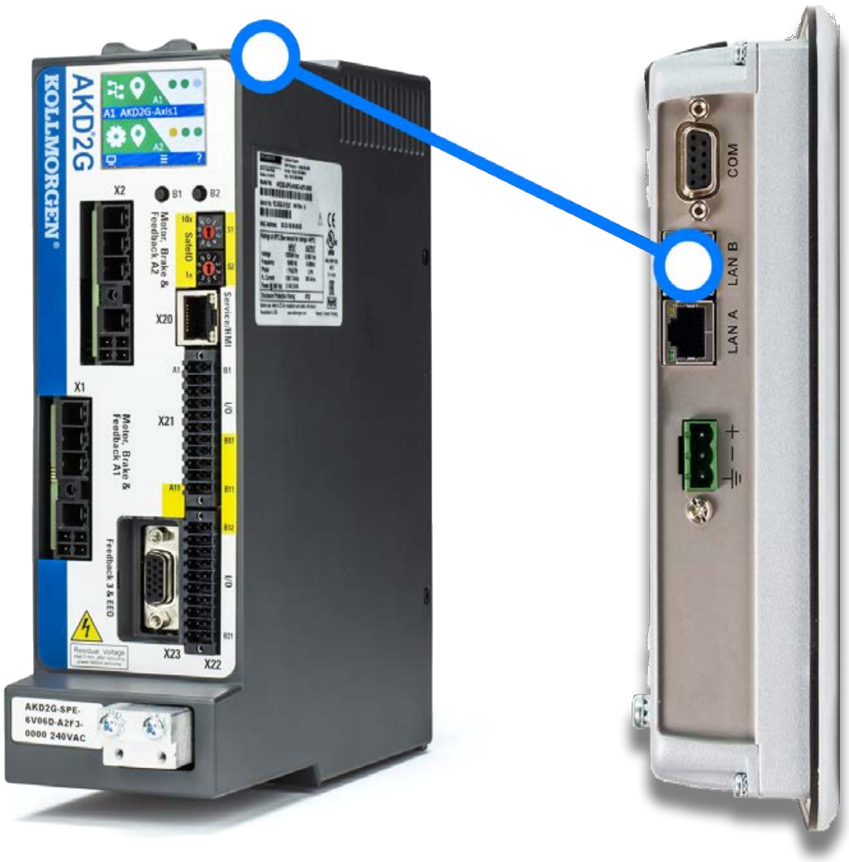
AKD®2G
Modbus Communications Manual



Edition February 2021, Revision A

Part Number 907-200010-00

Keep all manuals as a product component during the life span of the product.
Pass all manuals to future users/owners of the product.

Record of Document Revisions:

Revision	Remarks	Date
A	Preliminary Edition	2/2021

Trademarks

- AKD is a registered trademark of Kollmorgen Corporation.
- MODBUS is a registered trademark of SCHNEIDER ELECTRIC USA, INC..

1 Modbus Overview

Modbus is a communication protocol that is used for reporting data from an industrial device to an HMI or PLC by using a serial interface. Modbus TCP extends the protocol to TCP/IP networks by embedding the same Protocol Data Unit (PDU) within TCP/IP packets. The AKD2G supports Modbus TCP with up to three connections.

Most AKD2G drive parameters are supported over Modbus TCP.

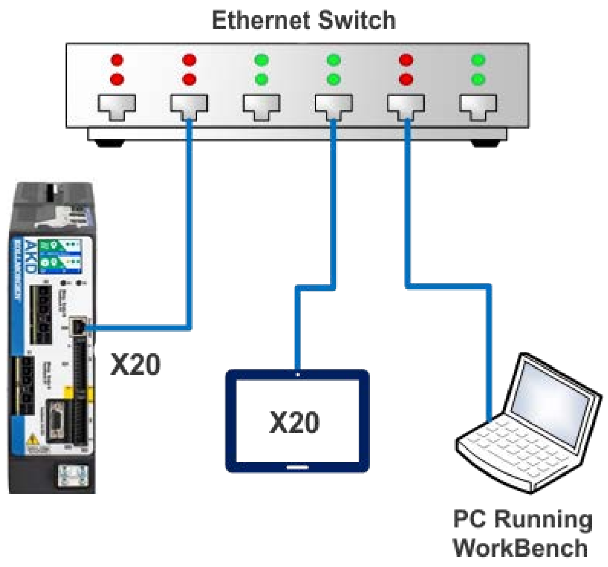
Modbus is supported on all drive variants and can be used at the same time as using another fieldbus like EtherCAT, CANopen or PROFINET.

Modbus uses the service/HMI port X20.

For information about the Modbus protocol, see: http://www.modbus.org.

2 Modbus Installation and Setup

Modbus TCP is provided over the HMI service port on the front of the drive by using the X20 connector (RJ45), the connector used for WorkBench. Connect the drive and a Modbus device such as an HMI to a working Ethernet network. For ease of testing and configuration, connect a PC running WorkBench to the same network.

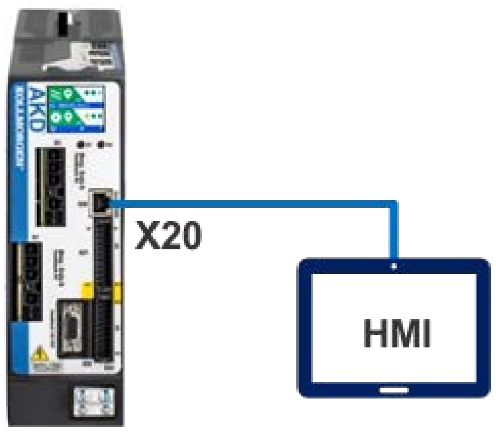


Once booted the drive displays its Ethernet IP address on the LCD display. The drive can be accessed using Modbus with this IP address and utilizing port 502. WorkBench uses the same address, but a different port number.

Once the devices are connected, the connected device can open a connection to the AKD2G using these settings:

- IP Address: read from drive display or WorkBench connect screen
- Port: 502

  If access is not required to the drives service port you can connect the AKD2G drive directly to the HMI without an Ethernet switch. In this configuration both the AKD2G drive and HMI needs to be configured with the static IP address on the same subnet.



To disable Modbus on AKD2G so the drive ignores all Modbus messages, set MODBUS.EN to 0 (disabled). By default Modbus is enabled, MODBUS.EN = 1.

3 Modbus Commands

The following commands are provided to use with the Modbus interface.

Commands	Description
MODBUS.CLRERRORS	Clears the Modbus error log.
MODBUS.DATA	Reads the Modbus register values as seen in Modbus.
MODBUS.EN	Enables or disables the Modbus interface.
MODBUS.ENDIAN	Swaps word order for 32-bit values.
MODBUS.ERRORCOUNT	The number of errors logged since the drive powered up or the error log was cleared with MODBUS.CLRERRORS.
MODBUS.ERRORMODE	Enables or disables Modbus error response messages.
MODBUS.ERRORS	Lists the last twenty entries form the Modbus error log.
MODBUS.INFO	Returns information about the active Modbus connections.
MODBUS.KEEPALIVE	Enables or disables the Modbus keepalive fault.
MODBUS.LIST	Lists all the parameters that can be mapped using MODBUS.MAP.
MODBUS.MAP	Sets the parameter mapped to each Modbus register.
MODBUS.MSGDUMP	Prints any logged Modbus messages.
MODBUS.MSGLOG	Enables or disables Modbus message logging.
MODBUS.RSTMAP	Resets MODBUS.MAP to default map.
MODBUS.WATCHDOG	The watchdog timeout in ms. 0 disables the watchdog.

4 Messaging Overview

Modbus TCP is a protocol that AKD2G uses to allow a HMI or PLC to read and write drive parameters for instance AXIS2.VL.FB or DIN.STATES. When a read request is sent to the drive, the drive returns the value of the specified parameter. When a write command is sent to the drive, the drive processes the data and returns a response stating whether the write was successful or not.

Some HMI’s can send the read and write commands for all tags on a screen to the drive at the same time. This is called a block read/write. If a read or write from an HMI register is unsuccessful it may cause the other objects on that HMI screen to not function properly.

AKD2G supports two Modbus function codes:

- Read Holding Registers. Function code 3.
- Write Multiple Registers. Function code 16.

NOTE

Modbus coils and discrete inputs are not supported.

Modbus holding registers are 16-bits in width. To access 32-bit quantities two consecutive 16-bit holding registers are used. For more information of how parameters are mapped to Modbus holding registers, see Mapping 16-Bit Parameters as 32-Bit and Mapping 32-bit Parameters and Endian.

4.1 Read Holding Registers (3)

This function code is used to read one or more registers from the drive.

Request

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server returns in the response the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol
	Length	2 Bytes	Number of following bytes including the unit identifier.
	Unit Identifier	1 Byte	Used for routing. AKD2G returns in the response the unit identifier from the request.
Data	Function Code	1 Byte	0x03
	Starting Address	2 Bytes	0 to 299. See Mapping.
	Quantity of Registers	2 Bytes	1 to 125. How many 16-bit values registers to read.

Normal Response

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server copies in the response the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol
	Length	2 Bytes	Number of following bytes including the unit identifier.
	Unit Identifier	1 Byte	Used for routing. AKD2G returns in the response the unit identifier from the request.
Data	Function Code	1 Byte	0x03
	Byte Count	1 Byte	How many data bytes follow. This is twice the Quantity of Registers from the request message. 2 × N*
	Register Value	2 × N* Bytes	Data

*N = Quantity of Registers

Error Response

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server copies in the response the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol
	Length	2 Bytes	3 = Number of following bytes including the unit identifier.
	Unit Identifier	1 Byte	Used for routing. AKD2G returns in the response the unit identifier from the request.
Data	Function Code	1 Byte	0x83
	Exception Code	1 Byte	See Modbus Error and Exception Response Codes

Example

Following is an example of reading the position loop feedback, AXIS1.PL.FB, starting at address 188 (0x00BC), with an actual value of 0x2A05 F200.

Request

Header	Transaction Identifier	nn nn
	Protocol Identifier	00 00
	Length	00 06
	Unit Identifier	FF
Data	Function	03
	Starting Address	00 BC
	Quantity of Registers	00 02

Response

Header	Transaction Identifier	nn nn
	Protocol Identifier	00 00
	Length	00 06
	Unit Identifier	FF
Data	Function	03
	Byte Count	08
	Register 588	2A 05
	Register 589	F2 00

4.2 Write Single Register (6)

This function code is used to write to one register and is supported in firmware versions from 02-05-02-000.

Request

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server returns the response of the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol.
	Length	2 Bytes	6 = Number of following bytes including the unit identifier.
	Unit Iden-tifier	1 Bytes	Used for routing. AKD2G returns in the response of the unit identifier from the request.
Data	Function Code	1 Bytes	0x06.
	Register Address	2 Bytes	0 to 299. See Mapping.
	Register Value	2 Bytes	Value that is written to the register.

Normal Response

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server returns the response of the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol.
	Length	2 Bytes	6 = Number of following bytes including the unit identifier.
	Unit Iden-tifier	1 Bytes	Used for routing. AKD2G returns the response of the unit identifier from the request.
Data	Function Code	1 Bytes	0x06.
	Register Address	2 Bytes	0 to 299. Reflecting the address from the request.
	Register Value	2 Bytes	Echo of the value in the request.

Error Response

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server returns the response of the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol.
	Length	2 Bytes	3 = Number of following bytes including the unit identifier.
	Unit Iden-tifier	1 Bytes	Used for routing. AKD2G returns the response of the unit identifier from the request.

Data	Function Code	1 Bytes	0x86.
	Exception Code	2 Bytes	See Modbus Errors and Modbus Error and Exception Response Codes .

Example

Following is an example of setting the register at address 122 (0x007A) to a value of 300000 (0x93E0).

Header	Transaction Identifier	nn nn
	Protocol Identifier	00 00
	Length	00 06
	Unit Identifier	FF
Data	Function	06
	Register Address	00 7A
	Register 122 Data	93 E0

Response

Header	Transaction Identifier	nn nn
	Protocol Identifier	00 00
	Length	00 06
	Unit Identifier	FF
Data	Function	06
	Register Address	00 7A
	Data Echo	93 E0

4.3 Write Multiple Registers (16)

This function code is used to write all registers of one drive parameter.

Request

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server returns in the response the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol
	Length	2 Bytes	Number of following bytes including the unit identifier.
	Unit Identifier	1 Byte	Used for routing. AKD2G returns in the response the unit identifier from the request.
Data	Function Code	1 Byte	0x10
	Starting Address	2 Bytes	0 to 299. See Mapping.
	Quantity of Registers	2 Bytes	1 to 123. How many 16-bit values registers to write.
	Byte Count	1 Byte	How many data bytes follow. This is twice the Quantity of Registers. 2 x N*
	Register Value	2 × N* Bytes	Data

*N = Quantity of Registers

Normal Response

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server returns in the response the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol
	Length	2 Bytes	Number of following bytes including the unit identifier.
	Unit Identifier	1 Byte	Used for routing. AKD2G returns in the response the unit identifier from the request.
Data	Function Code	1 Byte	0x10
	Starting Address	2 Bytes	0 to 299. Reflecting the address from the request.
	Quantity of Registers	2 Bytes	2 or 4

Error Response

Header	Transaction Identifier	2 Bytes	Used for transaction pairing. A Modbus server returns in the response the transaction identifier from the request.
	Protocol Identifier	2 Bytes	0 = Modbus Protocol
	Length	2 Bytes	3 = Number of following bytes including the unit identifier.
	Unit Identifier	1 Byte	Used for routing. AKD2G returns in the response the unit identifier from the request.
Data	Function Code	1 Byte	0x90
	Exception Code	1 Byte	See Modbus Error and Exception Response Codes

Example

Following is an example of setting the homing velocity, AXIS1.HOME.V, starting at address 122 (0x007A) to a value of 300000 (0x0004 93E0).

Request

Header	Transaction Identifier	nn nn
	Protocol Identifier	00 00
	Length	00 0B
	Unit Identifier	FF
Data	Function	10
	Starting Address	00 7A
	Quantity of Registers	00 02
	Byte Count	04
	Register 122	00 04
	Register 123	93 E0

Response

Header	Transaction Identifier	nn nn
	Protocol Identifier	00 00
	Length	00 06
	Unit Identifier	FF
Data	Function	10
	Starting Address	00 7A
	Quantity of Registers	00 02

4.4 Modbus Error and Exception Response Codes

If a Modbus transaction fails a code is returned indicating why it failed. AKD2G supports the following codes from the Modbus specification and additional codes for scenarios not covered by the standard codes.

Standard Codes

Code	Description	
1	Illegal Function	The received function code is not supported by the drive. AKD2G only supports function codes 3 and 16.
2	Illegal data address	The received address is not supported by the drive. AKD2G only supports addresses 40001 to 40300. If multiple registers are being accessed this code would be returned if any of the registers are beyond the supported range.
3	Illegal Data Value	When writing to the drive the value sent is incorrect. The correct range depends on which parameter is mapped to that register with MODBUS.MAP.
4	Slave Device Failure	An unexpected error occurred in the drive while processing the Modbus message.
5	Acknowledge	Not generated by AKD2G.
6	Slave Device Busy	Not generated by AKD2G.
8	Memory Parity Error	Not generated by AKD2G.
10	Gateway Path Unavailable	Not generated by AKD2G.
11	Gateway Target Device Failed To Respond	Not generated by AKD2G.

Kollmorgen Specific Codes

Code	Description	
33	Register Not Mapped	MODBUS.MAP is empty for this address.
34	Command Error	Error reading or writing this register. MODBUS.ERRORS provides further details.
35	Invalid Length	One of the following causes. 1. The received quantity of registers was zero. 2. The received quantity of registers was greater than 123 for writes or 125 for reads. 3. When processing function code 16 (write multiple registers) the received byte count was not twice the received quantity of registers.

4.5 Handling Broken Connections

AKD2G uses a watchdog timer and TCP keepalive to detect if communications between the drive and the HMI has been lost.

The Modbus watchdog fault, F7100, is generated if the drive does not receive any Modbus messages within a time window set by MODBUS.WATCHDOG. Zero disables the watchdog. The timer does not start until the first Modbus message is received. If standard process is used to disconnect no fault is generated. There is no need to write to a special register; any Modbus message resets the watchdog timer. If the drive stops receiving Modbus messages faults will occur on both axes.

The Modbus/TCP specification indicates all Modbus ports must use the TCP keepalive feature. Keepalive allows the AKD2G to detect if the connection is broken even if no Modbus reads or writes are currently being executed while the socket is open. This is achieved by injecting extra packets if no other Modbus messages are being sent. Keepalive will fault if the cable is unplugged or cut or the PLC/HMI loses power unexpectedly. The keepalive fault is number F7101. Setting MODBUS.KEEPALIVE to zero stops the keepalive fault from being generated; by default faults will occur on both axes.

4.6 High Message Rates

Some HMIs can send Modbus messages to drives at a very high rate. To allow the AKD2G drive to continue working it must regulate how fast it processes Modbus messages. If some Modbus messages are received too quickly the AKD2G slows down its response to some Modbus messages. The first few messages are replied to promptly, subsequent messages take longer to be responded to.

MODBUS.INFO reports the current message rate for each open connection and if the message rate is too high or if the message rate is throttled.

```
-->MODBUS.INFO
Connection count: 1
Connection 4 address: 192.168.1.5:61155
Connection 4 read count: 16
Connection 4 write count: 0
Connection 4 connect time: 9s
Connection 4 message rate: 1.818182 messages per second
Connection 4 high message rate: false
-->
```

4.7 Message Logging

The AKD2G drive records the contents of all Modbus messages sent to and from the drive. In addition to the contents of the packets the IP address and port number or the remote endpoint are also recorded. DRV.RUNTIME is also latched with each message.

The logged data returned by MODBUS.MSGDUMP is shown exactly as it is transferred over Ethernet. See Read Holding Registers (3) and Write Multiple Registers (16) on how this data is formatted.

The message log contains the last twenty messages sent or received. Logging is turned on and off with MODBUS.MSGLOG.

To enable logging set MODBUS.MSGDUMP = 1.

```
-->MODBUS.MSLOG 1
-->
```

To view the logged messages.

```
-->MODBUS.MSDUMP
```

```
39d:21h:30m:48s): 1 242 0 0 0 6 255 3 0 0 0 4 from 10.8.232.204:62944
39d:21h:30m:48s): 1 242 0 0 0 11 255 3 8 0 0 0 2 0 0 0 0 to
10.8.232.204:62944
39d:21h:30m:56s): 1 243 0 0 0 6 255 3 0 0 0 4 from 10.8.232.204:62944
39d:21h:30m:56s): 1 243 0 0 0 11 255 3 8 0 0 0 2 0 0 0 0 to
10.8.232.204:62944
-->
```

5 Mapping

The following table provides which parameters by default that are accessed by using Modbus. This mapping is viewed and changed using MODBUS.MAP.

```
-->MODBUS.MAP
[40001] AXIS1.PL.FB
[40002] AXIS1.PL.FB
[40003] AXIS1.VL.FBFILTERED
[40004] AXIS1.VL.FBFILTERED
[40005] AXIS1.ACTIVE
[40006] AXIS1.ACTIVE
[40007] Empty
[40008] Empty
…
-->
```

5.1 Default Map

When a AKD2G drive is received the mapping is set by default as provided in the following table. The mapping is reset back to the default mapping using MODBUS.RSTMAP or DRV.RSTVAR .

Modbus Address	Two Axis Drive	One Axis Drive
40001 – 40002	AXIS1.PL.FB (see AXIS#.PL.FB)	AXIS1.PL.FB (see AXIS#.PL.FB)
40003 – 40004	AXIS1.VL.FBFILTER (see AXIS#.VL.FBFILTER)	AXIS1.VL.FBFILTER (see AXIS#.VL.FBFILTER)
40005 – 40006	AXIS1.ACTIVE (see AXIS#.ACTIVE)	AXIS1.ACTIVE (see AXIS#.ACTIVE)
40007 – 40008	AXIS1.MOTIONSTAT (see AXIS#.MOTIONSTAT)	AXIS1.MOTIONSTAT (see AXIS#.MOTIONSTAT)
40009 – 40010	AXIS1.DISSOURCES (see AXIS#.DISSOURCES)	AXIS1.DISSOURCES (see AXIS#.DISSOURCES)
40011 – 40012	AXIS1.FAULTED (see AXIS#.FAULTED)	AXIS1.FAULTED (see AXIS#.FAULTED)
40013 – 40020	Empty	Empty
40021 – 40022	AXIS1.MOTIONCONTROL (see AXIS#.MOTIONCONTROL)	AXIS1.MOTIONCONTROL (see AXIS#.MOTIONCONTROL)
40023 – 40024	AXIS1.MT.MOVE (see AXIS#.MT.MOVE)	AXIS1.MT.MOVE (see AXIS#.MT.MOVE)
40025 – 40026	AXIS1.MT.ACC 0 (see AXIS#.MT.ACC)	AXIS1.MT.ACC 0 (see AXIS#.MT.ACC)
40027 – 40028	AXIS1.MT.DEC 0 (see AXIS#.MT.DEC)	AXIS1.MT.DEC 0 (see AXIS#.MT.DEC)
40029 – 40030	AXIS1.MT.V 0 (see AXIS#.MT.V)	AXIS1.MT.V 0 (see AXIS#.MT.V)

Modbus Address	Two Axis Drive	One Axis Drive
40031 – 40032	AXIS1.MT.P 0 (see AXIS#.MT.P)	AXIS1.MT.P 0 (see AXIS#.MT.P)
40033 – 40034	AXIS1.MT.CNTL 0 (see AXIS#.MT.CNTL)	AXIS1.MT.CNTL 0 (see AXIS#.MT.CNTL)
40035 – 40050	Empty	Empty
40051 – 40052	AXIS2.PL.FB (see AXIS#.PL.FB)	Empty
40053 – 40054	AXIS2.VL.FBFILTER (see AXIS#.VL.FBFILTER)	Empty
40055 – 40056	AXIS2.ACTIVE (see AXIS#.ACTIVE)	Empty
40057 – 40058	AXIS2.MOTIONSTAT (see AXIS#.MOTIONSTAT)	Empty
40059 – 40060	AXIS2.DISSOURCES (see AXIS#.DISSOURCES)	Empty
40061 – 40062	AXIS2.FAULTED (see AXIS#.FAULTED)	Empty
40063 – 40070	Empty	Empty
40071 – 40072	AXIS2.MOTIONCONTROL (see AXIS#.MOTIONCONTROL)	Empty
40073 – 40074	AXIS2.MT.MOVE (see AXIS#.MT.MOVE)	Empty
40075 – 40076	AXIS2.MT.ACC 0 (see AXIS#.MT.ACC)	Empty
40077 – 40078	AXIS2.MT.DEC 0 (see AXIS#.MT.DEC)	Empty
40079 – 40080	AXIS2.MT.V 0 (see AXIS#.MT.V)	Empty
40081 – 40082	AXIS2.MT.P 0 (see AXIS#.MT.P)	Empty
40083 – 40084	AXIS2.MT.CNTL 0 (see AXIS#.MT.CNTL)	Empty
40079 – 40100	Empty	Empty
40101 – 40102	DIN.STATES	DIN.STATES
40103 – 40300	Empty	Empty

Reading from addresses marked “Empty” returns a zero and any value written to these addresses are ignored. Writes to read only parameters are also ignored.

5.2 Modbus Addresses

Modbus implementations are not consistent with how addresses are shown. Here are two different ways Modbus addresses are presented.

- Modbus Network Address
  This is the 2-byte address that is part of the TCP message. For more information, see Read Holding Registers (3) and Write Multiple Registers (16) messages.
  These addresses start at zero.
  AKD2G supports network addresses in the range 0 to 299.
- Modbus HMI Address
  This representation is commonly used with HMIs and PLCs.
  This is the representation used by AKD2G.
  The address is prefixed with 40000 to indicate this address is a holding register that is accessed with function codes 3 and 16. It is common for these addresses to start at 1 not zero as used by the Modbus network address.
  AKD2G supports HMI addresses in the range 40001 to 40300.

To convert from Modbus HMI addresses to network addresses subtract 40001.

Modbus HMI Address	Modbus Network Address
40001	0
40002	1
…	
40300	299

NOTE

AKD2G uses 1-based addressing.

5.3 Editing the Map

AKD2G allows changing what parameter is read or written when accessing each Modbus address. Modbus address between 40001 and 40300 can be remapped to access different parameters. Parameters that output strings cannot be mapped.

For example to change the Modbus address 40005 to AOUT2.VALUEU use the following.

```
-->MODBUS.MAP 40005 AOUT2.VALUEU
-->
```

To remove the mapping for a Modbus address set MAP to “Empty”.

```
-->MODBUS.MAP 40005 Empty
-->
```

The mapping can be changed at any time. Use DRV.NVSAVE to keep the modified mapping after power cycling the drive.

NOTE

Not all drive parameters can be read or written over Modbus. The parameters that cannot be mapped over Modbus are mostly accessing strings or printing information, for example DRV.INFO or DRV.NAME. If mapping a drive parameter that is not mappable using MODBUS.MAP will return an error.

```
-->MODBUS.MAP 40001 DRV.INFO
Error: [0137] Parameter is not mappable.
-->
```

The MODBUS.LIST command is used to list all drive parameters that can be used with MODBUS.MAP.

```
-->MODBUS.LIST
AIN1.CUTOFF,Float,32-bit,ReadWrite
AIN1.DEADBAND,Float,16-bit,ReadWrite
AIN1.DEADBANDMODE,Unsigned,16-bit,ReadWrite
AIN1.OFFSET,Float,16-bit,ReadWrite
AIN1.VALUE,Float,16-bit,ReadOnly
AIN2.CUTOFF,Float,32-bit,ReadWrite
AIN2.DEADBAND,Float,16-bit,ReadWrite
AIN2.DEADBANDMODE,Unsigned,16-bit,ReadWrite
AIN2.OFFSET,Float,16-bit,ReadWrite
AIN2.VALUE,Float,16-bit,ReadOnly
…
-->
```

To map an array parameter use a # symbol to append the index. Array parameters, such as AXIS#.MT.ACC, are parameters where the index must be entered into the array after the parameter name.

```
-->AXIS1.MT.ACC 5
10000.170 [rpm/s]
-->MODBUS.MAP 40001 AXIS1.MT.ACC#5
-->
```

Array parameters display as multiple lines in the MODBUS.LIST.

```
-->MODBUS.LIST
…
AXIS1.MT.ACC#1,Float,32-bit,ReadOnly
AXIS1.MT.ACC#2,Float,32-bit,ReadOnly
…
AXIS1.MT.ACC#32,Float,32-bit,ReadOnly
…
-->
```

5.4 Mapping 32-bit Parameters and Endian

Many of the AKD2G drive parameters are 32-bits. Use two 16-bit Modbus registers in contiguous addresses to access a 32-bit parameter.

For example to map AXIS1.PL.FB two addresses need to be mapped to the same parameter.

```
-->MODBUS.MAP 40005 AXIS1.PL.FB
-->MODBUS.MAP 40006 AXIS1.PL.FB
-->
```

By default, all accesses are big endian so the high word is in the lower address. This example demonstrates how the data is arranged if AXIS.PL.FB is 360.

Modbus Address	MODBUS.MAP	MODBUS.DATA	
40005	AXIS1.PL.FB	0	High word
40006	AXIS1.PL.FB	360	Low word

To switch to little endian set MODBUS.ENDIAN to 1.

```
--> MODBUS.ENDIAN 1
-->
```

The data in the two addresses now has the low word in the lower address.

Modbus Address	MODBUS.MAP	MODBUS.DATA	
40005	AXIS1.PL.FB	360	Low word
40006	AXIS1.PL.FB	0	High word

If only one register is mapped for a 32-bit parameter then its only possible to read or write to the lower 16-bits of the parameter.

5.5 Mapping 16-Bit Parameters as 32-Bit

When using some HMIs with AKD2G it is convenient to have every parameter appear in the Modbus map as a 32-bit value (two consecutive 16-bit Modbus registers).

If the same 16-bit parameter is mapped to two consecutive Modbus addresses the value can now be read and written as a 32-bit quantity that is sign extended and uses the current endian choice.

For example, if USER.INT1 is 5 and big endian is selected, MODBUS.ENDIAN is 0, then the data returned is:

Modbus Address	MODBUS.MAP	MODBUS.DATA	
40001	USER.INT1	0	High word
40002	USER.INT1	5	Low word

If little endian is selected, MODBUS.ENDIAN is 1, then the data returned is:

Modbus Address	MODBUS.MAP	MODBUS.DATA	
40001	USER.INT1	5	Low word
40002	USER.INT1	0	High word

If the parameter is signed the value that is returned is sign extended. For example, if USER.INT1 is -5 the value returned with big endian selected is:

Modbus Address	MODBUS.MAP	MODBUS.DATA	
40001	USER.INT1	65535 (FFFFh)	High word
40002	USER.INT1	65531 (FFFBh)	Low word

5.6 Edge Triggered Action Commands

AKD2G has a number of action commands that can be mapped over Modbus. For example AXIS2.EN, AXIS2.DIS and AXIS1.MT.MOVE.

These action commands are mappable by MODBUS.LIST and are listed by using the “ActionCommand” type.

```
-->MODBUS.LIST
…
DRV.DIS,ActionCommand,16-bit,ReadWrite
AXIS1.DIS,ActionCommand,16-bit,ReadWrite
AXIS1.EN,ActionCommand,16-bit,ReadWrite
AXIS1.CLRFAULTS,ActionCommand,16-bit,ReadWrite
AXIS1.MT.MOVE#0,ActionCommand,16-bit,ReadWrite
AXIS1.MT.MOVE#31,ActionCommand,16-bit,ReadWrite
…
-->
```

When these action commands are mapped to Modbus they are 16-bit read-write registers. Any values to these registers can be written and when the value is read back the last value that was written is provided. The action is performed on a rising edge if the previous value was zero. Any non-zero value can be written to trigger the action. Repeatedly writing a non-zero will only trigger the action on the first non-zero write. To trigger the action again a zero needs to be written first.

Action commands are queued and are executed in the order they are issued. If a long command is issued (for instance AXIS#.CLRFAULTS) followed by AXIS#.EN, the enable is not issued until the clear faults command is completed. If an ActionCommand fails it is logged to MODBUS.ERRORS.

For example to clear the faults on axis 2 over Modbus it is possible to map the action command to a Modbus register. Initially the register starts at zero and is set to zero if the mapping changes. If 1 is written to the Modbus register, AXIS2.CLRFAULTS is executed on the drive.

```
-->MODBUS.MAP 40100 AXIS2.CLRFAULTS
-->MODBUS.DATA 40100
0
-->
```

5.7 USER.INT1 to USER.INT10

Sometimes there are scenarios where there is a need to perform actions that cannot be achieved with a single parameter mapped over Modbus. AKD2G has a set of 10 scratchpad variables, USER.INT#, that users can read and write to over Modbus. The action table is used to start complex operations that are triggered when USER.INT# changes.

6 Modbus Scaling

When parameters are accessed over Modbus there are several scaling options: signed, unsigned and float. MODBUS.LIST defines the type of scaling is used for each parameter that is accessed over Modbus.

NOTE

When reading and writing parameters over Modbus they are scaled similarly in WorkBench.

6.1 Signed and Unsigned

Signed and unsigned are integers where the value that is passed over Modbus is the same value as displayed in WorkBench.

For example if AXIS1.DISMODE is set to 2 in WorkBench then 2 is read or written over Modbus.

6.2 Float

These are floating point numbers. To use the fractional part the value is accessed over Modbus 1000 times the value displayed in WorkBench. These parameters are always signed when access is through Modbus.

Example: if AXIS1.PL.KP is 3.270 in WorkBench then 3270 is read or written over Modbus.

This example demonstrates how axis units affect positions, velocities and accelerations. If AXIS1.PL.CMD is 180 degrees with the units set to degrees, AXIS1.UNIT.PROTARY is 2, then 180000 is read over Modbus.

Caution should be taken when using AXIS#.UNIT Parameters. It is possible to set AXIS#.UNIT Parameters so that they do not work as expected when reading or writing these parameters.

Example: if the units for position is counts, AXIS#.UNIT.PROTARY is 0. Counts exceeds the 32-bit range accessible by using Modbus after one revolution. Setting AXIS#.UNIT.PROTARY to 2 sets the degrees and the position range accessible with the 32-bit Modbus register as ±2147483 degrees and approximately 5965 revolutions.

When reading 32-bit parameters and the value exceeds 32-bits the value is truncated and the higher bits are lost. Values larger than 32-bits cannot be written, if a 32-bit value is written the value is sign extended.

7 Modbus Errors

Up to twenty Modbus errors are stored in the Modbus error list (MODBUS.ERRORS). If more errors occur, the oldest error is dropped, and the new error is stored at the end of this list. MODBUS.ERRORCOUNT is the number of errors logged since the drive powered up or the log was cleared with MODBUS.CLRERRORS. All Modbus errors are stored regardless of the state of the error mode (MODBUS.ERRORMODE).

A block read/write request is always fully executed. If a register access causes an error, the error is added to the error list and processing continues with the next register.

If MODBUS.ERRORMODE is set to 1, the drive will not return an error response. Check MODBUS.ERRORS to verify that the last request was successfully completed. If it returns 0, the request was successful.

Modbus Error Parameters	Description
MODBUS.ERRORCOUNT	The number of errors logged since the drive powered up or the error log was cleared with MODBUS.CLRERRORS.
MODBUS.ERRORMODE	Enables or disables error response messages: 0: Send error response messages 1: Do not send error response messages (default)
MODBUS.ERRORS	Lists up to twenty Modbus errors. Each entry contains the Modbus address and the error code of the failed Modbus request.
MODBUS.CLRERRORS	Clears all errors stored in MODBUS.ERRORS.

The WorkBench terminal displays the list of error messages:

```
-->MODBUS.ERRORCOUNT
2
-->MODBUS.ERRORS
(39d:21h:30m:48s) Write 40002: Modbus error code 34 (Command Error),
Error 7 (Command is read-only)
(39d:21h:45m:22s) Read 45001: Modbus error code 38 (Length Invalid)
-->
```

8 AKD2G vs. AKD: Modbus

All Modbus addresses are dynamically mapped using parameter names with MODBUS.MAP.

AKD2G uses 1-based addressing vs. AKD uses zero-based.

The optional Modbus specific scaling that was supported in addition to WorkBench units was removed; WorkBench units are used for all Modbus reads and writes.

The default for MODBUS.ERRORMODE changed from 0 to 1. The AKD2G Modbus error log on contains more information than the AKD drive. The AKD2G Modbus error log is no longer readable with Modbus registers and MODBUS.ERRORS must be used.

AKD2G Dynamic mapping can be changed at any time with MODBUS.MAP. For AKD MODBUS.DYNMAP must be set before changing the map and MODBUS.DYNMAP must be cleared when finished.

Dynamic mapping can only be changed by using WorkBench when using the Modbus screens or terminal. AKD allows the mapping to be changed by using the Modbus register interface in addition to WorkBench.

The following table provides the equivalent Modbus parameters on AKD2G compared to AKD.

AKD	AKD2G	Notes
	MODBUS.EN	
	MODBUS.INFO	
MODBUS.ADDR	MODBUS.MAP	Uses different syntax
MODBUS.CLRDYNMAP	MODBUS.RSTMAP	
	MODBUS.DATA	
	MODBUS.ENDIAN	
	MODBUS.WATCHDOG	
	MODBUS.KEEPALIVE	
	MODBUS.LIST	
MODBUS.DYNMAP		No longer needed with MODBUS.MAP
MODBUS.CLRERRORS	MODBUS.CLRERRORS	
MODBUS.DIO		Map DIN.STATES, DOUT.STATES, DOUT#STATEU, DIO.STATES, DIO#.STATEU
MODBUS.DRV		Map AXIS#.MOTIONCONTROL, AXIS#.EN, AXIS#.DIS, AXIS#.STOP
MODBUS.DRVSTAT		Map AXIS#.MOTIONSTAT, AXIS#.ACTIVE, AXIS#.SAFE.STO.ACTIVE, AXIS#.HWLS.POSSTATE, AXIS#.HWLS.NEGSTATE, AXIS#.SWLS.STATE
MODBUS.ERRORMODE	MODBUS.ERRORMODE	
MODBUS.ERRORS	MODBUS.ERRORS	

AKD	AKD2G	Notes
	MODBUS.ERRORCOUNT	
MODBUS.HOME		Map AXIS#.MOTIONCONTROL, AXIS.HOME.MOVE, AXIS#.HOME.SET
MODBUS.MOTOR		Map AXIS#.MOTIONCONTROL, AXIS#.MOTOR.BRAKE, AXIS#.MOTOR.BRAKECONTROL
MODBUS.MSGDUMP	MODBUS.MSGDUMP	
MODBUS.MSGLOG	MODBUS.MSGLOG	
MODBUS.MT		Map AXIS#.MT.MOVE
MODBUS.PIN MODBUS.POUT MODBUS.PSCALE MODBUS.SCALING		Use AXIS#.UNIT parameters
MODBUS.SM		Map AXIS#.MOTIONCONTROL, AXIS#.SM/.MOVE, AXIS#.STOP

About KOLLMORGEN

Kollmorgen is a leading provider of motion systems and components for machine builders. Through world-class knowledge in motion, industry-leading quality and deep expertise in linking and integrating standard and custom products, Kollmorgen delivers breakthrough solutions that are unmatched in performance, reliability and ease-of-use, giving machine builders an irrefutable marketplace advantage.

Join the Kollmorgen Developer Network for product support. Ask the community questions, search the knowledge base for answers, get downloads, and suggest improvements.

North America
KOLLMORGEN
201 West Rock Road
Radford, VA 24141, USA

Web:	www.kollmorgen.com
Mail:	support@kollmorgen.com
Tel.:	+1 - 540 - 633 - 3545
Fax:	+1 - 540 - 639 - 4162

Europe
KOLLMORGEN Europe GmbH
Pempelfurtstr. 1
40880 Ratingen, Germany

Web:	www.kollmorgen.com
Mail:	technik@kollmorgen.com
Tel.:	+49 - 2102 - 9394 - 0
Fax:	+49 - 2102 - 9394 - 3155

South America
KOLLMORGEN
Avenida João Paulo Ablas, 2970
Jardim da Glória, Cotia – SP
CEP 06711-250, Brazil

Web:	www.kollmorgen.com
Mail:	contato@kollmorgen.com
Tel.:	+55 11 4615-6300
Fax:	+1 - 540 - 639 - 4162

China and SEA
KOLLMORGEN
Room 302, Building 5, Lihpao Plaza,
88 Shenbin Road, Minhang District,
Shanghai, China.

Web:	www.kollmorgen.cn
Mail:	sales.china@kollmorgen.com
Tel.:	+86 - 400 668 2802
Fax:	+86 - 21 6248 5367