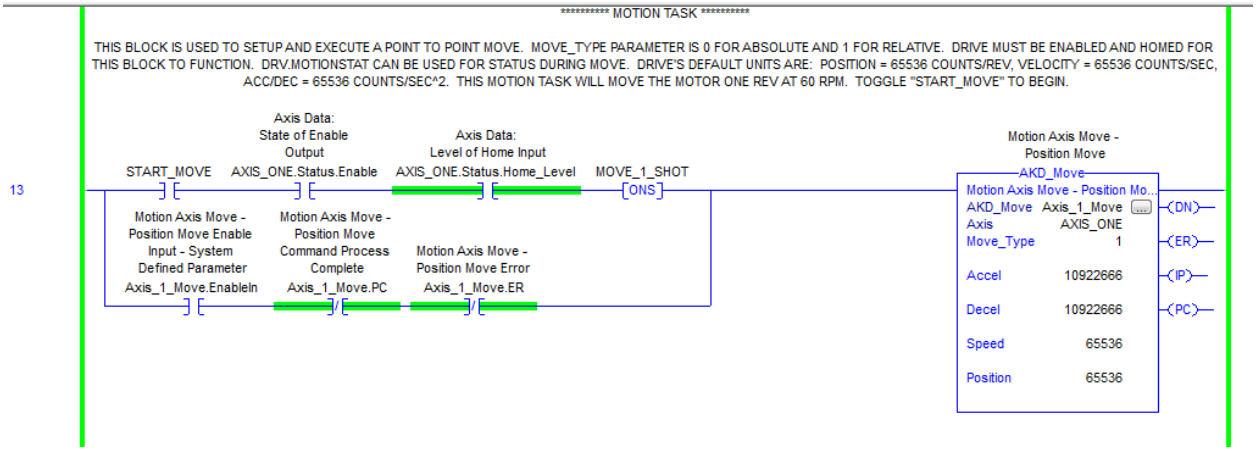
First let’s look at position units. From above, the default scaling is 65536 for EIP.POSUNIT. 65536 is 2^16,

so the actual counts per rev of the motor is given by 2^32/EIP.POSUNIT or 2^32/2^16= 2^16 or 65536 counts.

This means if the AKD_Move block is programmed with a position attribute value of 65536 as in the sample project, when triggered, the motor will make 1 revolution. Again this is true regardless of the Workbench units.



Here is the rung from the Sample Project for the AKD_Move AOI.



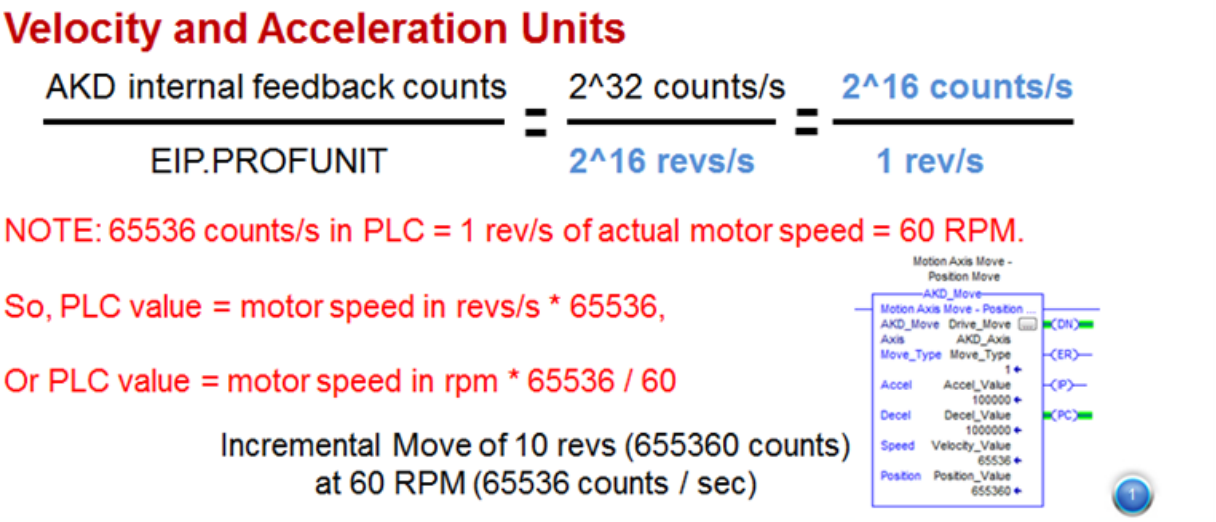
A very important point is that the PLC/HMI must to the math and conversion from real-world units (i.e. inches, inches/sec, etc.) to revolutions and revolutions/sec, etc. and then convert the required revolutions and rev/sec to counts and counts/rev, etc. based on the Ethernet/IP scaling. Also keep in mind the values entered are integer based and not floating point. This means the smallest positional value/increment that can be commanded is 1 count (not fractions of counts). Here is an example:

Horizontal axis, 0.2 inch/rev ballscrew, 5:1 gearbox. Desired units are inches, inches/sec, inches/sec^2.

1 inch	1 rev of ballscrew	5 motor rev	65536 counts	=	1638400 counts
	0.2 inch	1 rev of ballscrew	1 motor rev		

Velocity and Acceleration units are also determined by the Ethernet/IP scaling.

Below, the example shows if the default value of 65536 is used for EIP.PROFUNIT then 65536 counts/sec is 1 rev/sec or 60 RPM. For 10 rps, 655360 is used to set the AKD_Move's Speed value.



Using the same example as above, let's suppose the PLC/HMI wants to set the target velocity during the move to be 5 inches/sec.

Horizontal axis, 0.2 inch/rev ballscrew, 5:1 gearbox. Desired units are inches, inches/sec, inches/sec^2.

5 inches	1 rev of ballscrew	5 motor rev	65536 counts	=	8192000 counts
1 sec	0.2 inch	1 rev of ballscrew	1 motor rev		sec

For acceleration and deceleration units, it follows the same convention with counts/sec^2.

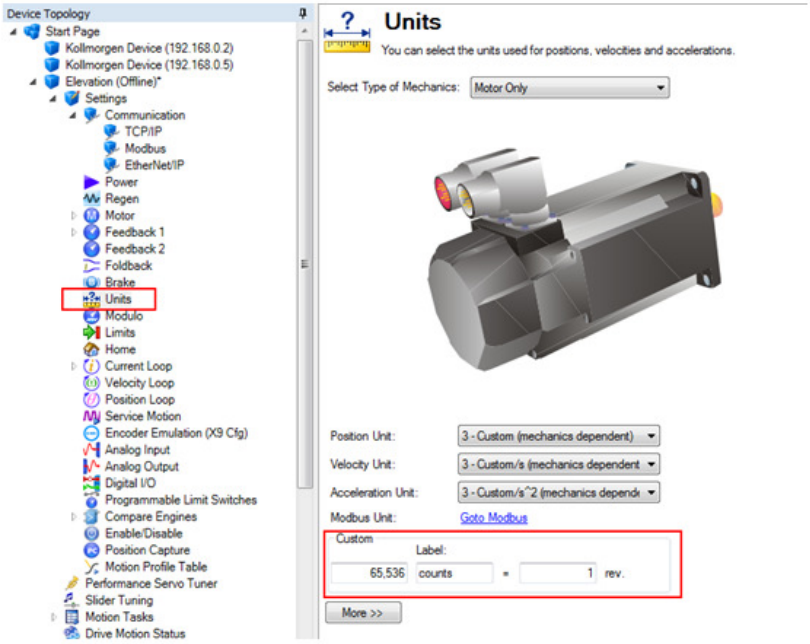
As mentioned previously, Workbench Units can be set to match Ethernet/IP units.

From the Units screen, selecting the mechanics to be "Motor Only", the Position, Velocity, and Acceleration units can be set to "3"- Custom mechanics dependent. This brings up the Custom entry boxes. As shown, 65536 counts = 1 rev .

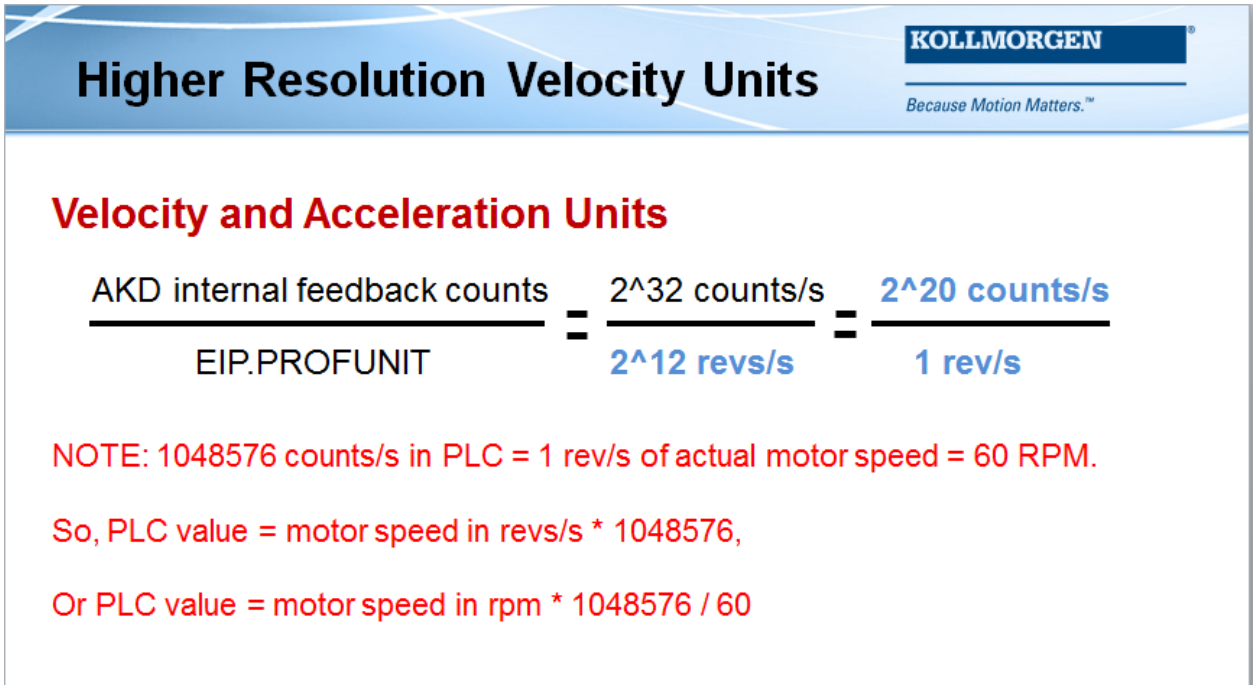
Workbench Units

KOLLMORGEN
Because Motion Matters.™

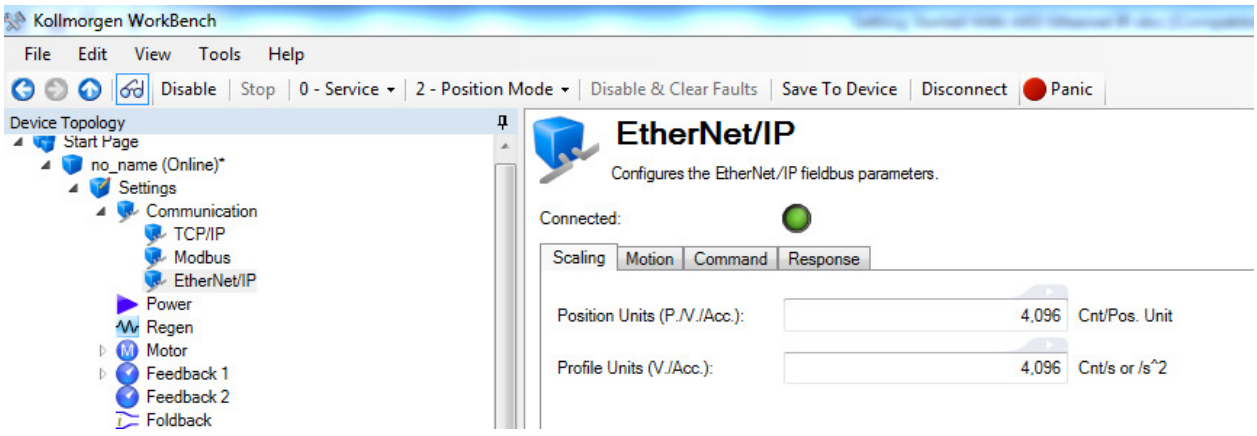
The user units in Workbench can be set to match the Ethernet/IP units. But keep in mind that the Workbench units are unrelated to Ethernet/IP units.



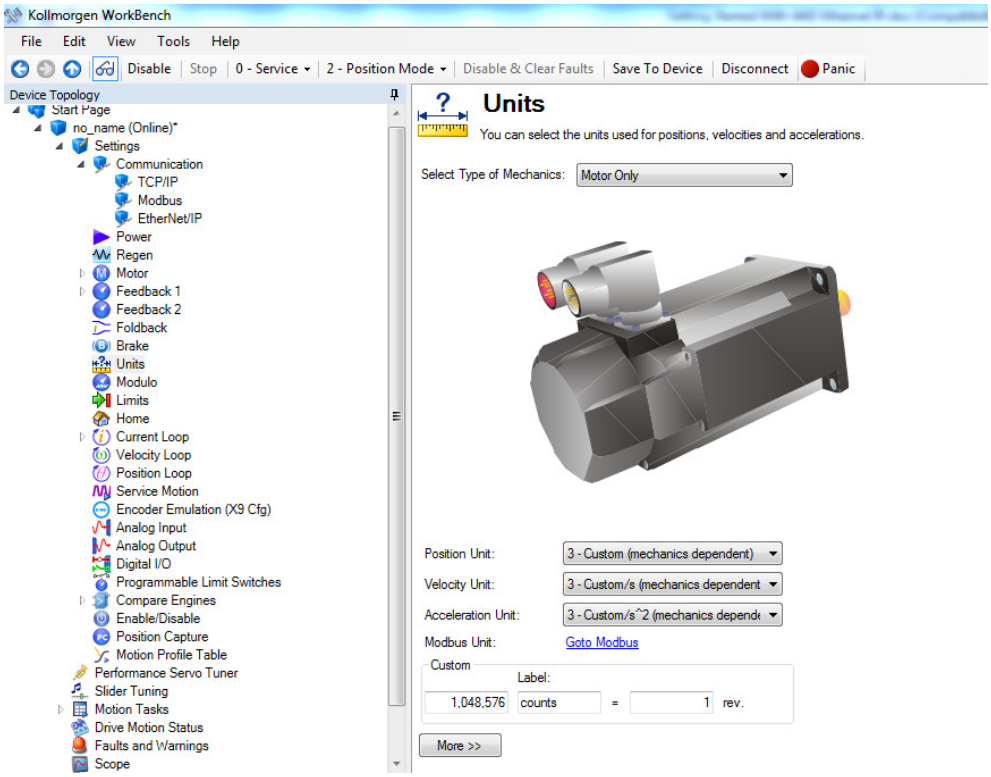
In many cases, the default scaling and resolution is adequate. However, there have been some applications that required a higher resolution than 65536 counts per motor revolution. In the following example, the desire is to increase the resolution from 65536 (2^16) counts to 1,048,576 (2^20) counts per rev. In this case, setting the EIP.POSUNIT and EIP.PROFUNIT to 4096 (2^12) will increase the resolution. As shown, the PLC math from real-world units to counts will need to change for the new scaling.



Changing the Ethernet/IP scaling.

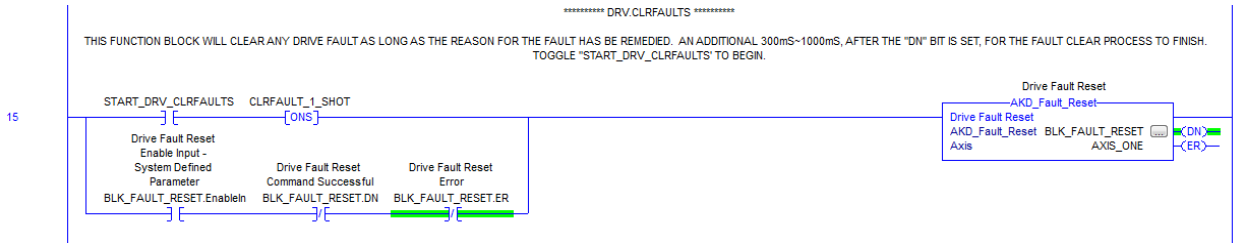


Setting Workbench units to match Ethernet/IP.

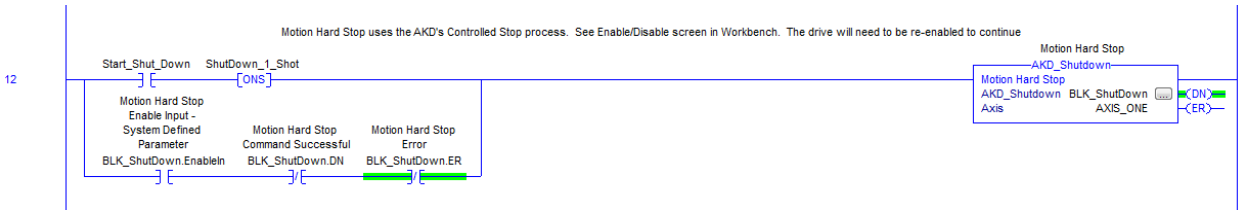


<u>STEP 8: Other AOIs demonstrated in the Sample Project:</u>

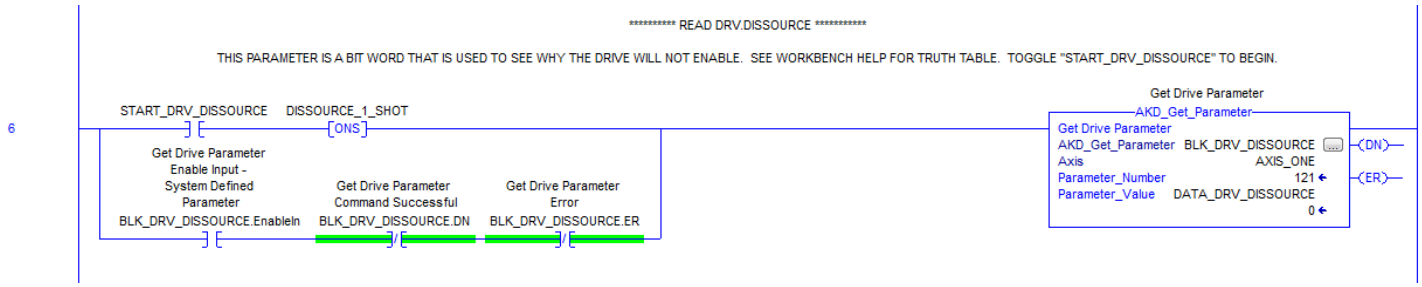
AKD_Fault_Reset:



AKD_Shutdown (Hard Stop/Controlled Stop):



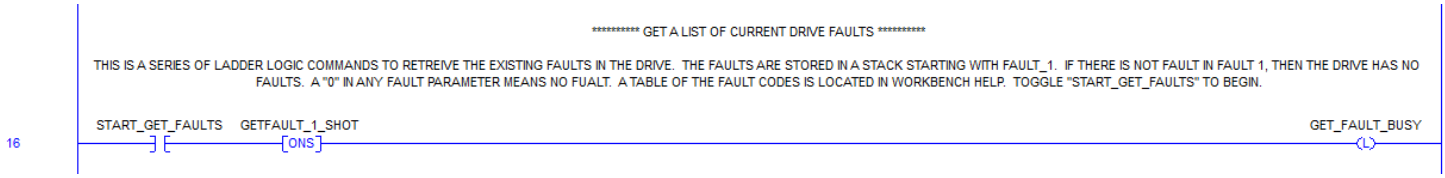
AKD_Get_Parameter (in this case, DRV.DISSOURCE).



Note from Appendix B there is a list of drive parameters and their instance values.

Instance	Parameter	Data Size	Data Type
121	DRV.DISSOURCES	2 Byte	Integer

Get A List Of Current Drive Faults code (beginning with the following rung).



STEP10: Diagnostics and Status

Once general motion is accomplished the next question is usually status and other diagnostics. Some of this information is built into the cyclic data when the axis is declared. This includes Status Words 1&2, Drive Motion Status, and Actual Position and Velocity.

6.2.3.1 Response Assemly Data Structure

Byte	Data	Comment
0	Status Word 1	Various status bits
1	Executing Block #	The index of the Motion Task which is currently being executed
2	Status Word 2	Various status bits
3	Response Type	Specifies the response type of this assembly, echoing the Response Type set in the command assembly.
4-7	Data	The response data for most Response Types*
8-11	Position	Actual Position*
12-15	Velocity	Actual Velocity*
16-19	Motion Status	Status bits. This provides the status word DRV.MOTIONSTAT. See the Parameter Reference Guide.
20-23	Reserved	
24-31	Parameter/Attribute Data	Response Data for Command Type 0x1F (Set Parameter) and the Attribute to Get*
32	Attribute to Get	Mirrors the Attribute to Get from the Command Assembly. If non-zero, the data will be in the Parameter Data field.
33	Map Type	0: Static Map (only bytes 0 to 35 are sent) 1: Custom Map 1 2: Dynamic Map (bytes 36-63 are dynamically configurable)
34-35	Reserved	
36-63	Response Dynamic Map	See EIP.RSPMAP (➜ p. 34).

* Least significant byte first for all data fields

Status 1, Status 2, Actual Position, Actual Velocity, and Motion Status data are updated in every response assembly.

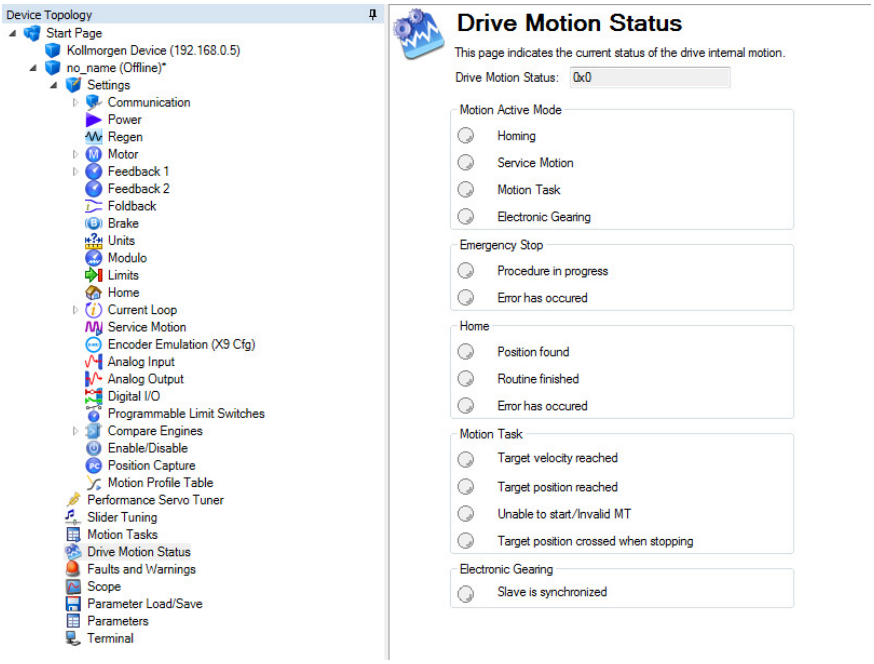
6.2.3.2 Status Word 1

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable State	Reserved	Homed	Current Direction	General Fault	In Position	Block in Execution	In Motion

6.2.3.3 Status Word 2

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2	Load Complete	Reserved	Reserved	Neg SW Limit	Pos SW Limit	Neg HW Limit	Pos HW Limit	Reserved

The Drive Motion Status information is the same as seen in Workbench. Also see Workbench Help for DRV.MOTIONSTAT for more details.



The sample project demonstrates how to access some of the information in the Response assembly for a given axis.

The rungs above reference status bits under the declared axis “AXIS_ONE”. From the controller tag database for AXIS_ONE. Note the full 64 byte command and response assemblies (AXIS_ONE.Output and AXIS_ONE.Input) as well as AXIS_ONE.PositionFeedback and AXIS_ONE.VelocityFeedback are available from the controller tags.

⊟	AXIS_ONE	{...}	{...}		AKD_Axis	Axis Data:
⊞	AXIS_ONE.Control	{...}	{...}		AKD_Control	Axis Data: Control bits to send to the drive
⊞	AXIS_ONE.Status	{...}	{...}		AKD_Status	Axis Data: Status bits received from the drive
⊞	AXIS_ONE.Input	{...}	{...}		AB:ETHERNET_MODULE_SINT_64B...	Axis Data: Data from the drive
⊞	AXIS_ONE.Output	{...}	{...}		AB:ETHERNET_MODULE_SINT_64B...	Axis Data: Data to the drive
⊞	AXIS_ONE.ResponseMsgType	0		Decimal	SINT	Axis Data: Response type contained in latest IO res...
⊞	AXIS_ONE.CommandTimeout	0		Decimal	INT	Axis Data: Time to allow for command response from...
⊞	AXIS_ONE.PositionFeedback	19466		Decimal	DINT	Axis Data: Actual Position Value
⊞	AXIS_ONE.VelocityFeedback	-67		Decimal	DINT	Axis Data: Actual Velocity Value

Expanding AXIS_ONE.Status the following information is given.

⊟ AXIS_ONE.Status	{...}	{...}		AKD_Status	Axis Data: Status bits received from the drive
AXIS_ONE.Status.Profile_In_Progr...	0		Decimal	BOOL	Axis Data: Profile Move In Progress
AXIS_ONE.Status.Block_In_Execu...	0		Decimal	BOOL	Axis Data: Block In Execution
AXIS_ONE.Status.On_Target_Posi...	0		Decimal	BOOL	Axis Data: On Target Position (1=Current Position Equals Last Target Position)
AXIS_ONE.Status.General_Fault	1		Decimal	BOOL	Axis Data: General Fault
AXIS_ONE.Status.Current_Direction	1		Decimal	BOOL	Axis Data: Current Motor Direction (0=Reverse, 1=Forward)
AXIS_ONE.Status.Home_Level	1		Decimal	BOOL	Axis Data: Level of Home Input
AXIS_ONE.Status.Reg_Level	0		Decimal	BOOL	Axis Data: Level of Registration Input
AXIS_ONE.Status.Enable	0		Decimal	BOOL	Axis Data: State of Enable Output
AXIS_ONE.Status.Fault_Input_Fault	0		Decimal	BOOL	Axis Data: Fault Input Active
AXIS_ONE.Status.Fwd_Limit	0		Decimal	BOOL	Axis Data: Forward Input Active
AXIS_ONE.Status.Rev_Limit	0		Decimal	BOOL	Axis Data: Reverse Input Active
AXIS_ONE.Status.Positive_Limit	0		Decimal	BOOL	Axis Data: Positive Software Limit Exceeded
AXIS_ONE.Status.Negative_Limit	0		Decimal	BOOL	Axis Data: Negative Software Limit Exceeded
AXIS_ONE.Status.FE_Fault	0		Decimal	BOOL	Axis Data: Following Error Fault
AXIS_ONE.Status.Block_Fault	0		Decimal	BOOL	Axis Data: Block Execution Fault
AXIS_ONE.Status.Load_Complete	0		Decimal	BOOL	Axis Data: Command Message Successfully Loaded

Likewise, note if the ladder needs to access the actual Position Feedback, the AXIS_ONE.PositionFeedback can either used using that tag or copied to another tag.