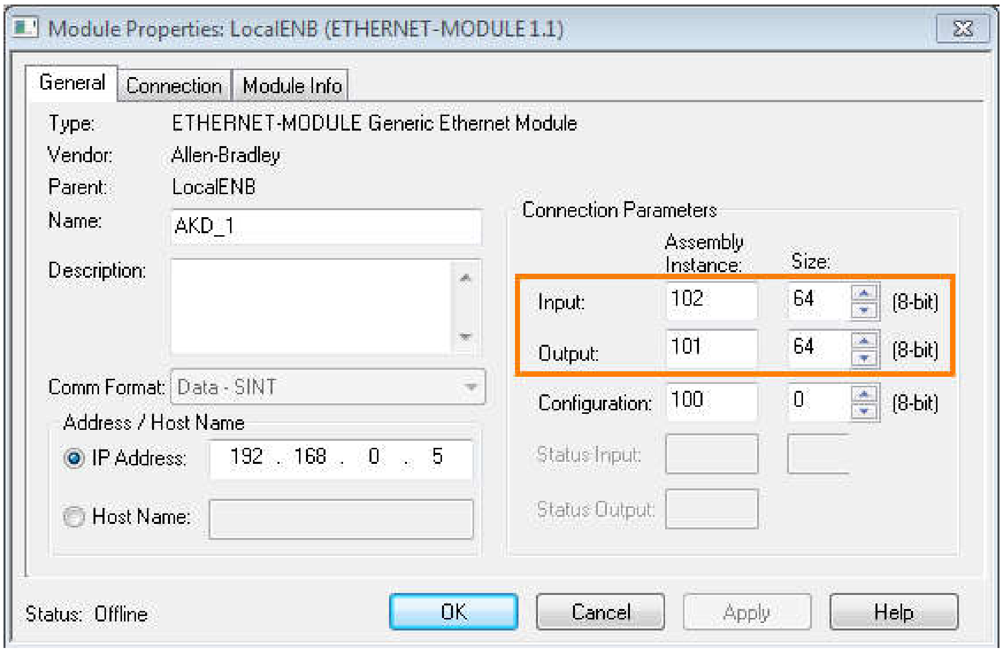
AKD Ethernet IP: Diagnostics and Dynamic Mapping Rev B 2-27-2019

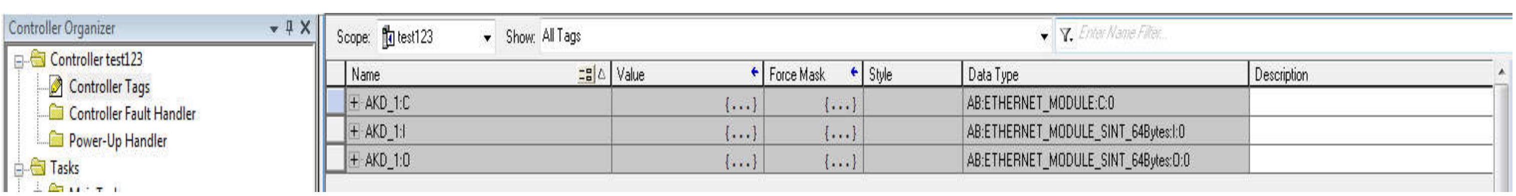
One question that comes up often is how to read parameters for information (i.e. position feedback) and diagnostics beyond the Sample Project provided.

For users of Compactlogix and Contrologix, cyclic data is supported with the AKD Ethernet IP drive (note the Micrologix 1400 does not support cyclic data and parameter read/writes must be achieved explicitly via the MSG blocks. This application note is specific to cyclic data.

Per the AKD Ethernet IP manual there is the command assembly and the response assembly both 0 to 63 (total 64 bytes). How this correlates to the PLC and drive is the AKD is setup as a Generic Ethernet Module in the AB PLC. Note there are 64 bytes for the input and 64 bytes for the output for the assemblies.



The configuration of the Generic Ethernet Module added the following to the controller tags:



Static vs. Dynamic Mapping

It is important to note there are portions of the assemblies that are static (preconfigured) and dynamic (flexible; not preconfigured). For example, below shows color coded portions of the command assembly. Bytes 0-32 are static and preconfigured. This mapping cannot be changed. Byte 33 is the Map Type. By default the Map Type is 0 which means only the static portion of both the Command and Response Assemblies are on the cyclic data. Per the description if Byte 33 in the command assembly is set to a value of 2 it “enables” Bytes 36-63 to be dynamically configured. Parameters dynamically mapped to the Command and Response Assemblies in the AKD drive will not be written to or read from unless the Map Type is 2: Dynamic.

Static Command Assembly:

6.2.2.1 Command Assembly Data Structure

Byte	Data	Comment
0	Control Word	The control word contains bits for enabling, moving, and handshaking with the drive.
1	Block #	The block number is used to start a particular Motion Task, in combination with the Start Block bit in the Control Word.
2	Command Type	Specifies the desired command to execute, such as Set Position or Set Parameter.
3	Response Type	Specifies the desired response data to return in the Response Assembly.
4-7	Data	The command data for most Command Types*
8-11	Position	Position data for Command Type 6 (Position Move)*
12-15	Velocity	Velocity data for Command Type 6 (Position Move) and 7 (Jog)*
16-19	Acceleration	Acceleration data for Command Type 6 (Position Move) and 7 (Jog)*
20-23	Deceleration	Deceleration data for Command Type 6 (Position Move) and 7 (Jog)*
24-31	Parameter/Attribute Data	Command Data for Command Type 0x1B (Set Position Controller Attribute) and 0x1F (Set Parameter)*
32	Attribute to Get	Index of desired Position Controller Attribute value to return in the Response Assembly bytes 24-31)

Map Type and Dynamic Command Assembly:

The Map Type in the command assembly determines what bytes and mapping are used.

The default is 0: Static Map where only bytes 0-35 are used.

If the PLC sets the value of byte 33 (Map Type) then this enables the dynamic mapping which means bytes 36-63 that are dynamically configured will also be sent in the case of the command assembly and received in the case of the response assembly. This means the data to/from the drive will be on the cyclic data updated every RPI scan.

Note the comment in Bytes 36-63 “See EIP.CMDMAP” (more on this later).

Byte	Data	Comment
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	Command Dynamic Map	See EIP.CMDMAP (➔ p. 32).

Like the Command Assembly, the Response Assembly has 0-63 bytes. Bytes 0-35 make up the response assembly's static mapping. Byte 33: Map Type will reflect the current Map Type as set by Byte 33 in the command assembly. If Byte 33 is 0 then only the static bytes 0-35 are read. If byte 33 reads a 2 (as in the case the PLC sets byte 33 to a value of 2) then the bytes 36-63 when dynamically mapped will read the drive's parameter data every RPI scan.

Static Response Assembly:

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	

Note the comment in Bytes 36-63 "See EIP.RSPMAP" (more on this later).

Map Type and Dynamic Response Assembly:

Byte	Data	Comment
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. 42).

Using Static Mapping:

There is already important diagnostic information built in to the default static map of the response assembly without additional dynamic mapping. These are:

1. Status Words 1 and 2
2. Actual Position
3. Actual Velocity
4. Motion Status (equivalent to the AKD parameter DRV.MOTIONSTAT).

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	

If other parameters and diagnostics are desired beyond what is contained in the static assembly then these drive parameters can be mapped using 2 methods in Workbench.

1. Using the terminal in Workbench with the EIP.CMDMAP and EIP.RSPMAP keywords (these methods are described in the AKD Ethernet IP Communications manual)
2. Using the Workbench GUI under device_name(Online)->Settings->Communication->Ethernet IP.

Before proceeding there are a few details to review historically related to firmware and dynamic mapping.

Changes were made in FW 1-17-3-000 where issues were resolved so 32 bit (4 byte) instances of the 64 bit (8 byte) instances make it easier to read and write data either via the AKD_Set_Parameter or AKD_Get_Parameter AOIs which can only handle 4 bytes of data (an AOI limitation) or as in the case when you are attempting to map a parameter dynamically (8 byte mapping quickly consumes the available number of bytes for the dynamic map area (bytes 36-63) so 4 byte versions may allow you to map more parameters.

Note while the keywords were added in FW 1-13-09-000 there were issues using them which was remedied in 1-17-03-000 per below.

Initially added but with issues:

Version: 01-13-09-000 Release Date: June 18, 2015

Field Bus Specific Issues

New Features

- **Added 32-bit access to 64-bit wide keywords for EthernetIP. (S-15756)**
  New Feature Details:
  Added 32-bit access to 64-bit wide keywords for EthernetIP.

Resolved:

Version: 01-17-03-000 Release Date: February 28, 2018

Field Bus Specific Issues

Fixed Bugs

- **Ethernet/IP: Set 'In Motion' bit during movement in torque mode. (6052)**
  Issue:
  When the drive was set to DRV.OPMODE 0, bit 0 in Ethernet/IP Status Word 1 was never set.

  Solution:
  With DRV.OPMODE set to 0, the drive will set bit 0 in Ethernet/IP Status Word 1 as long as it has not detected zero velocity. Detection of zero velocity can be influenced using the parameters CS.TO and CS.VTHRESH.

- **Ethernet/IP: Error when adding 32-bit versions of 64-bit parameters to EIP.CMDMAP or EIP.RSPMAP (5141,D-07536)**
  Issue:
  When trying to add a 32-bit instance of a 64-bit parameter (e.g. instance 147: FB1.OFFSET (32 bit version)) to the dynamically mappable part of the Ethernet/IP command or response assembly via EIP.CMDMAP or EIP.RSPMAP, an error was returned, stating that the address to be mapped is invalid.

  Solution:
  The 32-bit instances of 64-bit parameters can now be added to the dynamically mappable part of the Ethernet/IP command or response assembly.

- **Ethernet/IP: Issues writing negative values to 32-bit instances of 64-bit parameters (4835,D-07536)**
  Issue:
  Negative values written to 32bit instances of 64bit parameters (e.g. instance 147: FB1.OFFSET (32 bit version)) were not properly extended to 64 bit, leading to large positive values being set instead. This was happening when writing via explicit messaging, Command Type 0x1F and dynamically mapped instance.

  Solution:
  Writing a negative value to 32-bit instances of 64-bit parameters now sets the correct value.

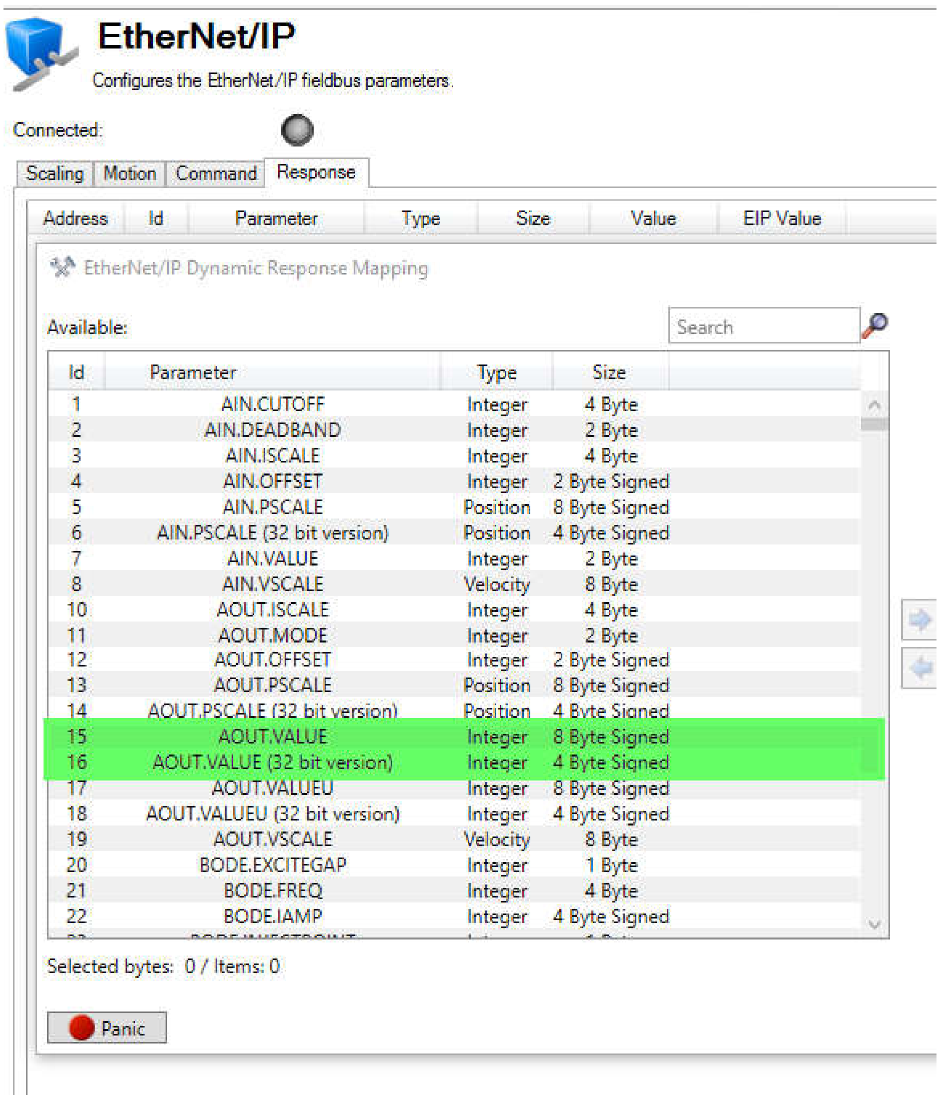
Also related in the same FW release:

New Features

- **CANopen-objects for DRV.WARNING1-3 and DRV.FAULT1-10 were missing on AKD-C. (9601,S-18760)**
  Solution:
  Add support of DRV.WARNING1-3 and DRV.FAULT1-10 on AKD-C (objects 2000h and 2001h).

- **Ethernet/IP: Added missing 32-bit instances of 64-bit parameters (D-07536)**
  Solution:
  Several 64-bit parameters did not have 32-bit instances, causing issues for Ethernet/IP masters unable to handle 64-bit instances. Those missing instances have been added. To be consistent with 32-bit instances always being on the index after their 64-bit counterpart, additional 64-bit instances have also been created. The new instances are located on indices 1071 to 1116.

Notice in the Workbench GUI under Communication->Ethernet/IP in the listing for example there are 2 instances of AOUT.VALUE: The original instance ID15 which is 8 byte signed and the 32 bit version (4 byte signed) ID 16. You can check in the listing for other parameters with both versions and instances.



It is also worth mentioning the Parameter Listings in the manuals of all the available instances will sometimes shows a Data Type of “Float” but in the Workbench GUI Dynamic Mapping List will shot it as “Integer”. Note the details below in regards to floating point values over Ethernet/IP. Also note the Parameter Listing table in the manual (shown below) doesn’t list the 32 bit version instances but indicates that to determine that ID take the ID of the 8 byte version and add 1 to it. Example: AIN.PSCALE, 8 byte instance 5. Since 5+1=6 then the AIN.PSCALE (32 bit version) will be ID6. If you refer to the screenshot of the Workbench Dynamic Mapping List above you can verify this is the case.

12 Appendix B: Parameter Listing

The parameters in this list correspond to drive parameters available in Workbench and are described in the Workbench help documentation and the AKD User's Guide.

Position values are scaled according to EIP.PROSUNIT.

Velocity and Acceleration values are scaled according to EIP.PROFUNIT.

Other floating point values are multiplied by 1000, such that a value displayed in Workbench as 1.001 will be transmitted through EtherNet/IP as 1001.

The lower 32-bits of parameters with the data size 8 can be read by accessing the instance number for the 8 byte parameter incremented by 1.

Instance	Parameter	Data Size	Data Type
1	AIN.CUTOFF	4 Byte	Float
2	AIN.DEADBAND	4 Byte	Float
3	AIN.ISCALE	4 Byte	Float
4	AIN.OFFSET	2 Byte Signed	Float
5	AIN.PSCALE	8 Byte (truncated to 4 Byte)	Position
7	AIN.VALUE	4 Byte	Float
8	AIN.VSCALE	4 Byte	Velocity
9	AIN.ZERO	Command	None
10	AOUT.ISCALE	4 Byte	Float
11	AOUT.MODE	2 Byte	Integer

There was also an issue in Workbench version 1.15 The issue was when you setup the AKD-EIP dynamic mapping the first time – you just select the parameters you want to map (in any order you want) and that all worked fined. For example – ID order 11, 1, 10.

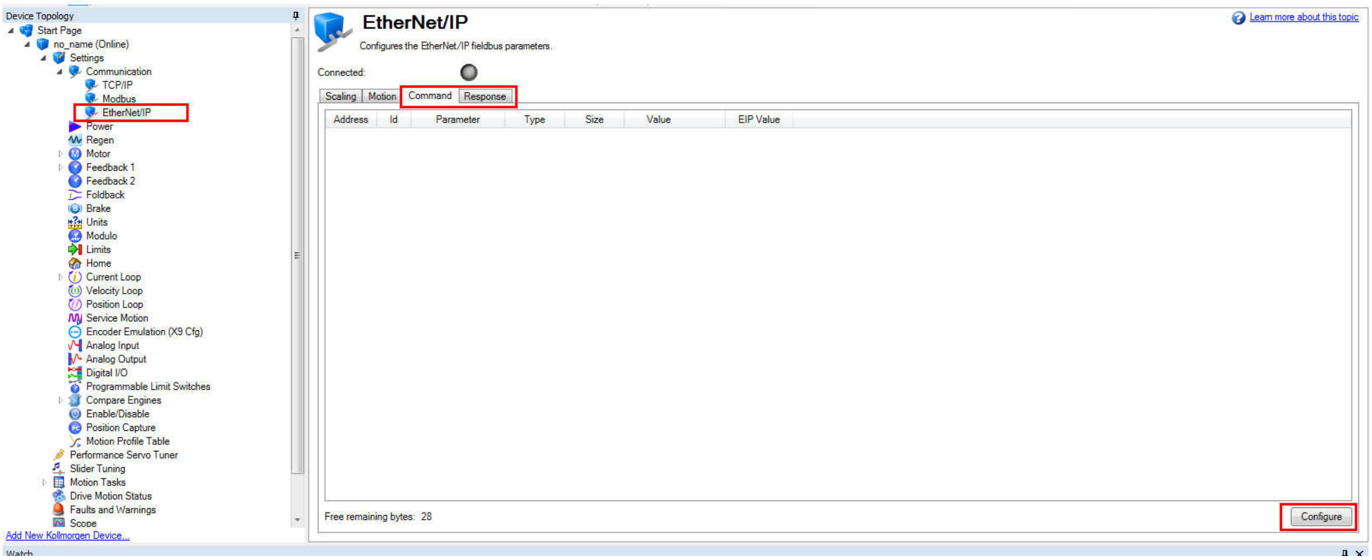
If you went back and edited the dynamic mapping (clicking “configure”) **it re-ordered the dynamic mapping “currently configured list” by lowest to highest ID number instead of keeping it in the previous order the user defined it as.** For example it showed – ID order 1, 10, 11 (ascending order).

This was resolved in Workbench version 1.16.

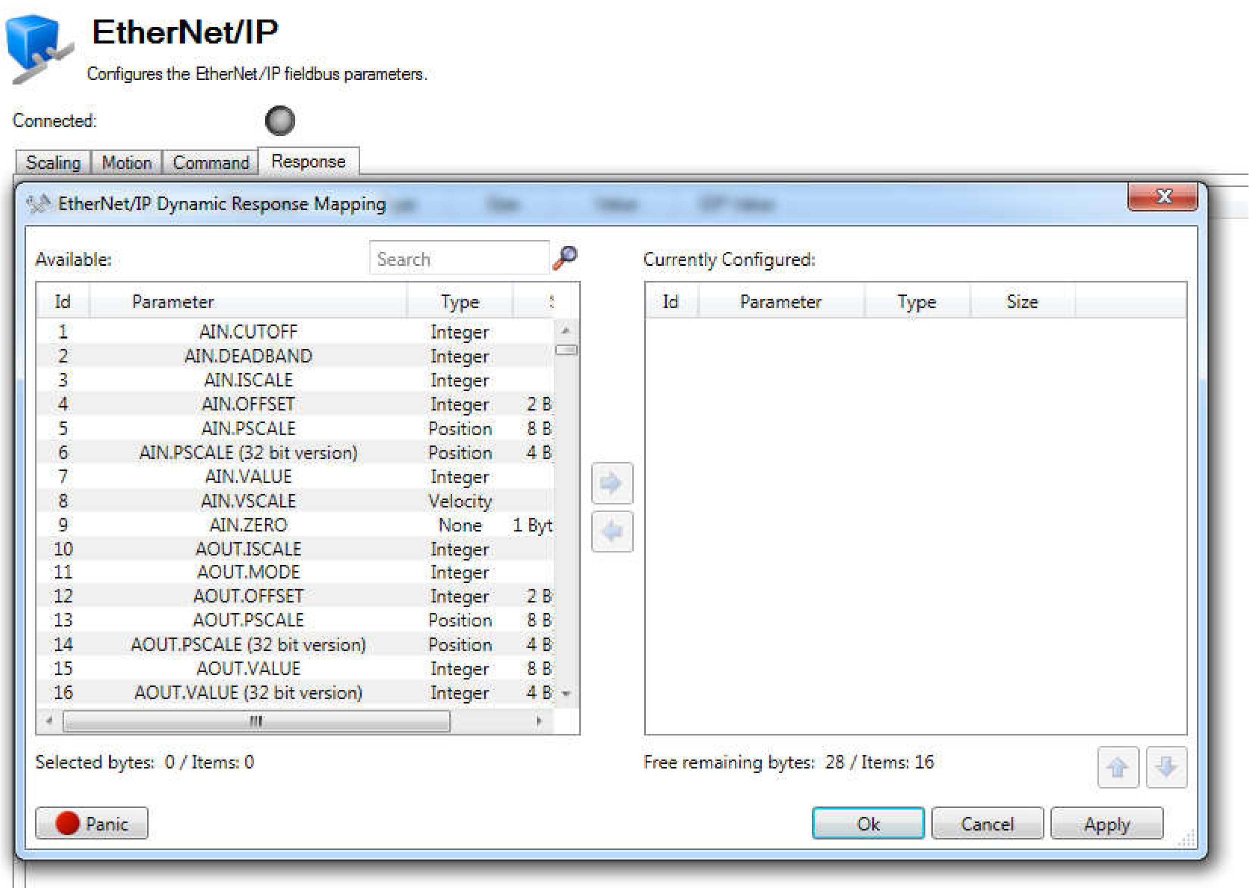
Key takeaways:

- It is best to use the latest versions of Workbench and AKD firmware on new applications.
- In general, every read/write should be tested for data validity between the PLC and the AKD by the programmer.

In this example I will be using the Workbench GUI to perform the mapping. To begin, I click on the "Configure" button. I will show only the Response mapping but the Command mapping works the same way.

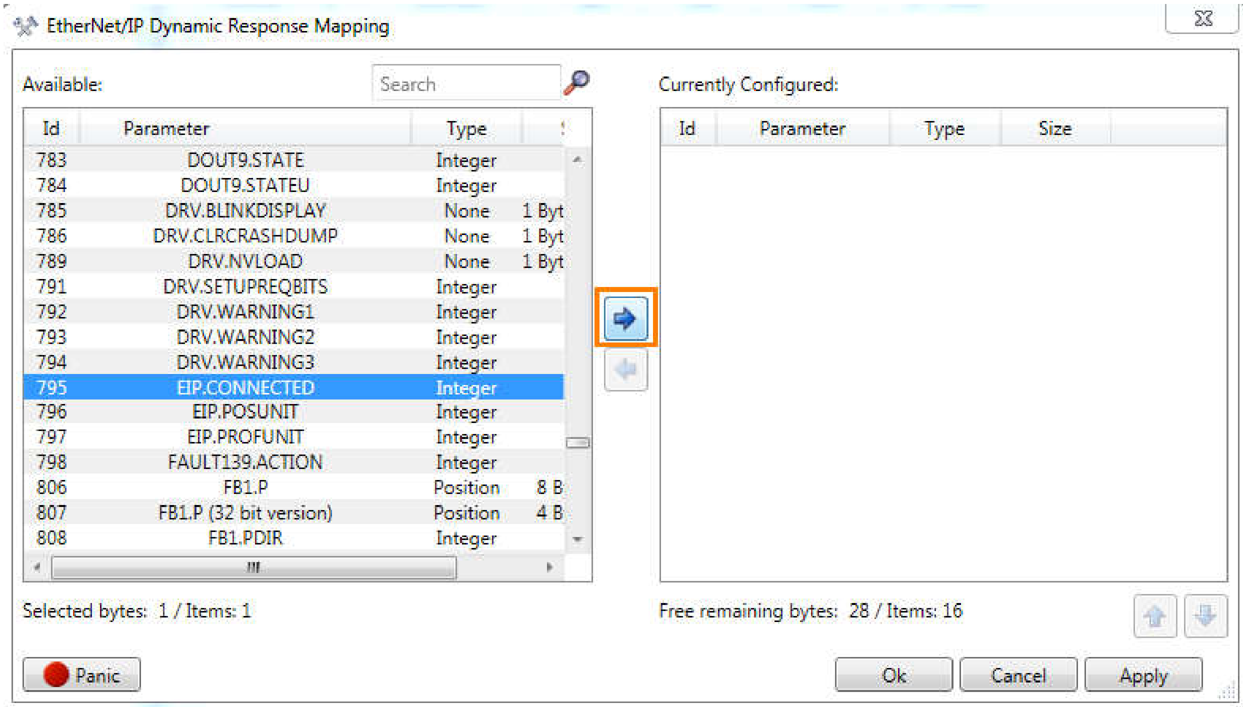


Clicking on the Response tab and then the configure button calls up the following window:



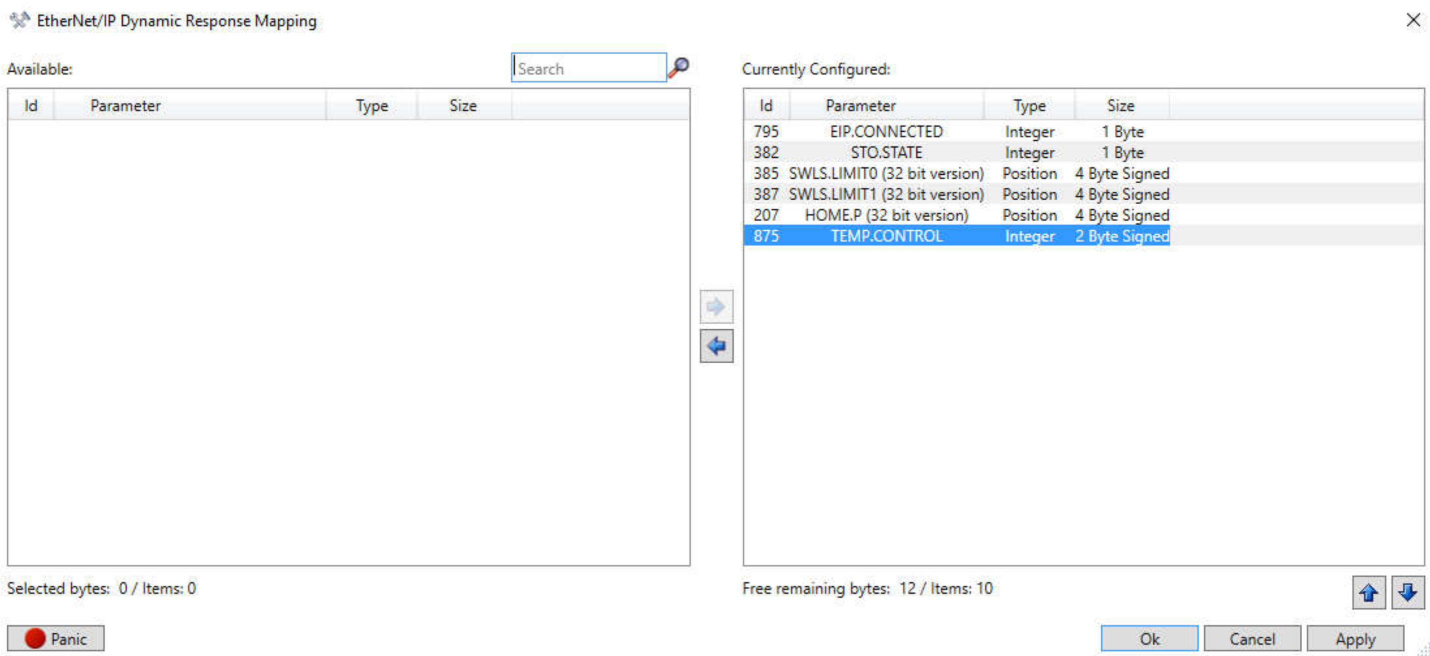
The available list of instances (ID#) with the corresponding AKD parameter name, data type, and data size (# of bytes) are given on the left. Currently configured (mapped) list with remaining free bytes and number of items. An important note is that as you add instances to the dynamic map, depending on data size, the dynamic map will begin to populate and fill bytes 36-63 according to size.

In this example, I will begin adding parameters to the dynamic map (the parameters selected in this example are arbitrary and actual mapping will be application dependent). You can scroll using the scroll bar and then highlight the desired instance (parameter) on the left and then to add to the dynamic map, click on the right arrow button:



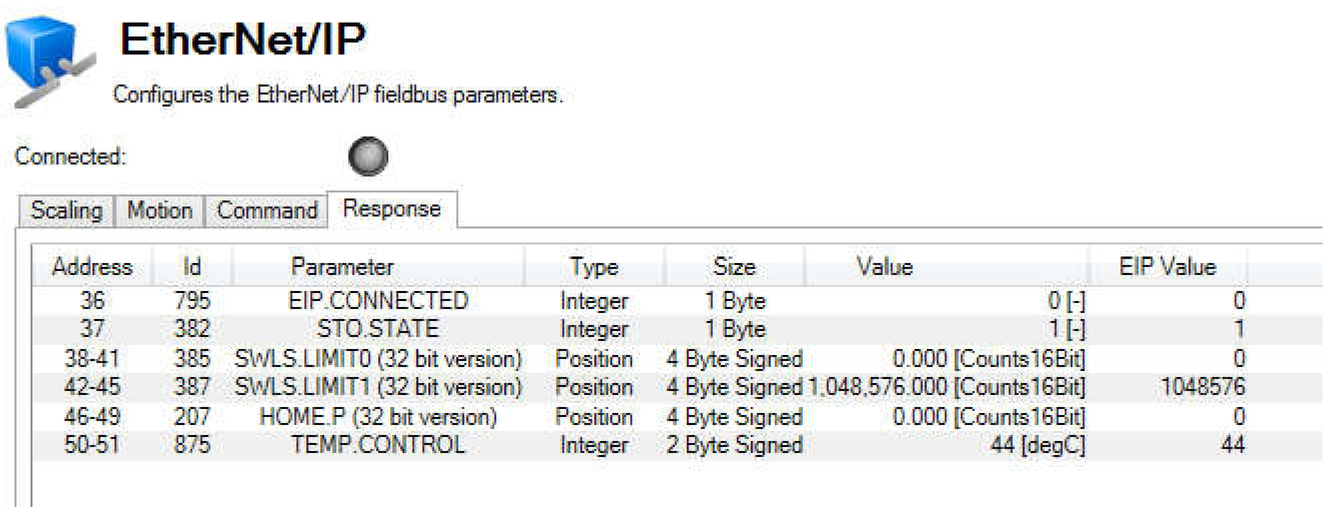
Repeat until the mapping appears as shown below:

Note that the free remaining bytes are 12 indicating how many bytes remain in the response assembly for dynamic mapping (36-63). Click Apply and Ok.



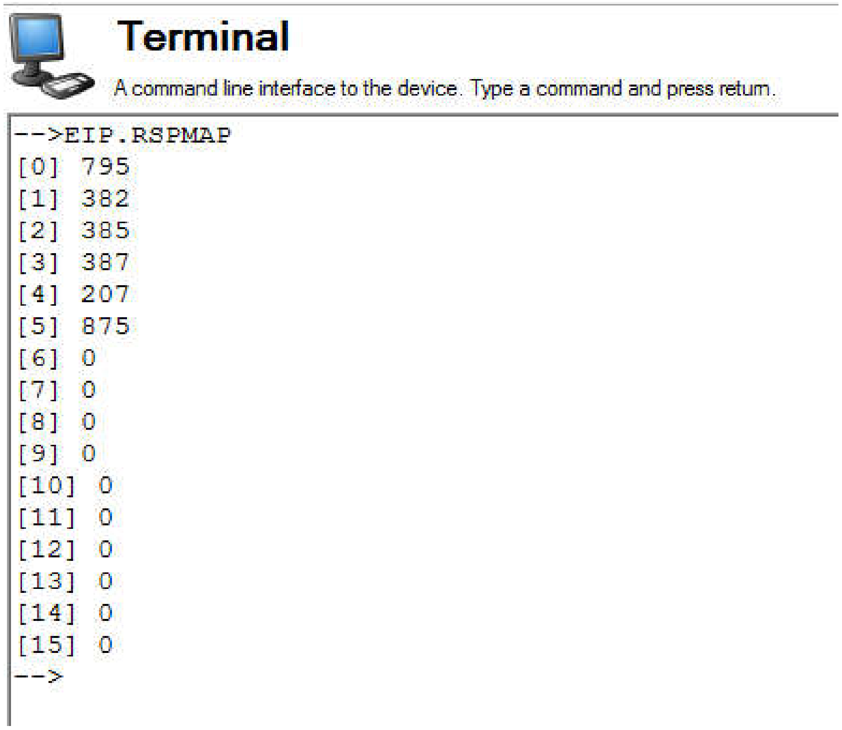
Id	Parameter	Type	Size
795	EIP.CONNECTED	Integer	1 Byte
382	STO.STATE	Integer	1 Byte
385	SWLS.LIMIT0 (32 bit version)	Position	4 Byte Signed
387	SWLS.LIMIT1 (32 bit version)	Position	4 Byte Signed
207	HOME.P (32 bit version)	Position	4 Byte Signed
875	TEMP.CONTROL	Integer	2 Byte Signed

The configuration is now as shown. Note the “Address” column indicates what bytes in the Response Assembly are used for the given ID/Parameter. This is important to note when referencing the data on the PLC side. Also notice the Value column which is the Workbench displayed value and the EIP value is the equivalent value over Ethernet IP (these are not always the same).



Address	Id	Parameter	Type	Size	Value	EIP Value
36	795	EIP.CONNECTED	Integer	1 Byte	0 [-]	0
37	382	STO.STATE	Integer	1 Byte	1 [-]	1
38-41	385	SWLS.LIMIT0 (32 bit version)	Position	4 Byte Signed	0.000 [Counts16Bit]	0
42-45	387	SWLS.LIMIT1 (32 bit version)	Position	4 Byte Signed	1,048,576.000 [Counts16Bit]	1048576
46-49	207	HOME.P (32 bit version)	Position	4 Byte Signed	0.000 [Counts16Bit]	0
50-51	875	TEMP.CONTROL	Integer	2 Byte Signed	44 [degC]	44

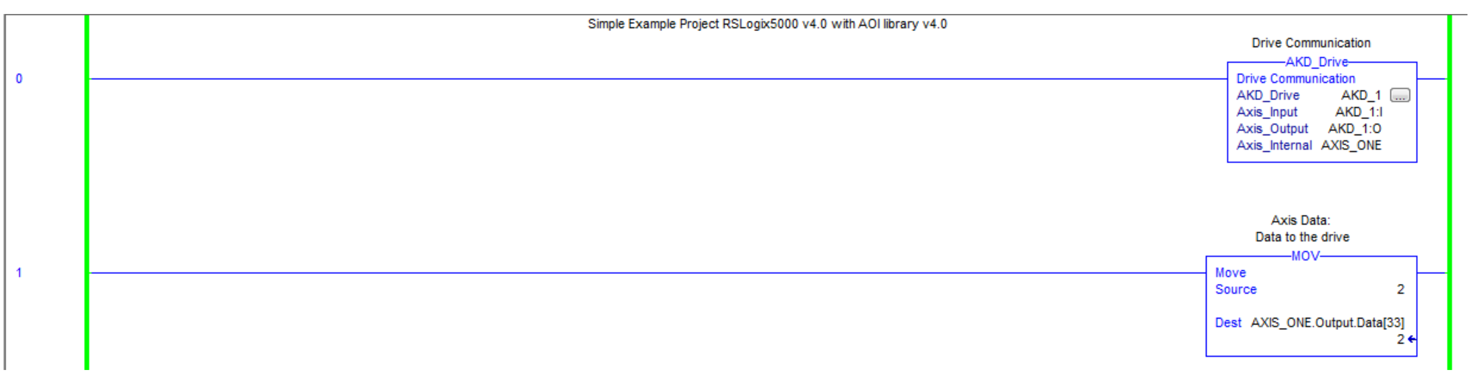
Note the GUI achieved the same mapping as if it was done in Workbench terminal:



When you've configured your mapping it is important to type a DRV.NVSAVE in the Workbench terminal or click on the "Save To Device" button on the Workbench toolbar to save the configuration to the drive's non-volatile memory.

As stated before, the cyclic data that is mapped using the EIP.CMDMAP or EIP.RSPMAP (or Communications->Ethernet IP GUI) are not active until the MAP TYPE set in the command assembly is set to a value of 2: Dynamic Map (bytes 36-63 are dynamically configurable. This essentially enables the dynamic mapped data to be passed on the RPI scan.

I added a rung (rung 1 below) that moves a value of 2 into byte 33 of the command assembly to enable the dynamic mapping and dynamic data transfer.



In order to read the values mapped in the PLC easily, I copied the data from the response assembly into tags. Note that some of the parameters I've chosen are 1, 2, and 8 bytes of data.
On the PLC side you have SINT, DINT, and LINT as choices (for these data types; for a complete listing see the table below). LINT is a 64 byte integer (8 bytes). DINT can handle 4 bytes of signed data. It is important to check that the data values from the drive over Ethernet IP will not be outside of the range of the data type for your controller tag. As mentioned before changes were made in FW 1-17-3-000 where issues were resolved so 32 bit (4 byte) instances of the 64 bit (8 byte) instances make it easier to read and write data either via the AKD_Set_Parameter or AKD_Get_Parameter AOIs which can only handle 4 bytes of data or in the case when you are attempting to map a parameter dynamically where 8 byte mapping quickly consumes the available number of bytes for the dynamic map area (bytes 36-63) so 4 byte versions may allow you to map more parameters.

In summary:

AKD Instance Data Type	AB PLC Tag Type (to copy the value into)
Command	SINT
1 Byte	SINT
1 Byte Signed	SINT
2 Byte	INT
2 Byte Signed	INT
4 Byte	DINT
4 Byte Signed	DINT
8 Byte	LINT
8 Byte Signed	LINT

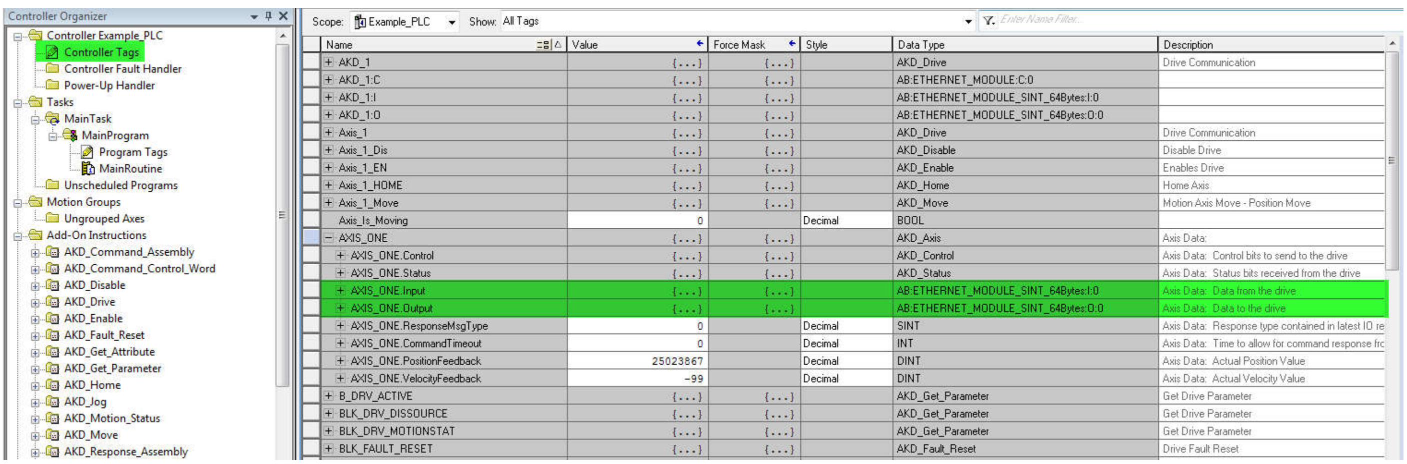
* Tag Type REAL is not applicable with the AKD Ethernet IP parameters.

Next the mapped bytes for the given axis (AKD) are copied from the mapped bytes for that parameter/instance into a tag in the ladder to hold the value (and dimension the data from bytes to SINT, INT, DINT, etc.).

Tags:

Tagname	Data Type	Bytes	# of Bytes
EIP_CONNECTED	SINT	36	1
STO_STATE	SINT	37	1
SWLS_LIMIT0	DINT	38-41	4
SWLS_LIMIT1	DINT	42-45	4
HOME_P	DINT	46-49	4
TEMP_CONTROL	INT	40-51	2

Also recall that the source tags above in the ladder logic come from:



The structure and tags above are created when you give the AKD_Drive (Drive Communication) AOI an Axis_Internal name (in this example “AXIS_ONE”). Hence the response assembly will be AXIS_ONE.Input.Data[Byte#]. If you name your axis a different name or you have multi-axes and multiple AKD_Drive blocks, reference the axis you want to dynamically map.

After downloading and running the PLC code, I setup a watch window and compared the values in the PLC with the EIP value in the Workbench GUI: