

**MAC00-EC4
&
MAC00E14**

**Industrial Ethernet
expansion modules for
MAC Servo Motors**

User Manual



JVL Industri Elektronik A/S

Important User Information



Warning



The MAC series of products are used to control electrical and mechanical components of motion control systems. You should test your motion system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

Please contact your nearest JVL representative in case of technical assistance. Your nearest contact can be found on our web site www.jvl.dk

Copyright 2010, JVL Industri Elektronik A/S. All rights reserved.
This user manual must not be reproduced in any form without prior written permission of JVL Industri Elektronik A/S.
JVL Industri Elektronik A/S reserves the right to make changes to information contained in this manual without prior notice.
Similarly JVL Industri Elektronik A/S assumes no liability for printing errors or other omissions or discrepancies in this user manual.

MacTalk and MotoWare are registered trademarks

JVL Industri Elektronik A/S
Blokken 42
DK-3460 Birkerød
Denmark
Tlf. +45 45 82 44 40
Fax. +45 45 82 55 50
e-mail: jvl@jvl.dk
Internet: <http://www.jvl.dk>

Contents

1 Introduction	7
1.1 Introduction	8
1.2 Hardware introduction	10
2 General Hardware description	11
2.1 I/O descriptions	12
2.2 Connector description	16
2.3 Cable accessories	18
3 MAC00-EC4 EtherCAT® module	19
3.1 Introduction to EtherCAT®	20
3.2 Protocol specifications	21
3.3 Commissioning	24
3.4 EtherCAT® objects	28
4 MAC00-EI4 EthernetIP module	33
4.1 Introduction to EthernetIP	34
4.2 Using non cyclic messages	36
4.3 Using cyclic messages I/O-messages	38
4.4 Commissioning	39
4.5 Examples	41
4.6 Configuration using different methods	43
4.7 Using and Selecting an Ethernet switch	46
4.8 Objects accessible using Explicit messages	47
4.9 Examples of applications	50
5 Appendix	59
5.1 Technical Data	60
5.2 Motor registers	61

1.1

Introduction



Industrial Ethernet is becoming more and more popular as it offers

- Very fast response time
- Predictable delay times (deterministic protocol)
- Safe transmission of data
- In a certain extended standard Ethernet hardware switches can be used

Compared with most of the “classic” non Ethernet based protocols the industrial Ethernet offers state of the art performance.

The MAC00-Ex4 Industrial Ethernet module can be configured by the end user to a number of different Ethernet protocols, for instance

- EtherCAT
- EthernetIP
- More to come

Main Features:

- High speed communication - 100Mbps/sec.
- 2 individual ports on the module offers Daisy chaining possibility.
- Standard M12 circular industrial connectors
- 1 Digital input (24V) and 1 digital output (24V) for local use around the motor
- Multiple alternative I/O possibilities available on request (OEM applications)
- LED's for easy monitoring of operation status
- Optional encoder I/O
- Rough design
- Access to all internal motor parameters and registers possible. No need of pre-setup of the motor.
- Optional RS232 connection available for monitoring and setup use if desired

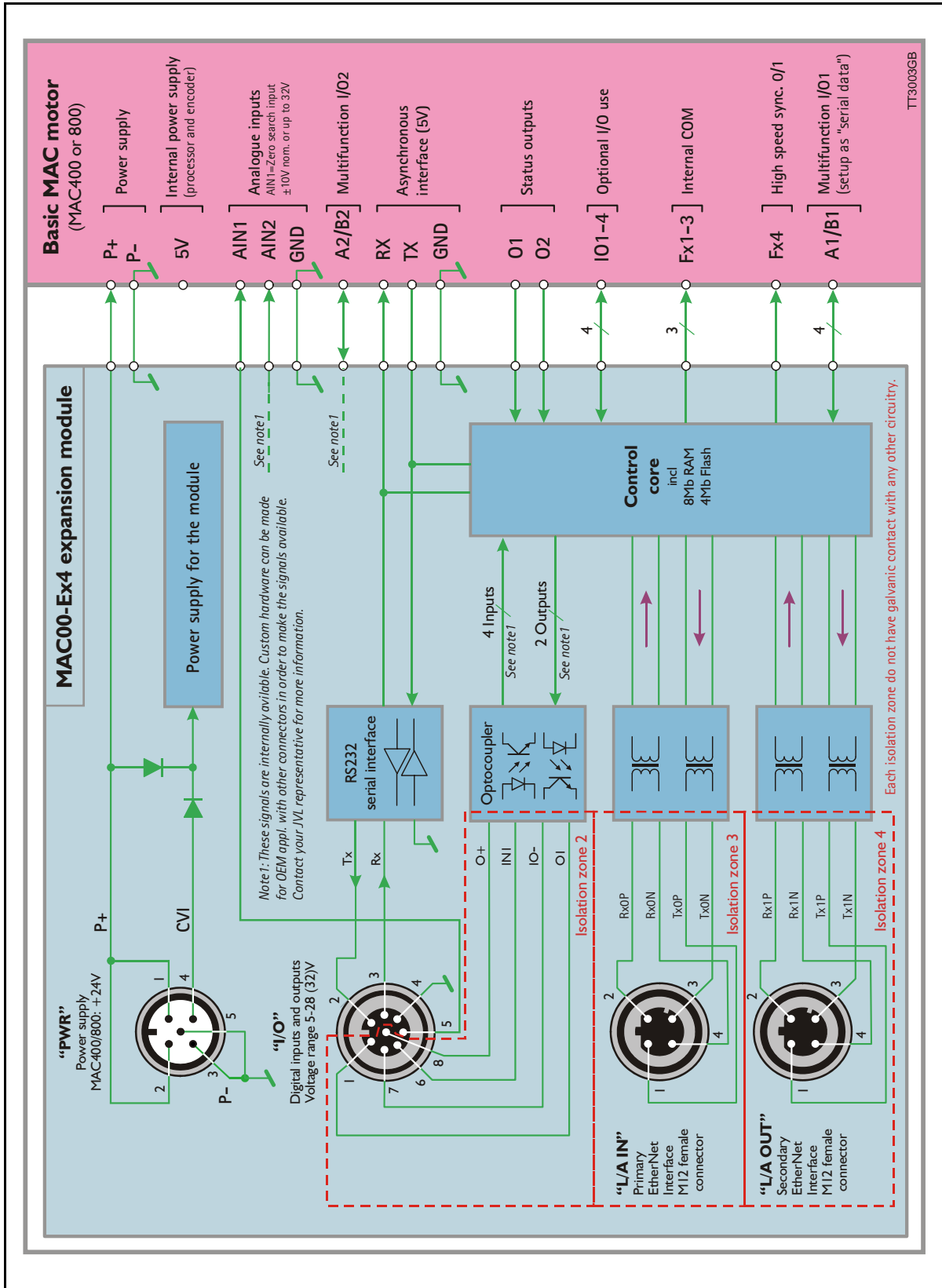


1.2

Hardware introduction

1.2.1 Overall hardware description

All internal and external main connections can be seen in the illustration below.



2 General Hardware description

2.1.1 External signals available at the MAC00-Ex4

Following signals are available at the MAC00-Ex4 module.

- **“L/A IN” and L/A OUT” connector.**
 - The Ethernet connection. L/A IN is connected to the upstream master and L/A OUT can be used downstream for the next motors/units in the chain.

- **“I/O” connector.**
 - AIN - analogue input +/-10V.
Can be used as input for the zero search sensor or as general analog input for speed or torque control depending on the what the actual operation mode in the motor has been setup for.
 - OI - user output I
Can be used as dedicated “in position” output (default) or as general output controllable over the Ethernet interface.
 - RS232 Interface.
Serial unbalanced interface for connection to a PC or a controller. The protocol is similar to the USB or RS485 interface, which means that all registers/parameters in the motor can be monitored or changed. RS232 is not recommended for long distances (> 10m).
 - INI - User input I.
Can be used as general input which can be read over the Ethernet interface.
 - I/O supply and gnd (IO- and O+).
Used as ground and supply for the user in/output (OI and INI).

- **“PWR” connector.**
 - 24V supply for the internal control circuitry in the motor

2.1.2 Hardware overview

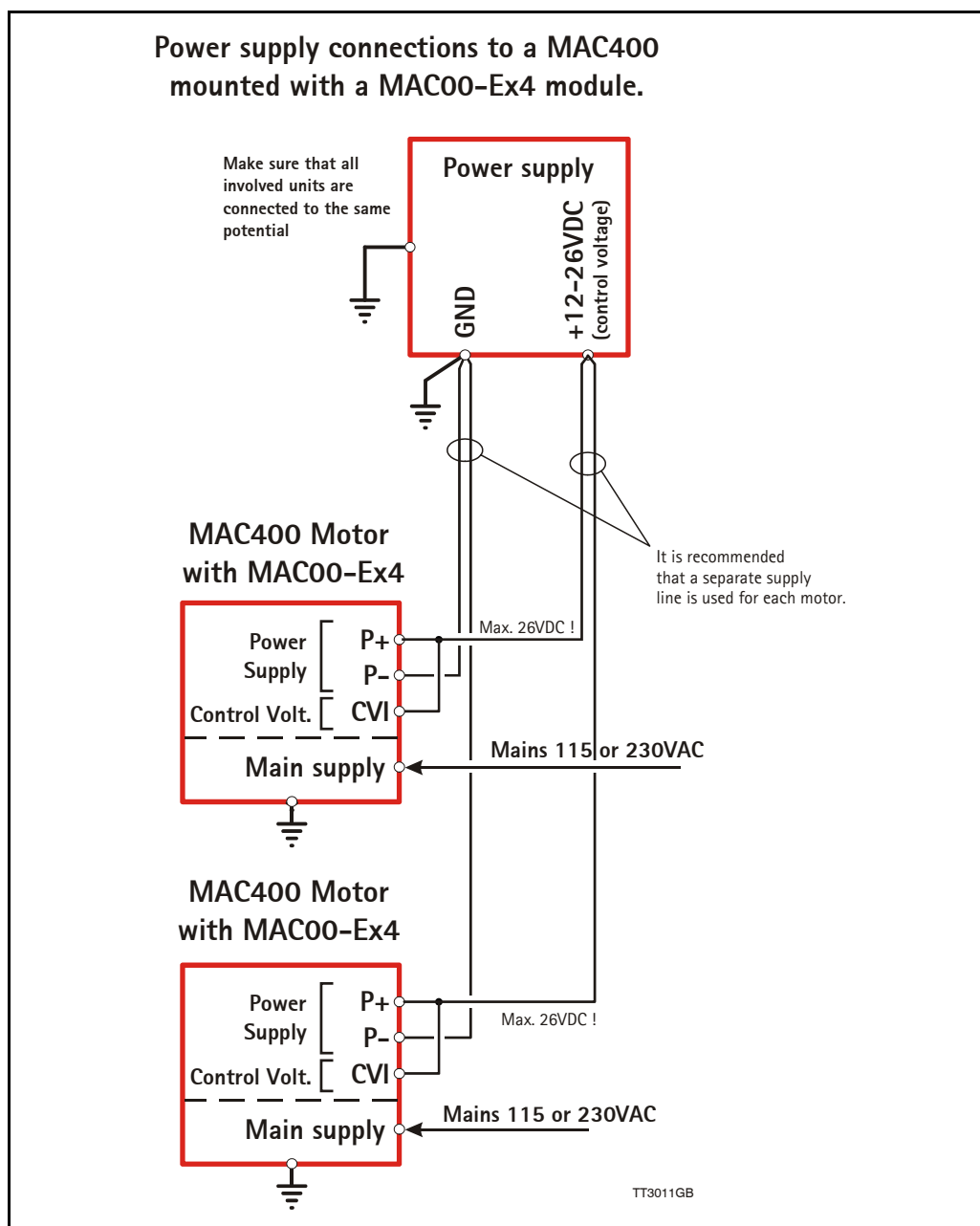
Picture arrives soon !

2.1.3 General power supply description

The Ethernet modules can only be used in the MAC400 and the MAC800 servomotor. The diagram below shows how to connect power to a MAC400 motor mounted with a MAC00-Ex4 module.

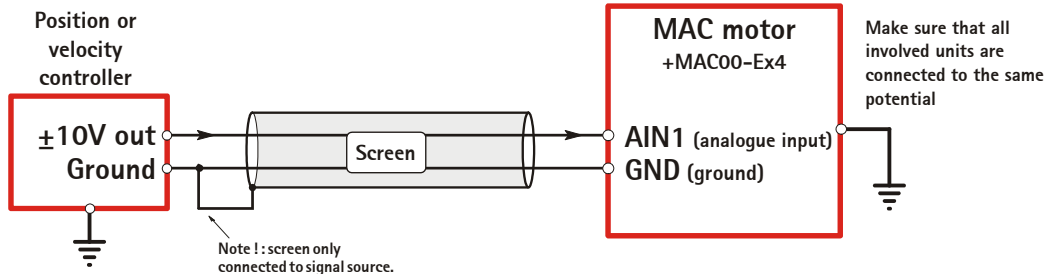
Please notice that the voltage connected to P+ and/or CVI must stay in the range +12-26VDC. Precautions must therefore be taken if the system also contains MAC50, 95, 140 or 141 which may require 48VDC in order to reach maximum motor speed.

See also the general power supply description in the MAC motor main manual LB0047. For further information concerning physical connections, see the *Expansion module MAC00-Ex4 connector description*, page 16.

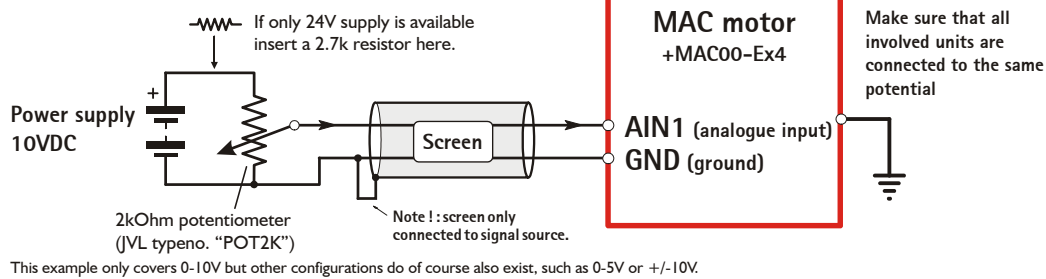


Analogue input connection at the MAC motor mounted with a MAC00-Ex4 module.

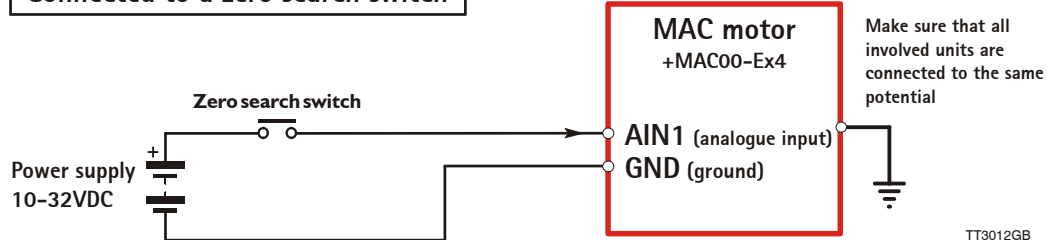
Connected to a external controller



Connected to a potentiometer



Connected to a zero search switch



TT3012GB

Note: Do not apply voltages higher than 32V to the analogue input (AIN)

2.1.4 Using the analogue input (AIN1).

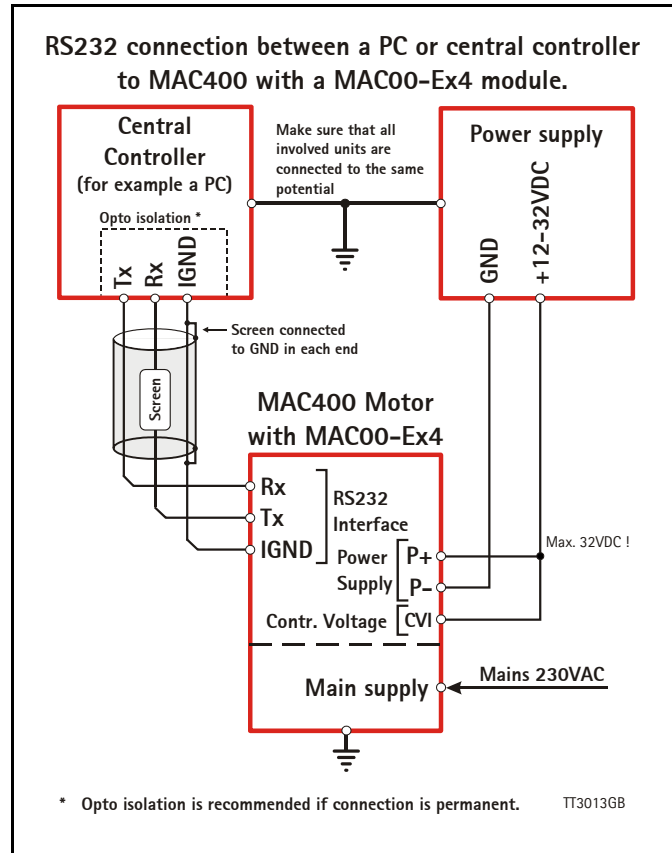
When a MAC00-Ex4 module is mounted in the MAC400 or MAC800 motor, the analogue inputs is available in the same manner as in the basic motor itself. The analogue inputs can be used for several applications and the function of the analogue input is determined by the mode in which the motor is set to operate. Typically the inputs is used for controlling the velocity, torque or position of the motor but the input is also used as digital input for zero search or in "Air Cylinder Mode" where it is used as trigger input for the movement done by the motor. For further information concerning physical connections, see the *Expansion module MAC00-Ex4 connector description*, page 16.

2.1.5 RS232 - General description when using the MAC00-Ex4 module

The RS232 interface is considered the main interface to the motor when the motor is set up using the MacTalk windows software from a PC or from any kind of controller using a RS232 interface.

When connecting the RS232 interface to a PC or controller, the following rules must be followed:

- 1 Only one motor can be connected at the interface line.
- 2 Use screened cable.
- 3 Ensure that GND (interface ground) is also connected.
- 4 Ensure that all units have a proper connection to safety ground (earth) in order to refer to the same potential.
- 5 The RS232 interface cable length should not exceed 10 metres.



Connectors:

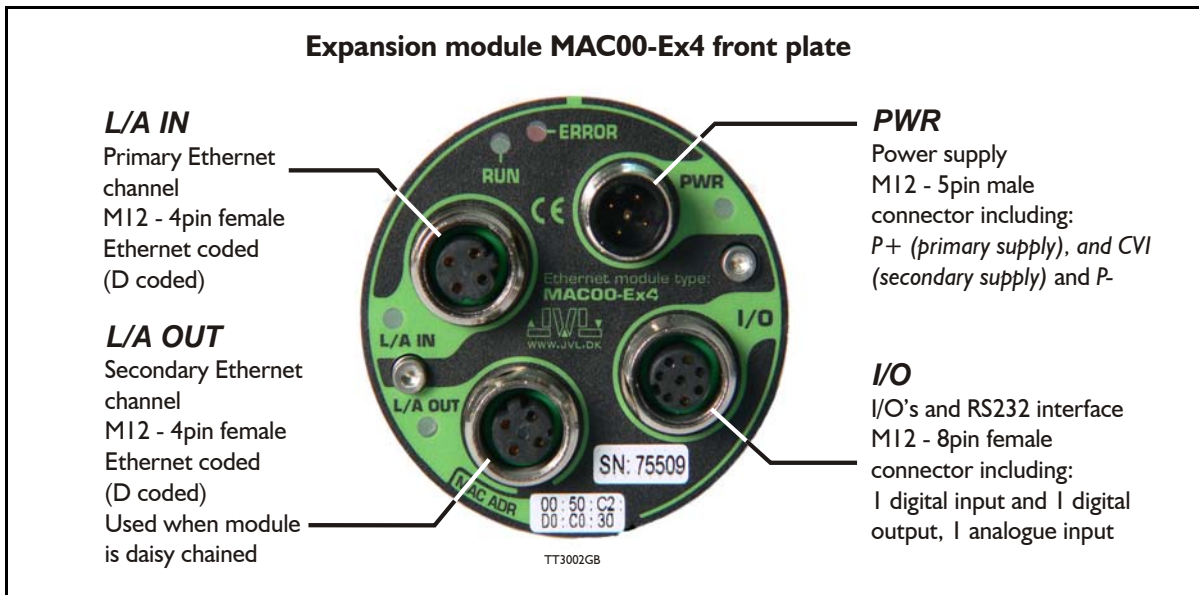
To see the specific connector pin-out please see the chapter *Expansion module MAC00-Ex4 connector description, page 16*.

A finished RS232 cable also exist. Please see *Cables for the MAC00-Ex4, page 18*

2.2

Connector description

Only MAC400&800



2.2.1 Expansion module MAC00-Ex4 connector description

The MAC00-Ex4 offers IP65 protection and M12 connectors which makes it ideal for automation applications where no additional protection is desired. The M12 connectors offer solid mechanical protection and are easy to unplug.

The connector layout:

"PWR" - Power input. M12 - 5pin male connector				
Signal name	Description	Pin no.	JVL Cable WI1000-M12F5T05N	Isolation group
P+	Main supply +12-24VDC. Connect with pin 2 *	1	Brown	1
P+	Main supply +12-24VDC. Connect with pin 1 *	2	White	1
P-	Main supply ground. Connect with pin 5 *	3	Blue	1
CVI	Control supply +12-24VDC. DO NOT connect >25V to this terminal !	4	Black	1
P-	Main supply ground. Connect with pin 3 *	5	Grey	1

* Note: P+ and P- are each available at 2 terminals. Make sure that both terminals are connected in order to split the supply current in 2 terminals and thereby avoid an overload of the connector.

(Continued next page)

2.2

Connector description

Only MAC400&800

“I/O” - I/O’s and interface. M12 - 8pin female connector.				
Signal name	Description	Pin no.	JVL Cable WI1000-M12 M8T05N	Isolation group (See note)
O1	Output 1 - PNP/Sourcing output	1	White	2
RS232: TX	RS232 interface. Transmit terminal Leave open if unused.	2	Brown	1
RS232: RX	RS232 interface. Receive terminal Leave open if unused.	3	Green	1
GND	Interface ground to be used together with the other signals in this connector. Also ground for the analogue input (AIN1 - pin 5)	4	Yellow	1
AIN1	Analogue input1 $\pm 10V$ or used for zero search	5	Grey	1
IN1	Digital input 1 - 12-32V tolerant.	6	Pink	2
IO-	I/O ground to be used with the I/O terminals O1 and IN1.	7	Blue	2
O+	Positive supply input to the output circuitry. Connect 5-32VDC to this terminal if using the O1 output.	8	Red	2
“L/A IN” - Ethernet port connector - M12 - 4pin female connector “D” coded				
Signal name	Description	Pin no.	JVL Cable WI1046- M12M4S05R	Isolation group (See note)
Tx0_P	Ethernet Transmit channel 0 - positive terminal	1	-	3
Rx0_P	Ethernet Receive channel 0 - positive terminal	2	-	3
Tx0_N	Ethernet Transmit channel 0 - negative terminal	3	-	3
Rx0_N	Ethernet Receive channel 0 - negative terminal	4	-	3
“L/A OUT” - Ethernet port connector. M12 - 4 pin female connector “D” coded				
Signal name	Description	Pin no.	JVL Cable WI1046- M12M4S05R	Isolation group (see note)
Tx1_P	Ethernet Transmit channel 1 - positive terminal	1	-	4
Rx1_P	Ethernet Receive channel 1 - positive terminal	2	-	4
Tx1_N	Ethernet Transmit channel 1 - negative terminal	3	-	4
Rx1_N	Ethernet Receive channel 1 - negative terminal	4	-	4
* Note: Isolation group indicate which terminals/circuits that a galvanic connected to each other. In other words group 1, 2, 3 and 4 are all fully independantly isolated from each other. Group 1 correspond to the housing of the motor which may also be connected to earth via the DC or AC input supply.				






2.3

Cable accessories

Only MAC400&800

2.3.1 Cables for the MAC00-Ex4

The following cables equipped with M12 connector can be supplied by JVL.

MAC00-Ex4 Connectors				Description	JVL Order no.	Picture
"L/A IN" 8pin male	"L/A OUT" 12pin Female	"I/O" 8pin Female	"PWR" 5pin Male			
		X		RS232 Interface cable. Connects directly from MAC00-Ex4 to a PC Length: 5m (197 inch)	RS232-M12-1-5-8	
		X		Cable with M12 male 8-pin connector loose wire ends 0.22mm ² (24AWG) and screen. Length: 5m (197 inch)	WI1000-M12M8T05N	
		X		Same as above but 20m (787 inch)	WI1000-M12M8T20N	
			X	Cable (Ø5.5mm) with M12 female 5-pin connector loose wire ends 0.35mm ² (22AWG) and foil screen. Length: 5m (197 inch)	WI1000-M12F5T05N	
			X	Same as above but 20m (787 inch)	WI1000-M12F5T20N	
X	X			Ethernet cable with M12 female 4pin D coded straight connector, and RJ45 connector (fits into std. Ethernetport)	WI1046-M12M4S05NRJ45	
X	X			Ethernet cable with M12 female 4pin D coded straight connector, loose ends.	WI1046-M12M4S05R	
X	X			Same as above but 15m (590 inch)	WI1046-M12M4S15R	
Protection caps. Optional if connector is not used to protect from dust / liquids.						
	X	X		IP67 protection cap for M12 female connector.	WI1000-M12FCAP1	
X			X	IP67 protection cap for M12 male connector.	WI1000-M12MCAP1	

Important: Please note that the cables are a standard type. They are not recommended for use in cable chains or where the cable is repeatedly bent. If this is required, use a special robot cable (2D or 3D cable).

3 MAC00-EC4 EtherCAT® module

3.1 Introduction to EtherCAT®

3.1.1 Intro to EtherCAT®.

EtherCAT® is a Real Time Ethernet technology which aims to maximize the use of the 100 Mbit, full duplex Ethernet bandwidth. It overcomes the overhead normally associated with Ethernet by employing "on the fly" processing hardware. An EtherCAT® net consists of a master system and up to 65535 slave devices, connected together with standard Ethernet cabling. The slave devices process the incoming Ethernet frames directly, extract or insert relevant data and transfer the frame to the next slave device, with a delay of approx. 4µs. The last slave device in the bus segment sends the processed frame back, so that it is returned by the first slave to the master as a kind of response frame. There are several protocols that can be used as the application layer. In the CANopen over EtherCAT® (CoE) technology, the CANopen protocol is applied to EtherCAT®. CANopen defines Service Data Objects (SDO), Process Data Objects (PDO) and the Object Dictionary structure to manage the parameters. Further information about EtherCAT®, is available from the EtherCAT® technology group (<http://www.ethercat.org>).

3.1.2 Abbreviations

Following general used terms are useful to know before reading the following chapters.

100Base-Tx	100 MBit Ethernet on twisted pairs
CAN	Controller Area Network
CANopen	Application layer protocol used in automation.
CoE	CANopen over EtherCAT®.
DC	Distributed Clock
EMCY	Emergency Object.
EoE	Ethernet over EtherCAT®.
ESI	EtherCAT® Slave Information
ESC	EtherCAT® Slave Controller
ETG	EtherCAT® Technology Group
EtherCAT®	Ethernet Control Automation Technologie
IPInternet	Protocol - IP address ~ the logical address of the device, which is user configurable (not used in EtherCAT®).
MAC	Media Access Controller - MAC address ~ the hardware address of the device (not used in EtherCAT®)
PDO	Process Data Object (for cyclic data)
SDO	Service Data Object (for acyclic data)
SII	Slave Information Interface
XML	eXtensible Markup Language - used for the ESI file.

3.2

Protocol specifications

3.2.1 EtherCAT® - communication

The EtherCAT® fieldbus system is standardised by the EtherCAT® user organisation (ETG). The driving force behind this is the German company, Beckhoff GmbH. Due to the advanced Ethernet technology used for EtherCAT®, in the future, customers can change from other fieldbus systems to EtherCAT® or generally equip new plant models with EtherCAT®.

Communication on EtherCAT® is based on a master/slave operation. The update cycle between master and slave depends on the number of EtherCAT® slaves, the amount of process data of the individual slaves, and the set update time of the master. Due to the ring topology, in every bus cycle only one telegram is sent on the bus. The bus cycle time thus remains exactly the same in every cycle.

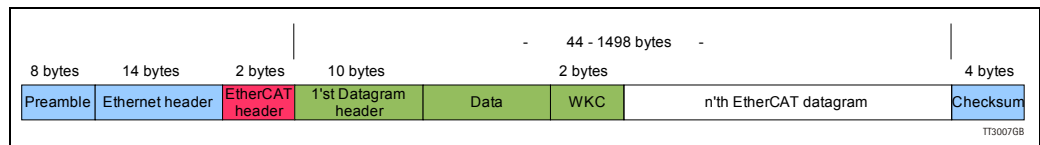
Slave addressing can be done in two ways:

- Auto increment addressing
- Fixed node addressing

With Auto increment addressing the master scans the net for slaves, and the slaves are then addressed in the sequence they are physically present on the net. With fixed node addressing, the addresses that each node has programmed, is used.

3.2.2 EtherCAT® frame structure

In EtherCAT®, the data between the master and the slaves is transmitted in Ethernet frames. An EtherCAT® Ethernet frame consists of one or several EtherCAT® telegrams, each addressing individual devices and/or memory areas. The telegrams can be transported either directly in the data area of the Ethernet frame or within the data section of a UDP datagram transported via IP. The EtherCAT® frame structure is pictured in the following figure. Each EtherCAT® telegram consists of an EtherCAT® header, the data area and a working counter (WKC), which is incremented by all EtherCAT® nodes that are addressed by the telegram and have exchanged associated data.



3.2.3 Sync managers

Sync managers control the access to the application memory. Each channel defines a consistent area of the application memory. The adapter module has four sync manager channels. The mailbox protocol (SDO's) and process data (PDO's) are described later in this chapter.

3.2.4 Sync manager watchdog

The sync manager watchdog monitors the output sync managers. If the output data is not updated by the EtherCAT® master within the configured time, the watchdog will activate time out and change the state of the adapter module from Operational to Safe-Operational.

Note: EtherCAT® has been designed so that it provides no way for a slave to monitor the connection to the master if the slave gets no output data.

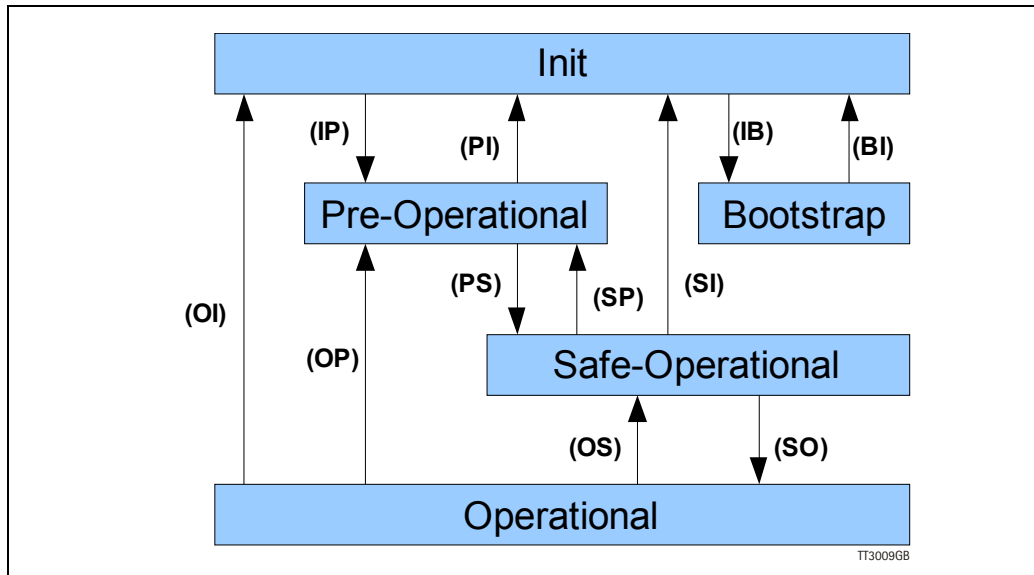
Note: The drive reaction to a communication fault must be configured in the module write flag register (object 2011 subindex 6 - motor set passive or motor set velocity =0).

3.2

Protocol specifications

3.2.5 EtherCAT® - State machine

Both the master and the slaves have a state machine with the states shown below. After boot the slaves are in INIT state, and then it's up to the master to request state transitions. The standardized EtherCAT® state machine is defined in the following figure. The bootstrap state is not supported.



The module enters the Init state directly after start-up. After this, the module can be switched to the Pre-Operational state. In the Pre-operational state the EtherCAT® mailbox communication is allowed and CoE objects can be accessed by SDOs. After the master has configured the slave, it can switch the module to the Safe-Operational state. In this state input I/O data (PDOs) is sent from the adapter module to the EtherCAT® master, but there is no output I/O data from the master to the module. To communicate output I/O data the master must switch the adapter module to the Operational state.

State description table:

State	Description
Init	State after device initialisation. No Application layer communication (no SDO and PDO communication).
Pre-operational	SDO communication possible. No PDO communication.
Safe-operational	Transmit PDO operational (drive sends data to master)
Operational	Drive fully operational, responds to data via receive PDO
Boot-strap	Not used.

3.2

Protocol specifications

3.2.6 **CANopen over EtherCAT®**

The application layer communication protocol in EtherCAT® is based on the CANopen DS 301 communication profile and is called CANopen over EtherCAT® (CoE). The protocol specifies the Object Dictionary in the adapter module, in addition to communication objects for exchanging cyclic process data and acyclic messages.

The EtherCAT® module uses the following message types:

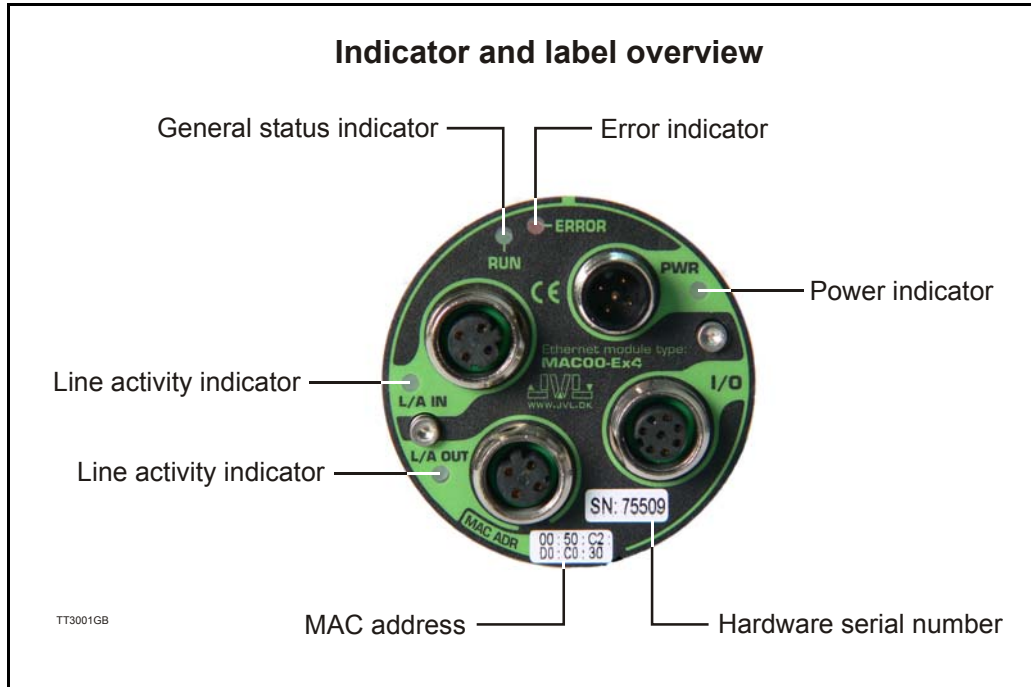
- Process Data Object (PDO). The PDO is used for cyclic I/O communication, in other words, process data.
- Service Data Object (SDO). The SDO is used for much slower acyclic data transmission.
- Emergency Object (EMCY). The EMCY is used for error reporting when a fault has occurred in the module or in the drive.

3.3

Commisioning

3.3.1 Indicator LED's - description.

The LED's are used for indicating states and faults of module. There is one power LED, two link/activity LED's (one for each Ethernet connector), and 2 status LED's.



LED indicator descriptions

LED Text	Colour	Constant off	Constant on	Blinking	Single flash	Double flash	Flickering
L/A IN	Green	No valid Ethernet connection.	Ethernet is connected.	-	-	-	Activity on line
L/A OUT	Green	No valid Ethernet connection.	Ethernet is connected.	-	-	-	Activity on line
RUN	Green	Device state = INIT	Device state = Operational	Device state = Pre-operational	Device state = Safe-operational	-	-
ERROR	Red	No error	Critical communication or controller error	General configuration error	Local error	Process data watchdog timeout / EtherCAT® watchdog timeout	Booting error
PWR	Green	Power is not applied.	Power is applied to both motor and module.	-	-	-	Power is applied to module but no communication with motor.

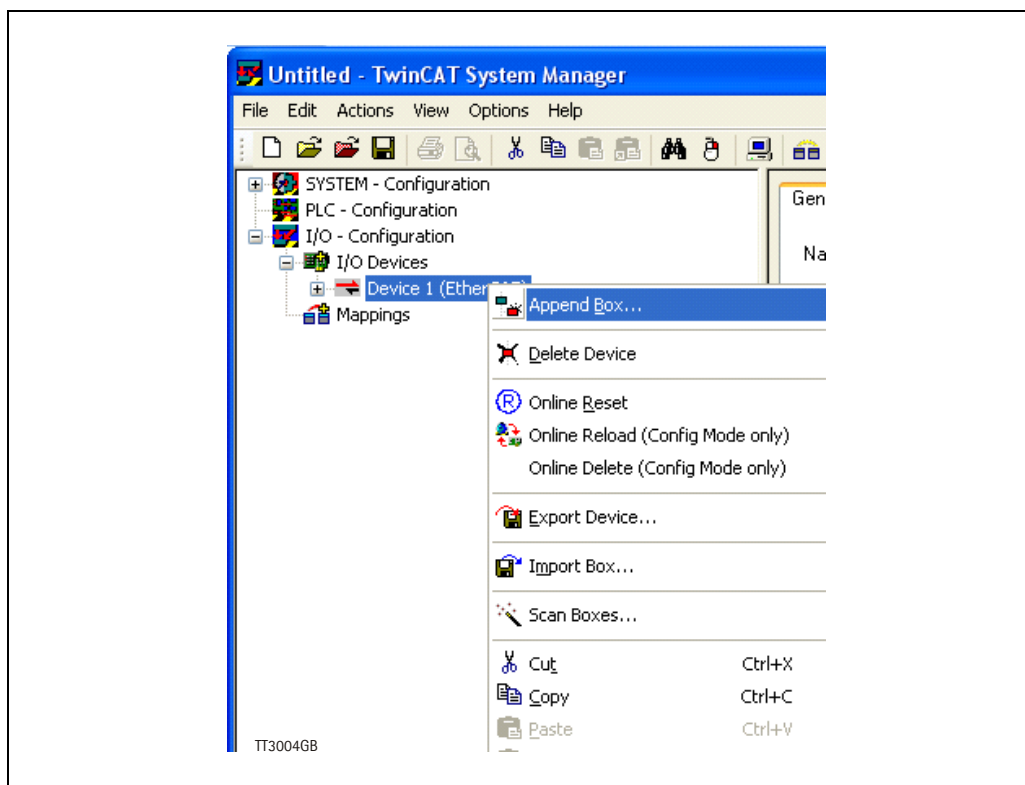
Notes:
Blinking: Flashing with equal on and off periods of 200ms (2.5Hz). **Single flash:** Repeating on for 200ms and off for 1s. **Double flash:** Two flashes with a period of 200ms followed by 1s off period. **Flickering:** Rapid flashing with a period of approx. 50ms (10 Hz).

3.3

Commissioning

3.3.2 Quick start with TwinCAT.

1. Copy the Ethernet slave information file ("JVL ECS V10.XML") to the folder "..\Twincat\IO\Ethernet\" on the master PC.
2. Apply power, and make sure the *PWR* (power) LED is lit.
3. Connect the Ethernet cable from Master to the L/A IN connector, and check that the corresponding LED is lit.
4. Start TwinCAT - system manager on the master, and make sure that a proper Ethernet I/O device is appended (consult your TwinCAT manual).
5. Right click the I/O device, and select "append box".

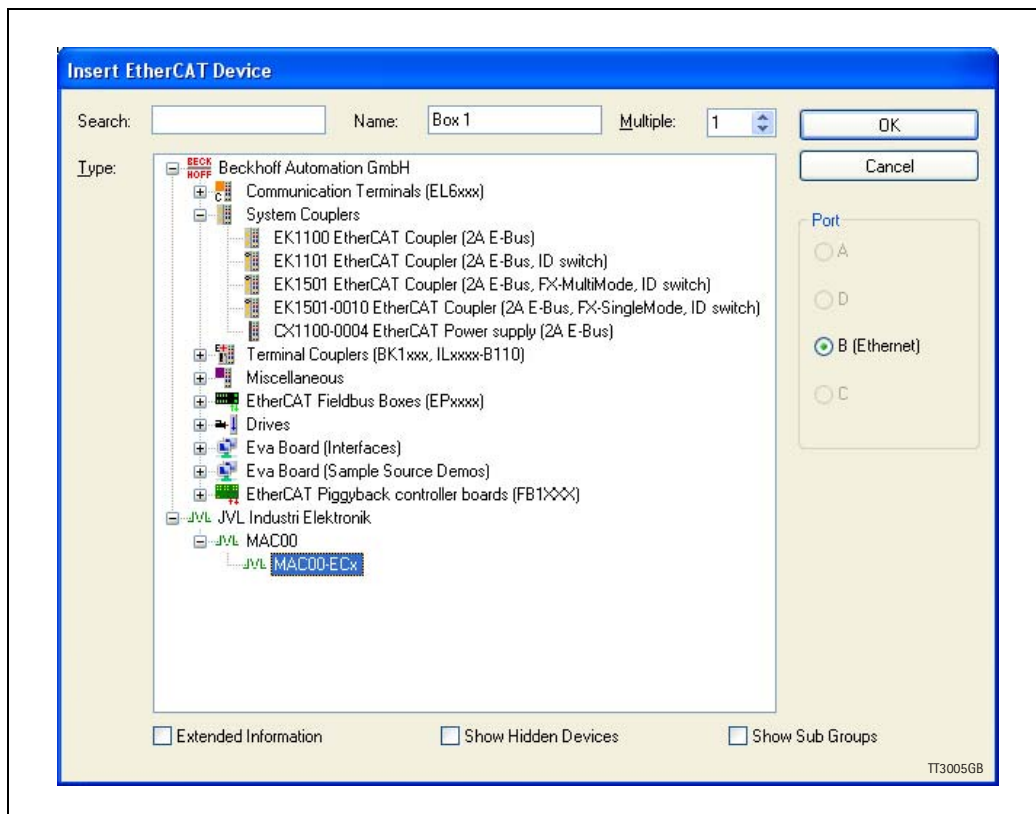


Continued next page

3.3

Commissioning

6. Unfold "JVL Industri Elektronik" and "MAC00".
7. Select "MAC00-ECx" and press the OK button.



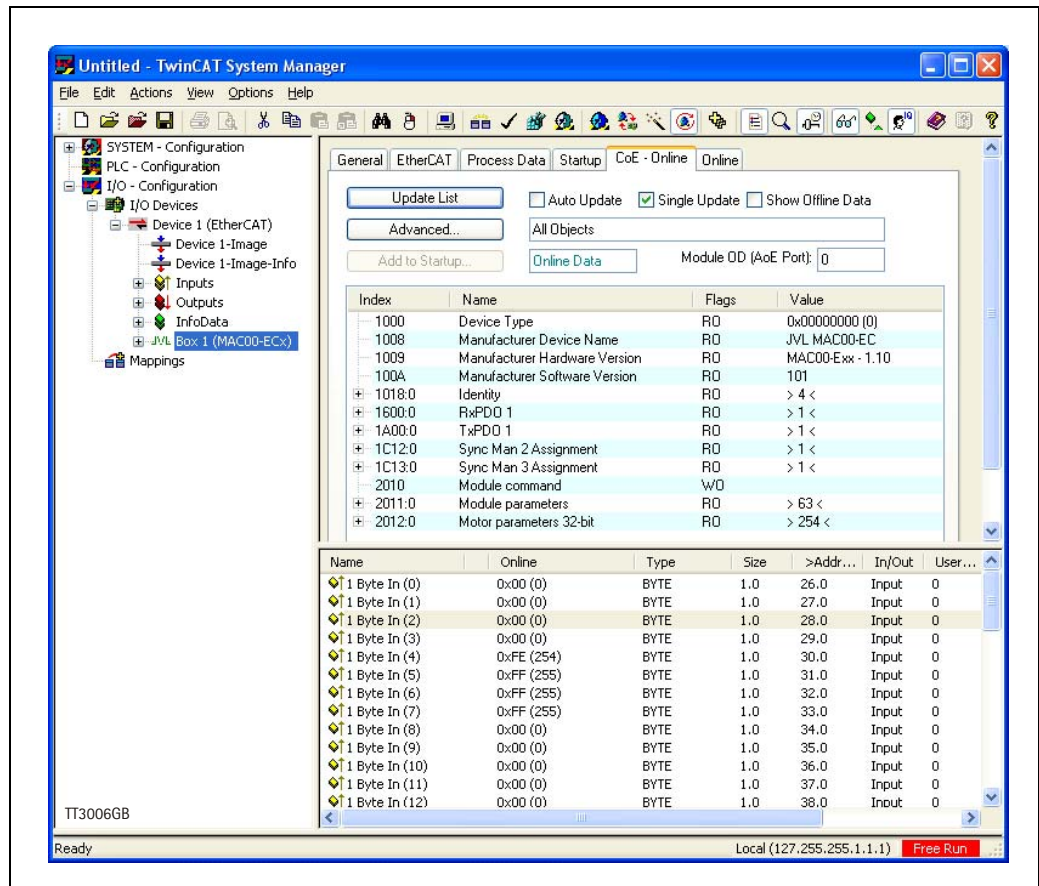
8. The device should now appear in the left side of the TwinCAT window, with a tiny JVL logo.
9. Press F4 (Reload I/O devices), and select the JVL device on the left side of the window.
10. The "L/A IN" LED should now be flashing and the process data should now appear on the bottom right side of the TwinCAT window.

Continued next page

3.3

Commissioning

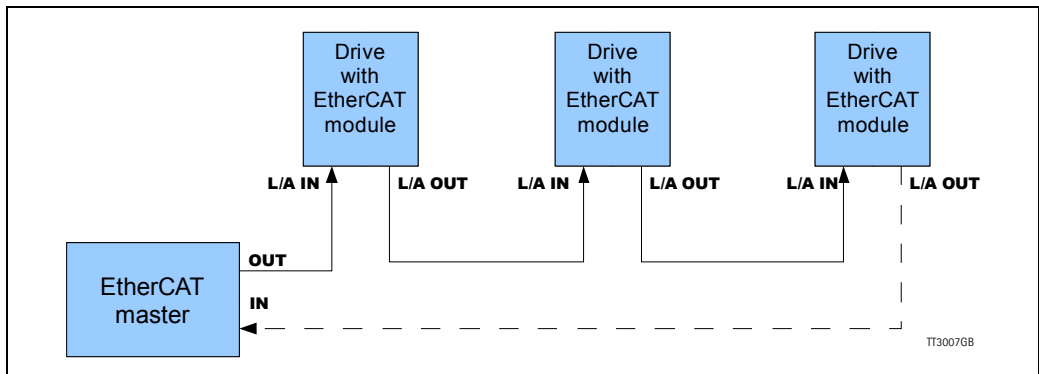
11. By pressing the "CoE online" tab, it's possible to inspect the CANopen objects, and modify motor and module parameters.



3.3.3

Mechanical installation

The network cables must be connected to the two M12 connectors (marked "L/A IN" and "L/A OUT") on the module. The cable from the EtherCAT® master is always connected to the "L/A IN" port. In the line topology, if there are more slave devices in the same line, the next slave device is connected to the port marked "L/A OUT". If there is a redundant ring, the right "L/A OUT" port of the last slave device is connected to the second port of the EtherCAT® master. See the figure below. Standard CAT 5 FTP or STP cables can be used. It is not recommended to use UTP cables in industrial environments, which is typically very noisy.



3.4

EtherCAT® objects

3.4.1 Process Data Object 21 (PDO)

PDO's (Process Data Objects) are used for cyclic transfer of time-critical process data between master and slaves. There is one receive PDO and one transmit PDO which is fully user configurable. Tx PDOs are used to transfer data from the slave to the master and Rx PDOs to transfer data from the master to the slave. It is possible to set up five, 32 bit registers in each PDO. The setup is done with MacTalk or via SDO object 0x2011 subindex 16-31. It requires a save in flash and a power cycle before the new configuration are used. If the configuration of the PDO's, is not altered by the user, the MAC00-EC4 module uses the default mapping shown in the tables below.

Default registers in transmit PDO 21 (Slave > Master)

Object index	Register no.	Motor register short	Motor register description
0	2	MODE_REG	Operating mode
1	10	P_IST	Actual position
2	12	V_IST	Actual velocity
3	169	VF_OUT	Actual torque
4	35	ERR_STAT	Status bits

Default registers in receive PDO 21 (Master > Slave)

Object index	Register no.	Motor register short	Motor register description
0	2	MODE_REG	Operating mode
1	3	P_SOLL	Target position
2	5	V_SOLL	Maximum velocity
3	7	T_SOLL	Maximum torque
4	-	-	-

3.4.2 Service Data Objects (SDO)

Service Data Objects (SDOs) are mainly used for transferring non time-critical data, for example, identification, configuration and acyclic data.

3.4.3 Emergency Objects

Emergency Objects (EMCYs) are used for sending fault information from the communication module and the drive to the EtherCAT® network. They are transmitted whenever a fault occurs in the drive or in the module. Only one Emergency Object is transmitted per fault. EMCYs are transmitted via SDO's.

3.4

EtherCAT® objects

3.4.4 Object Dictionary

An important part of the CoE protocol is the Object Dictionary, which is different objects specifying the data layout. Each object is addressed using a 16-bit index and possibly a sub index. There are some mandatory objects and some manufacturer specific objects. The objects in the CoE Object Dictionary can be accessed with SDO services.

3.4.5 Mandatory objects:

Name	Index (hex)	Sub Index	Data Type	Read only	Default	Description
Device type	1000		UNSIGNED32	X	0x0	Contains information about the device type.
Error Register	1001		UNSIGNED8	X		This is the mapping error register, and it is part of the emergency object. If some of the sub index are high, an error has occurred. See also section 4.3.21. Mandatory
		0				Generic error. Mandatory
		1				Current
		2				Voltage
		3				Temperature
		4				Communication (Overrun)
		5				Device profile specific
		6				Reserved
	7	Manufacturer specific				
Manufacturer device name	1008		VISIBLE STRING	X	JVL - MAC00-ECx	
Manufacturer hardware version	1009		VISIBLE STRING	X	1.0	
Manufacturer software version	100A		VISIBLE STRING	X	1.0	Example: Version x.x
Identity object	1018		IDENTITY	X		Contain general information about the module
		0	1..4	X	4h	Number of entries. Mandatory
		1	UNSIGNED32	X	0x0117	Vendor ID, contains a unique value allocated to each manufacturer. 117h is JVLs vendor ID. Mandatory.
		2	UNSIGNED32	X	0x0200	Product Code, identifies a specific device version. The MAC00-EC4 has the product code 200h
		3	UNSIGNED32	X	0x20020	Revision number.
	4	UNSIGNED32	X		Serial number	

3.4

EtherCAT® objects

3.4.6 Manufacturer specific objects.

The manufacturer specific objects, provides access to all module registers, and all motor registers, as well as a module command object.

	Index (hex)	Sub Index	Type	Read only	Default	Description	
Module command	2010	0	UNSIGNED32			Module command object. See possible commands below.	
Module parameters	2011	0	UNSIGNED8	X	63	Subindex count	
		1	UNSIGNED32	X		High 16 bit of MAC address (placed in low 16 bit of word)	
		2	UNSIGNED32	X		Low 32 bit of MAC address	
		3	UNSIGNED32			IP address	
		4	UNSIGNED32			Net mask	
		5	UNSIGNED32			Gateway	
		6	UNSIGNED32			0x0	Setup bits
		7	UNSIGNED32			0	Digital outputs on module
		8-15	UNSIGNED32			-	Reserved for future use
		16	UNSIGNED32			2	Register no. to place in TxPDO 21, position 1.
		17	UNSIGNED32			10	Register no. to place in TxPDO 21, position 2.
		18	UNSIGNED32			12	Register no. to place in TxPDO 21, position 3.
		19	UNSIGNED32			169	Register no. to place in TxPDO 21, position 4.
		20	UNSIGNED32			35	Register no. to place in TxPDO 21, position 5.
		21	UNSIGNED32			-	Reserved for future use
		22	UNSIGNED32			-	Reserved for future use
		23	UNSIGNED32			-	Reserved for future use
		24	UNSIGNED32			2	Register no. to place in RxPDO 21, position 1.
		25	UNSIGNED32			3	Register no. to place in RxPDO 21, position 2.
		26	UNSIGNED32			5	Register no. to place in RxPDO 21, position 3.
		27	UNSIGNED32			7	Register no. to place in RxPDO 21, position 4.
		28	UNSIGNED32			0	Register no. to place in RxPDO 21, position 5.
		29	UNSIGNED32			-	Reserved for future use
		30	UNSIGNED32			-	Reserved for future use
		31	UNSIGNED32			-	Reserved for future use
		32	UNSIGNED32		X	-	Module serial no.
		33	UNSIGNED32		X	-	Module hardware version
34	UNSIGNED32		X	-	Module software version		
35	UNSIGNED32		X	-	No. of internal motor communication timeouts		
36	UNSIGNED32		X	-	No. of retry frames to motor		
37	UNSIGNED32		X	-	No. of discarded frames to motor		
38	UNSIGNED32		X	-	Total no. of frames to motor		
39-46	UNSIGNED32		X	-	Reserved for future use		
47	UNSIGNED32		X	-	Digital inputs on module		
48	UNSIGNED32		X	-	Status bits		
49-63					Reserved for future use		
Motor parameters	2012	0	UNSIGNED8	X	254	Subindex count	
		N	UNSIGNED32			Access to the motor parameter n	

Note: Module parameters are not automatically saved to permanent memory after a change. The parameters can be saved permanently by applying a "Save parameters to flash" command afterwards.

3.4

EtherCAT® objects

3.4.7 Object 0x2010 - Subindex 0

This object is used for sending commands to the module and is write only. The possible commands are listed in the table below.

Command no.	Function
0x0	No operation
0x10	Save parameters to flash

3.4.8 Object 0x2011

The module registers is mapped to object 0x2011. The subindex 3-31 is R/W, the rest is read only.

3.4.9 Object 0x2011 - Subindex 1-5

Reserved for future use.

3.4.10 Object 0x2011 - Subindex 6 Setup bits

This register is used to setup how the module should react on different events.

Bit	1-31	0
Output	Reserved	0 : Ethernet error handling = motor set passive mode 1 : Ethernet error handling = motor set velocity to 0

3.4.11 Object 0x2011 - Subindex 7 Digital inputs on module

With this object the status of the 4 digital inputs can be read.

Bit	4-31	3	2	1	0
Input	Reserved	IN4*	IN3*	IN2*	IN1*

* The availability of the inputs depends on the actual version of the module used. Example MAC00-EC4 only support Input 1 (IN1).

3.4.12 Object 0x2011 - Subindex 16-23 Register no. to place in TxPDO 21

These registers contain the numbers that define the registers which are in the TxPDO 21. That is the register's, which is transmitted from slave to master cyclically. If some of these registers are changed, it is necessary to issue a "save in flash" command and to reboot the device before the changes take effect.

3.4.13 Object 0x2011 - Subindex 24-31 Register no. to place in RxPDO 21

These registers contain the numbers that define the registers which are in the RxPDO 21. That is the register's, which is transmitted from master to slave cyclically. If some of these registers are changed, it is necessary to issue a "save in flash" command and to reboot the device before the changes take effect.

3.4 EtherCAT® objects

3.4.14 Object 0x2011 - Subindex 32-38

These registers contain HW, SW and communication information of the module.

3.4.15 Object 0x2011 - Subindex 47 Digital outputs on module

With this object the digital outputs can be controlled.

The value written to this object is directly shown on the digital outputs.

Bit	2-31	1	0
Output	Reserved	Output2* (O2)	Output1* (O1)

* The availability of the outputs depends on the actual version of the module used.
Example MAC00-EC4 only support Output I (O1).

3.4.16 Object 0x2011 - Subindex 48 Status bits

This register is used for miscellaneous information about the module.

Bit	8-31	7	0-6
Output	Reserved	1=No communication with the motor	Reserved

3.4.17 Object 0x2012

Object 0x2012 are for acyclic view or change of motor registers, se register descriptions in specific motor manual.

3.4.18 EtherCAT® Slave Information file

EtherCAT® Slave Information file (ESI) is a XML file that specify the properties of the slave device for the EtherCAT® master and contains information on the supported communication objects. EtherCAT® Slave Information files for JVL drives are available through your local JVL representative. If TwinCAT is used for master then the XML-file shall be copied to the folder "..\TwinCAT\Io\EtherCAT\".

4 **MAC00-EI4 EthernetIP module**

4	MAC00-EI4 EthernetIP module	33
4.1	Introduction to EtherNetIP	34
4.2	Using non cyclic messages	36
4.3	Using cyclic messages I/O-messages	38
4.4	Commissioning	39
4.5	Examples	41
4.6	Configuration using different methods	43
4.7	Using and Selecting an Ethernet switch	46
4.8	Objects accessible using Explicit messages	47
4.9	Examples of applications	50

4.1 Introduction to EthernetIP

4.1.1 Intro to EtherNet/IP

The JVL MAC00-EI -module makes communication using EtherNet/IP possible with the JVL motor.

The Ethernet technology gives the advantages of fast data access using standard off the shelf hardware which again has the advantage of large accessibility and low prices.

The JVL implementation is done in a way that minimizes the complexity of getting a system up and running but still utilizes the benefits of industrial ethernet.

The JVL EtherNet/IP implementation supports both explicit messaging and I/O messages with up to 5 free configurable input and output words.

With a basic knowledge of the JVL motor operation through the register structure and a basic knowledge of the EtherNet/IP technology, a motor can be setup and controlled in a very short time without first doing extensive studies in complex motion control standards etc.

EtherNet/IP is basically divided in 2 groups of data, explicit and I/O messages in other words messages requiring fast data response time and data not so time critical typically used for configuration purposes. In the EtherNet/IP terminology these messages are also called Explicit messages (not time critical) and I/O messages (time critical).

In the motion control world, time critical data would be actual position, actual status and actual speed and actual torque where data not time critical would be such as motor temperature and setup parameters.

EtherNet/IP is object based similar to DeviceNet and follows the standards issued by ODVA.

For more information on EtherNet/IP please visit www.ODVA.org for further details on EtherNet/IP and to get the EtherNet/IP standard specification issued by ODVA.

The JVL implementation supports manufacture specific objects to gain access to each register in the motor.

4.1 Introduction to EthernetIP

4.1.2 EthernetIP specification

The JVL implementation supports manufacturer specific objects to gain access to each register in the motor.

Supported standard EthernetIP classes

Type	Class
Identity Object, class	0x01
Message router object, class	0x02
Assembly object, class	0x04
TCP/IP interface object, class	0xF5
Ethernet link object, class	0xF6

On top of this the JVL manufacture specifi class object 0x64 has been added.

4.1.3 Identity object class 0x01

Holds information about the JVL device on the network. Typical used by other devices to identify devices on the network.
(for further specification please refer to the EtherNet/IP appx.)

4.1.4 Message router object class 0x02

Handles all messages to/from object's in the device.

4.1.5 Assembly object class 0x04

Object that binds all IO data to a connection point.

4.1.6 TCP/IP interface object class 0xF5

Holds all information on the Ethernet connection, such as the IP-adres, Network mask and GateWay.

4.1.7 Ethernet link object class 0xF6

Holds information on link specific counters and instances associated with the communication interface.

To gain access to the motor registers Class object 0x64 is used.

See section "Objects accessible using Explicit messages" for further details.

4.2 Using non cyclic messages

4.2.1 Using non cyclic messages (Explicit messages)

Non cyclic messages in the EtherNet/IP domain is called Explicit messages. This message type is typically used to perform configuration and other non-time critical operations.

Explicit messages can be send as a connected or unconnected message.

All registers in the motor can be accessed explicitly using object class 0x64. The register range in the motor is from 1-255 all 32bit size.

For a complete register list please See “Motor registers” on page 53.

The object class 0x64 explained in details:

Service type and code supported:

Set_Attribute_Single0x10

Get_Attribute_Single0xE

Instances supported: 0x01-0xFF (motor registers 1-255)

4.2 Using non cyclic messages

4.2.2 Example 1

We would like to set the motor into velocity mode.
This requires that the mode register 2 = 0x1.
Velocity mode is 0x1, Position mode = 0x2 etc.
All modes of operation is further described in the servo manual.

Package:
Class: 0x64
Service: 0x10 (write data)
Instance: 0x2 (mode register in the motor)
Attribute: 0x1

Data: **0x01 0x00 0x00 0x00**

This will set the mode register in the motor into velocity -mode
Motor Register 2 = 1

If we choosed the Littleendian format we would form the data structure in this way:
Data: **0x00 0x00 0x00 0x01**

Now if we want to read a value from the motor we use the service code 0xE.

4.2.3 Example 2

After setting the motor into velocity mode it will start running. Now the actual velocity can be read while the motor is running.

Package:
Class: 0x64
Service: 0xE (write data)
Instance: 0x5 (mode register in the motor)
Attribute: 0x1

Now the response data is received:

Data: **0x01 0x15 0x00 0x00**

This value 0x15 is the decimal value 277 which corresponds to 100 RPM. This is the default velocity value.

So basically the motor can be controlled and all needed data can be retrieved using explicit messages. This method is not suitable when data is needed fast and frequently for this purpose I/O messaging (Implicit messaging) is used.

Not only motor registers are accessible using explicit messages, also static data such as serial numbers, network status etc are accessible. These informations are accessible according to the EtherNet/IP standard and follows the implemented classes: **0x1, 0x4, 0xF5, 0xF6**. These classes are explained in details in the EtherNet/IP standard (obtained from www.ODVA.org) and in

For further info please See “Objects accessible using Explicit messages” on page 47.

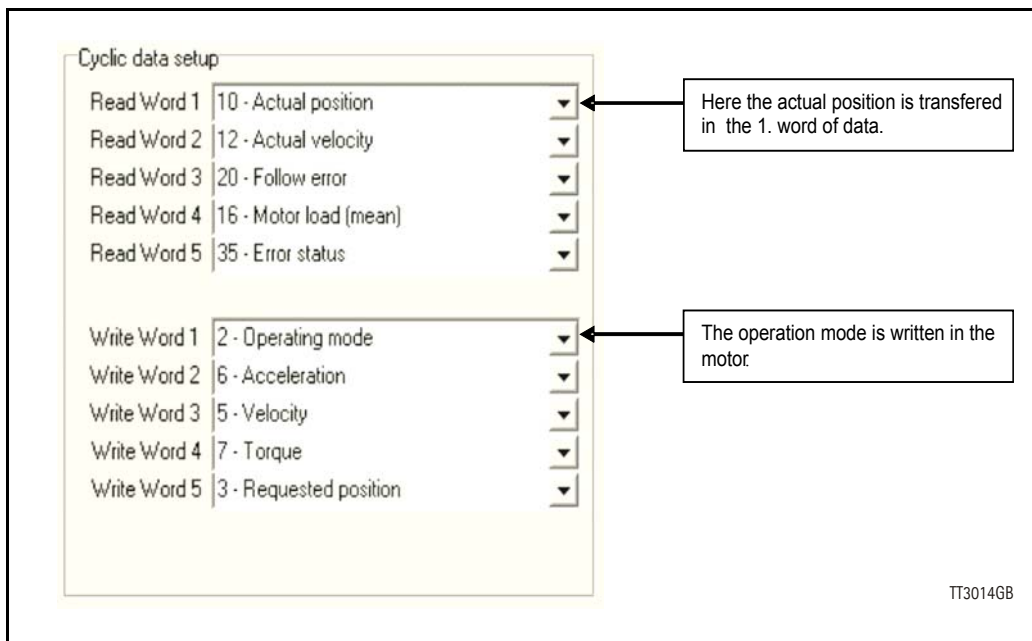
4.3 Using cyclic messages I/O-messages

4.3.1 Cyclic messages.

I/O messaging also referred to as Implicit messages is used when data is needed fast and frequent. That is fast dynamic changing data such as position, velocity, torque etc.

These data is send cyclic using the assembly class object 0x04.

The JVL assembly consists of 5 I/O words that is freely configurable. This means that 5 input motor registers can be selected and another 5 motor registers for output purposes. The terms Input and output is considered from the scanner so input is data flowing from the motor to the scanner and output is vice versa.



All words are 4 bytes.

In the example shown above the 5 read words (data read from the motor) are:

Motor register 10 (Actual position)	The actual motor position
Motor register 12 (Actual velocity)	The actual velocity of the motor
Motor register 20 (Follow error)	The actual follow error the motor is experiencing
Motor register 16 (Motor load - mean)	The load the motor is experiencing over time
Motor register 35 (Error status)	Bit-field that holds both error information and status of movements etc.

The 5 write registers are configured to hold the following data:

Motor register 2 (Operating mode)	0=passive, 1=Velocity, 2=position etc
Motor register 6 (Acceleration)	The requested acceleration to be used.
Motor register 5 (Velocity)	The requested Velocity to be used
Motor register 7 (Torque)	The max. allowed Torque to be used
Motor register 3 (Requested position)	The requested position if operating mode = 2 (position)

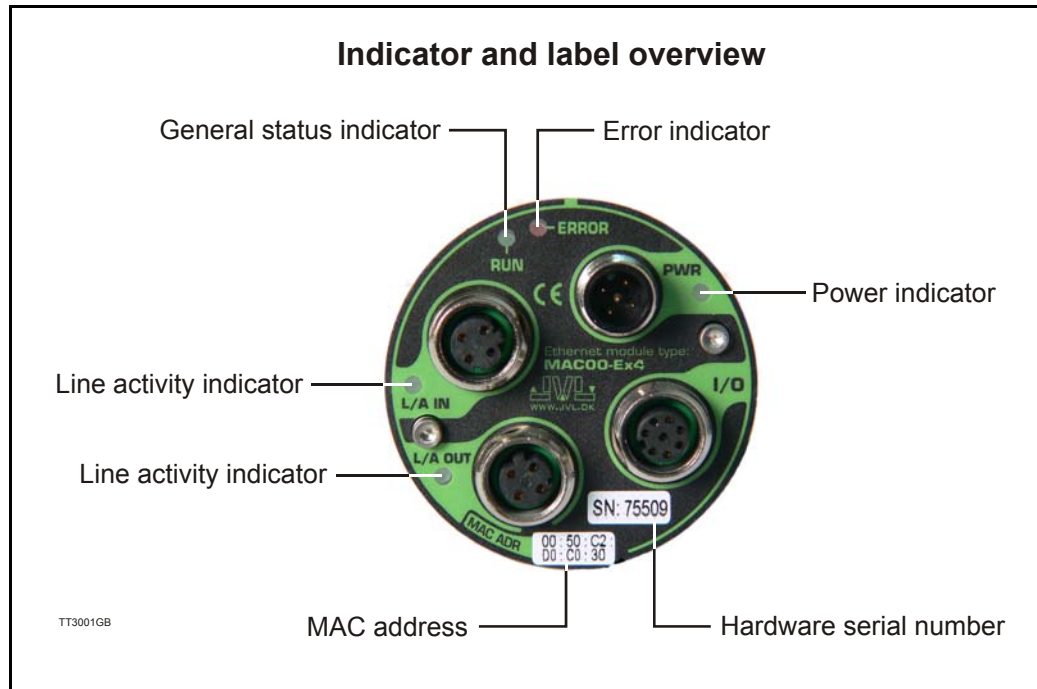
For a complete register description please See "Motor registers" on page 53.

4.4

Commissioning

4.4.1 Indicator LED's - description.

The LED's are used for indicating states and faults of module. There is one power LED, two link/activity LED's (one for each Ethernet connector), and 2 status LED's.



LED indicator descriptions

LED Text	Colour	Constant off	Constant on	Blinking	Flickering
L/A IN	Green	No valid Ethernet connection.	Ethernet is connected.	-	Activity on line
L/A OUT	Green	No valid Ethernet connection.	Ethernet is connected.	-	Activity on line
RUN	Green	TBD	TBD	TBD	TBD
ERROR	Red	TBD	TBD	TBD	TBD
PWR	Green	Power is not applied.	Power is applied.	-	Power is applied to module but no communication with motor

Notes:

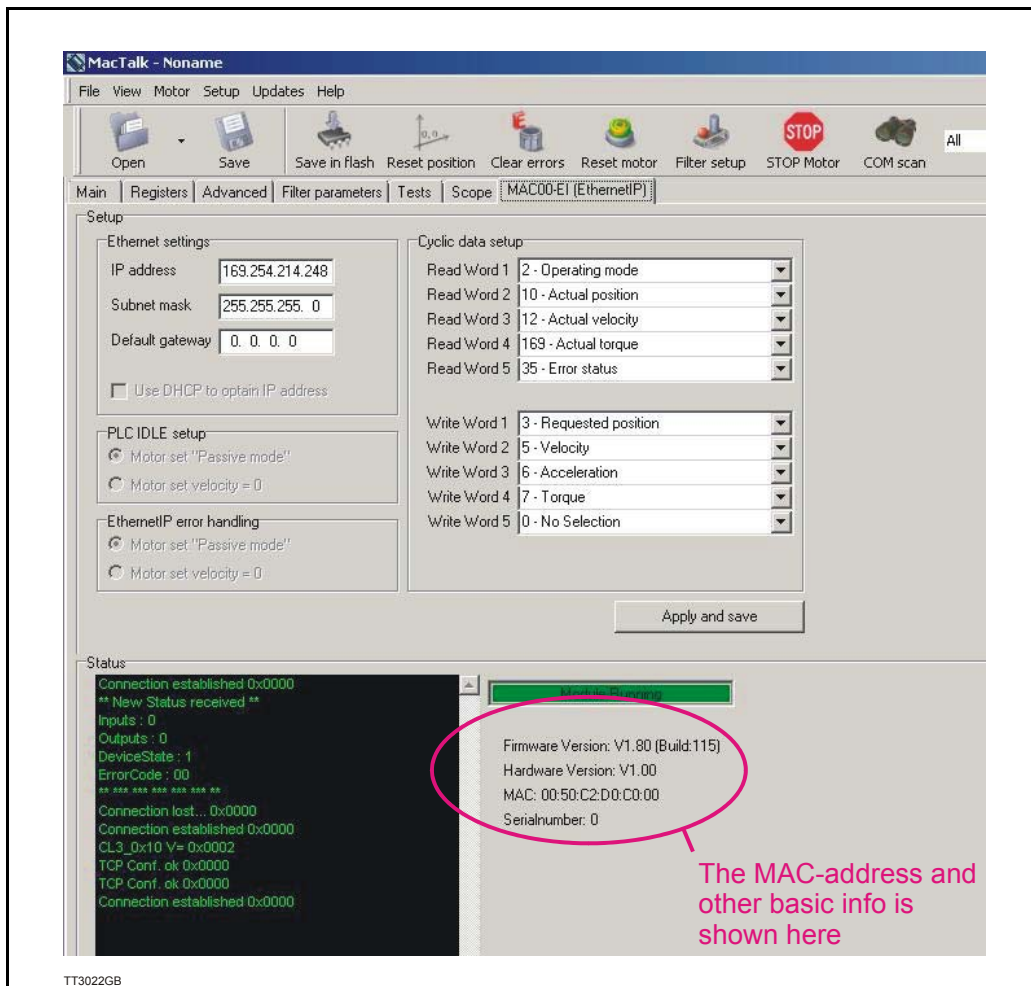
Blinking : Flashing with equal on and off periods of 200ms (2.5Hz). **Flickering** : Rapid flashing with a period of approx. 50ms (10 Hz).

4.4

Commissioning

4.4.2 MacTalk Ethernet configuration

The module is by default setup with the following Ethernet configuration:



After adjusting all settings press "Apply and save" for the settings to take effect and for permanently saving the setup.

Information such as EtherNet/IP firmware version, MAC-address and module status is displayed in the "Status" -field. Notice that the MAC-address is unique for each module and can not be changed.

A label at the frontplate of the module also indicate the MAC-address.

Basic use of MacTalk is described in the MAC-motor manual (lit. no. LB0047-xxGB)

4.5

Examples

4.5.1 Running Velocity control

To use the JVL motor in velocity mode the following registers are basically of interest.

1. "Mode" - Mode register register 2
2. "V_SOLL" - Velocity register 5
3. "A_SOLL" - Acceleration register 6
4. "Error/Status" - Error and status register 35

So, to control these registers the assembly object needs to be configured. From MacTalk the setup is configured as this.

Read Word	Value	Description
Read Word 1	12	Actual velocity
Read Word 2	10	Actual position
Read Word 3	198	Bus voltage
Read Word 4	169	Actual torque
Read Word 5	35	Error status

Write Word	Value	Description
Write Word 1	2	Operating mode
Write Word 2	3	Requested position
Write Word 3	5	Velocity
Write Word 4	7	Torque
Write Word 5	6	Acceleration

With the settings illustrated above we initiate the velocity mode by writing 0x1 to the first word-value, this is velocity mode.

From the scanner the registers is accessed using the assembly object and accessing the registers R/W on words 1-5.

Since different PLC's have different methods of implementation the basic steps is described in the following.

1. Set the needed velocity. $V_SOLL = V \times 2.77$ [rpm]
Ex. We need the motor to run with a constant speed of 1200 RPM. So, $V_SOLL = 1200/2.77 = 433$ cnt/smp
2. Set the needed acceleration. $A_SOLL = A \times 271$ [RPM/s²]
Ex. We need the motor to accelerate with 100000 RPM/s² so, $A_SOLL = 100000/271 = 369$ cnt/smp².
3. Now set the motor into velocity mode and thereby activate the motor.
Ex. The motor needs to be activated by setting it into velocity mode, so we need to set the mode register to the value 1. Mode = 1 which is velocity mode, now the motor will use the acceleration and the velocity just configured.

4.5

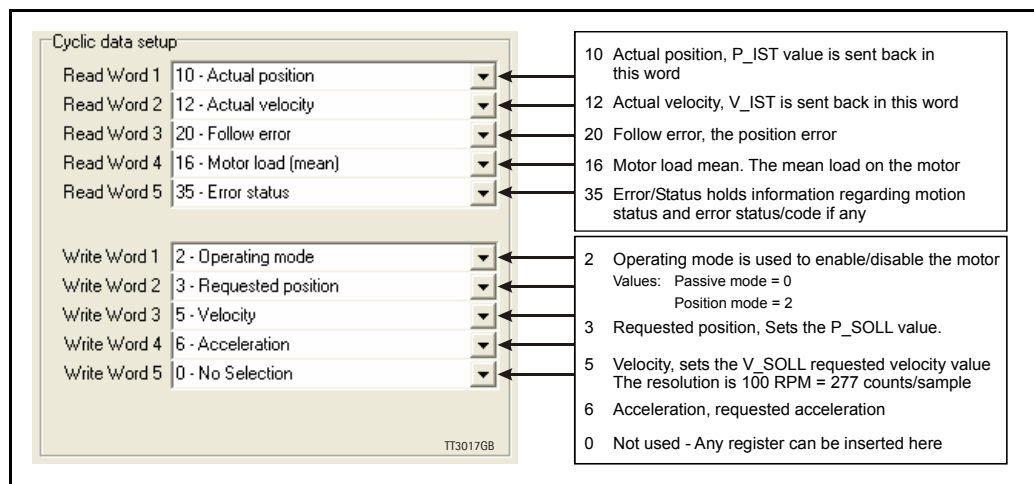
Examples

4.5.2 Running Position control

Running the motor in position control requires that the mode register is set for position control. The following registers is of particular interest when position mode is used.

1. "Actual position" -P_IST, register 10
2. "Actual velocity" -V_IST, register 12
3. "Follow error" - The actual position error, register 20
4. "Motor load mean" - average motor load, register 16
5. "Error/Status" -register 35
6. "Requested position" -P_SOLL, register 3
7. "Requested velocity" -V_SOLL, register 5
8. "Requested acceleration" -A_SOLL, register 6

In this mode the position is controlled by applying a requested position to the "P_SOLL" -register and the actual position is monitored in the "P_IST" register. The V_SOLL and A_SOLL registers sets the velocity and acceleration used when the actual positioning occurs.



4.5.3 General considerations

The register 35 in the motor holds information on the actual error/status. So it is crucial that this register is configured in the assembly object and thereby obtained and monitored in the scanner. In case of an error situation the motor will stop and the cause will be present in the register 35 and hence in the I/O -data.

This register also holds information on the motion status such as:

- In position, bit 4
- Accelerating, bit 5
- Decelerating, bit 6

For a complete register list please See "Motor registers" on page 53.

The JVL motor is basically put into a working mode and into a passive mode where the motor axle is de-energized, by setting register 2 into either 0 = "passive mode" or into one of the supported modes.

Example.

I = "Velocity mode" / 2 = "Position mode" / etc.

So in order to Stop or Start the motor this register can be supported in the I/O data or by sending an explicit message.

4.6 Configuration using different methods

Basically a JVL motor works by loading a configuration into RAM memory from the non volatile flash memory when 24V power is applied and the motor is initialized.

The motor only holds one configuration and this configuration can be stored into the NV flash memory.

Several approaches can be used to configure the motor with data and finally saving them permanently in the NV flash.

A very general approach could be by using the PC-based software tool MacTalk, which offers both basic motor setup and control and the possibility to save all parameters in a separate file for backup purposes.

This software package utilizes the serial connection to communicate with the motor from any standard Windows PC.

Configuration over EtherNet/IP is possible by using explicit messages to address each register to be setup and then command the motor to save the configuration in flash afterwards for permanent storage.

Using this method the motor only needs to be setup ones and is easy achievable from the scanner itself either as an initialization routine each time the PLC initializes, and thereby avoiding the permanent storage in the motor or simply using a configuration routine that simply sends the required explicit messages to address the needed registers followed by the message to save the settings permanently.

IP-address and other network settings still needs to be setup using MacTalk.

Ex. Setting up a motor sending messages explicitly

We want to change the default motor settings and save them permanently into flash. The following settings needs to be changed:

1. Velocity
2. Acceleration
3. Torque

The registers needed to be addressed are:

Velocity = V_SOLL = register 5
Acceleration = A_SOLL = register 6
Torque = T_SOLL = register 7

To address individual registers explicitly we use the class 0x64 for the purpose.

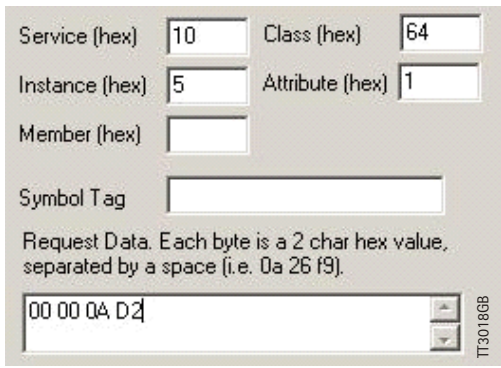
4.6 Configuration using different methods

First we change the Velocity setting, we want the motor to spin with 1000 RPM.

The message for addressing V_SOLL is formed:

We need to scale 1000 RPM to the correct value in the motor the factor is $1 \text{ RPM} = 2.77 \text{ counts/sample}$ so we need to send the value $2770 = 0x00000AD2$.

The instance refers to the register number, so we need to set instance to 5 (V_SOLL)
Please notice that the value is represented as 32bit.



The screenshot shows a configuration window with the following fields:

- Service (hex): 10
- Class (hex): 64
- Instance (hex): 5
- Attribute (hex): 1
- Member (hex): (empty)
- Symbol Tag: (empty)
- Request Data: 00 00 0A D2

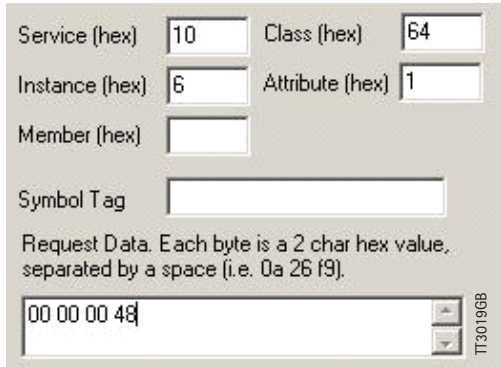
Below the Request Data field, there is a note: "Request Data. Each byte is a 2 char hex value, separated by a space (i.e. 0a 26 f9)." and a vertical label "TT3019GB" on the right side.

Next we set the acceleration to be used.

We need the acceleration to be 20000 RPM /s²

This value also needs to be scaled, the factor is $1 \text{ RPM/s}^2 = 0.0036 \text{ counts/sample}^2$ so, in order to reach 20000 we need to send the value $72 = 0x00000048$.

Acceleration is instance 6 (A_SOLL).



The screenshot shows a configuration window with the following fields:

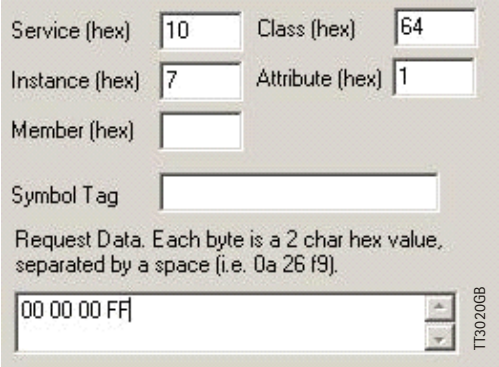
- Service (hex): 10
- Class (hex): 64
- Instance (hex): 6
- Attribute (hex): 1
- Member (hex): (empty)
- Symbol Tag: (empty)
- Request Data: 00 00 00 48

Below the Request Data field, there is a note: "Request Data. Each byte is a 2 char hex value, separated by a space (i.e. 0a 26 f9)." and a vertical label "TT3019GB" on the right side.

4.6 Configuration using different methods

Then configure the maximum motor torque to be used.

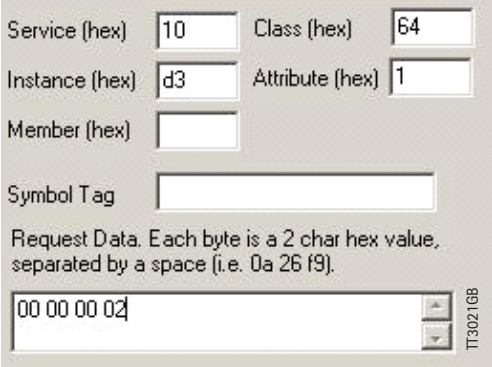
The motor can reach a peak torque of 300% the rated value. This value corresponds to 1023 in the register. We need 25% so we write $255 = 0x000000FF$ to instance 7 (T_SOLL).



Service (hex) Class (hex)
Instance (hex) Attribute (hex)
Member (hex)
Symbol Tag
Request Data. Each byte is a 2 char hex value, separated by a space (i.e. 0a 26 f9).
 TT3020GB

And finally we send the command that saves the settings permanently in flash. This is basically a matter of writing the "save in flash" command to the command register 211 in the motor. The command is 2 and the instance is $211 = 0xD3$. Value = $0x00000002$. Now the motor saves the setting and resets.

It is required to toggle the 24V power in order to do a internal synchronization.



Service (hex) Class (hex)
Instance (hex) Attribute (hex)
Member (hex)
Symbol Tag
Request Data. Each byte is a 2 char hex value, separated by a space (i.e. 0a 26 f9).
 TT3021GB

4.7 Using and Selecting an Ethernet switch

Depending on the network size and requested package interval (RPI) a suitable switch must be used. Also if multiple separated networks needs to be connected a switch is used.

Depending on the actual size of the network different requirements needs to be meet. Generally using EtherNet/IP with a fair package interval a 1 Gbps switch is typical adequate along with the following features:

- Autonegotiation, full duplex 100 MBit
- Port mirroring for network analyzing and troubleshooting purposes. This feature makes it possible to route traffic out on a separate port connected to a network analyzer for debugging purposes and general performance monitoring.

The JVL EtherNet/IP module has a small build in 2 port switch use full if a small amount of motors is connected in a daisy chaining topology.

The disadvantage of this approach is that the package RPI timing is reduced as each motor needs to handle the incoming traffic for the other motors connected on the line.

4.8 Objects accessible using Explicit messages

4.8.1 Type definitions:

UINT 16bit
DINT 32bit
STR String of ASCII-chars

4.8.2 Identity object class 0x01

Holds data on different module specific data.

Instance = 1

Attr. ID	Access	Name	Data type	Description
1	R	Vendor ID	UINT	JVL vendor ID = 936 (0x3A8)
2	R	Device Type	UINT	Value=10
3	R	Product code	UINT	Value = 1
4	R	Revision	UINT	Major = 1.byte, minor = 2. byte
5	R	Status	UINT	Status
6	R	Serial number	DINT	Serial number
7	R	Product name	STR	"MAC00-E1x"

See the EtherNet/IP spec. for further details section Vol2 sect.5-3.

Supported Services

0x1 Get_Attribute_All
0x10 Set_Attribute_Single
0xE Get_Attribute_Single

4.8 Objects accessible using Explicit messages

4.8.3 Assembly object class 0x04

Holds pre-configured motor registers to be accessed.

Instances

0x64 Write Data to motor register.

0x65 Read motor register data.

Attr. ID	Access	Name	Data type	Description
3	R/W	Get/Set Assembly	20 bytes	Get/Set all assembly data
4	R	Bytes	UINT	Bytes transferred in assembly

Supported Services

0x10 Set_Attribute_Single

0xE Get_Attribute_Single

This object can be used to access the predefined registers, configured from MacTalk.

They are also accessed when using the implicit connection cyclically.

If other registers than the one defined in the assembly object needs to be accessed then the class 0x64 needs to be used.

This class accesses each register in the motor for a more dynamically way of controlling registers explicitly.

The vendor specific class 0x64 is specified in details later in this chapter.

4.8.4 TCP/IP object class 0xF5

Holds data on different module specific data.

Attr. ID	Access	Name	Data type	Description
1	0xE	Status	DINT	Status bit-field
2	0xE	Configuration capability	DINT	DINTbitfield = 5 (BOOTP+DHCP)
3	0x10	Configuration control	DINT	Bitfield = 0 (use NV-setup)
4	0xE	Physical link object	6 bytes	Size+path
5	0x10	TCP/IP interface zup	22bytes	IP+subnet+GTW info etc.
6	0x10	Host name	DINT	Host name

See the EtherNet/IP spec. for further details section Vol2 sect.5-3.

Supported Services

0x1 Get_Attribute_All

0x10 Set_Attribute_Single

0xE Get_Attribute_Single

4.8 Objects accessible using Explicit messages

4.8.5 TCP/IP object class 0xF6

Holds information for a IEEE 802.3 communication interface

Attr. ID	Access	Name	Data type	Description
1	0xE	Interface speed	DINT	Speed in Mbit/s
2	0xE	Interface status	DINT	Bitfield
3	0xE	MAC-address	6 bytes	MAC
4	--	Not Implemented	--	--
5	--	Not Implemented	--	--
6	0x10	Interface Control	DINT	Bitfield

See EtherNet/IP spec. for further details Vol2 sect. 5-4

Supported Services

0x1 Get_Attribute_All
0x10 Set_Attribute_Single
0xE Get_Attribute_Single

4.8.6 Vendor specific JVL object class 0x64

Holds preconfigured motor registers to be accessed.

Instances

1..255 Motor registers

Attr. ID	Access	Name	Data type	Description
1	0xE / 0x10	Get/Set register	DINT	Get/Set the specified motor register

Supported Services

0x10 Set_Attribute_Single
0xE Get_Attribute_Single

4.9 **Examples of applications**

4.9.1 **Introduction**

The following pages contains an example of how the MAC motor can be implemented in a EtherNet-IP network.

4.9 Examples of applications

4.9.2 Rockwell RSLogix example 1.

This is a simple example demonstrating the usage of both explicit messages and IO-assemblies to control a JVL MAC400 servo motor.

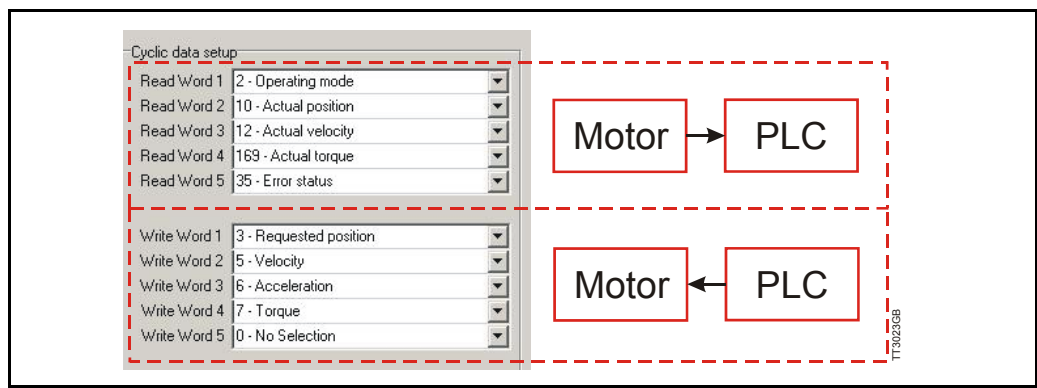
This example holds a few tags to control the inputs and outputs and a 3 rung ladder program to demonstrate the explicit message usage.

With this example it is possible to control the positioning of the motor using the "Position-mode" and set profile data such as velocity, acceleration and torque parameters using the IO-assembly.

The example is developed for use on a CompactLogix L23E PLC using the Rockwell Logix500 software package and MacTalk from JVL.

The JVL MacTalk application is used to setup the IO assembly to fit the example.

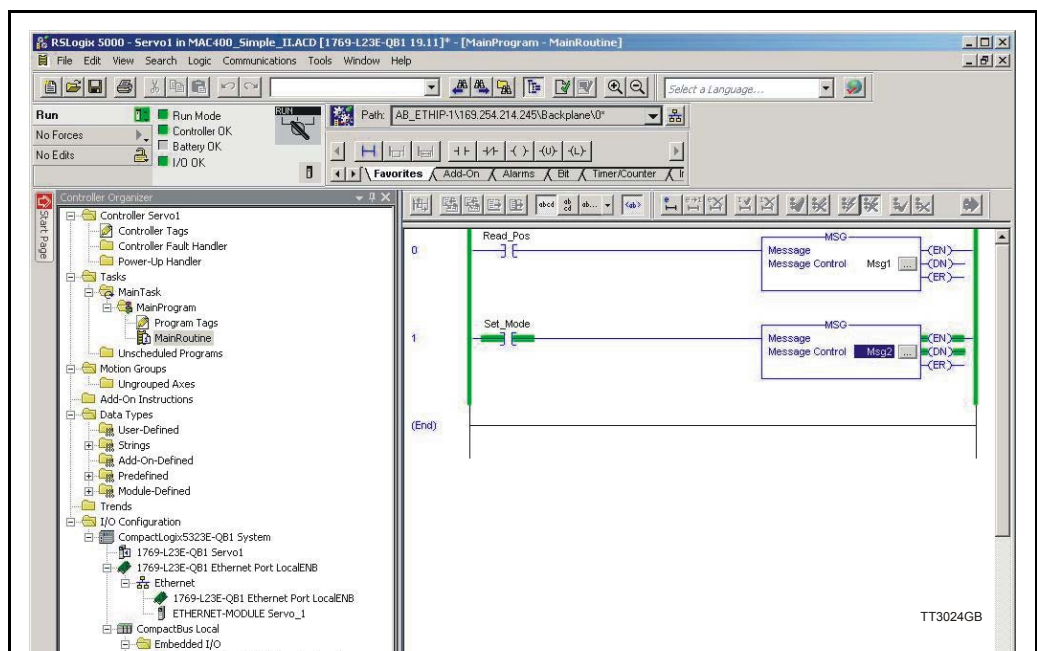
Although this example expects default setup in the JVL motor, the IO assembly needs to be setup according to the following MacTalk setup (located at the EthernetIP tab).



The fixed sized assembly instances is divided into 5 read words and 5 write words.

4.9.3 The RSLogix ladder program.

3 different messages for setting data and retrieving data from the motor. All 3 messages are triggered by separate variables from the controller tag-list.



4.9 Examples of applications

4.9.4 Message descriptions.

Msg1 reads information from the motor and is setup in the following way:
Reads (GET_ATTRIBUTE_SINGLE) the actual position register in the motor (instance 10) and stores the 4 byte value in the "ACTUAL POSITION" tag.

Message Configuration - Msg1

Configuration* | Communication | Tag

Message Type: CIP Generic

Service Type: Get Attribute Single

Service Code: e (Hex) Class: 64 (Hex) Instance: 10 Attribute: 1 (Hex)

Source Element:

Source Length: 0 (Bytes)

Destination: ActualPosition

New Tag...

Enable Enable Waiting Start Done Done Length: 4

Error Code: Extended Error Code: Timed Out

Error Path: Error Text:

OK Annuller Anvend Hjælp

TT3025GB

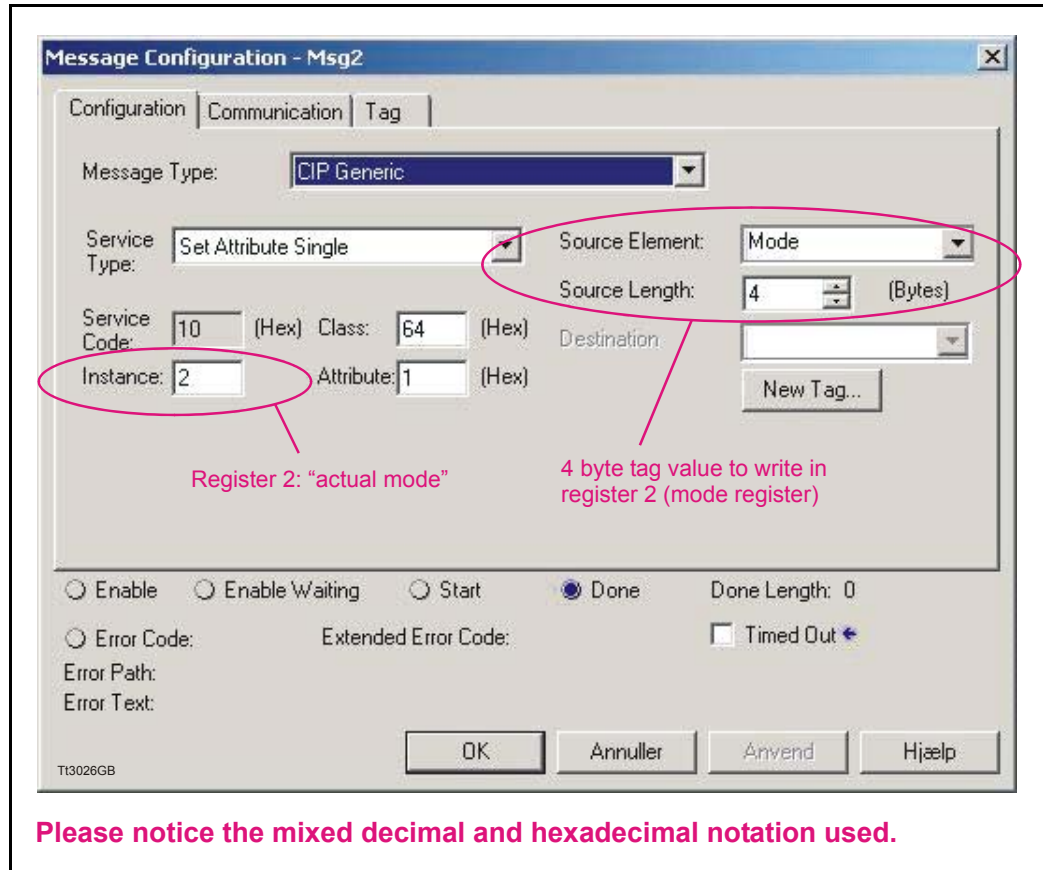
Register 10: "actual position"

Stores the value into this tag

Please notice the mixed decimal and hexadecimal notation used.

4.9 Examples of applications

Message 2 and 3 (Msg2, Msg3) are writing values to specific registers in the motor. They are configured in the following way:
Writes (SET_ATTRIBUTE_SINGLE) the value from the "MODE"-tag into the motor register 2 (Operation mode).



Explicit messages are always 4 bytes long and uses Class 0x64 to access the internal motor registers.

The instance refers to the actual motor register.

Instance = 2 points to the motor active mode -register.

Explicit messages are typical used for configuration purpose or for rare data update situation that does not require a cyclic update timing.

4.9 Examples of applications

4.9.5 Assembly data

The complete list of Controller tags defined.

The screenshot shows a table of controller tags for Servo1. The table has columns for Name, Value, Force Mask, Style, Data Type, and Description. A pink circle highlights the 'Servo_1:0' and 'Servo_1:1' entries. Below the screenshot are two detailed views: 'Write assembly' for Servo_1:0 and 'Read assembly' for Servo_1:1.

Name	Value	Force Mask	Style	Data Type	Description
ActualPosition	200000		Decimal	DINT	Variable that holds result from explicit msg1
diTemp	0		Decimal	DINT	used in msg3, set error = diTemp
Local1:C	{...}	{...}		AB:Embedded_IQ...	
Local1:I	{...}	{...}		AB:Embedded_IQ...	
Local2:C	{...}	{...}		AB:Embedded_D...	
Local2:I	{...}	{...}		AB:Embedded_D...	
Local2:O	{...}	{...}		AB:Embedded_D...	
Mode	2		Decimal	DINT	Used in msg2, more = mode (1= velocity, 2=position)
Msg1	{...}	{...}		MESSAGE	
Msg2	{...}	{...}		MESSAGE	
Msg3	{...}	{...}		MESSAGE	
Oneshut	0		Decimal	BOOL	Triggers explicit msg2, set mode
Oneshut2	0		Decimal	BOOL	Triggers explicit msg3, set error=diTemp
Run1	0		Decimal	BOOL	Triggers explicit msg1, get actual position
Run2	0		Decimal	BOOL	Not Used
Servo_1:C	{...}	{...}		AB:ETHERNET_...	
Servo_1:I	{...}	{...}		AB:ETHERNET_...	Read words, see MacTalk
Servo_1:I.Data	{...}	{...}	Decimal	DINT[5]	Read words, see MacTalk
Servo_1:I.Data[0]	2		Decimal	DINT	Read words, see MacTalk
Servo_1:I.Data[1]	200000		Decimal	DINT	Read words, see MacTalk
Servo_1:I.Data[2]	0		Decimal	DINT	Read words, see MacTalk
Servo_1:I.Data[3]	0		Decimal	DINT	Read words, see MacTalk
Servo_1:I.Data[4]	524304		Decimal	DINT	Read words, see MacTalk
Servo_1:O	{...}	{...}		AB:ETHERNET_...	Write words see MacTalk
Servo_1:O.Data	{...}	{...}	Decimal	DINT[5]	Write words see MacTalk
Servo_1:O.Data[0]	200000		Decimal	DINT	Write words see MacTalk
Servo_1:O.Data[1]	8000		Decimal	DINT	Write words see MacTalk
Servo_1:O.Data[2]	2		Decimal	DINT	Write words see MacTalk
Servo_1:O.Data[3]	512		Decimal	DINT	Write words see MacTalk
Servo_1:O.Data[4]	0		Decimal	DINT	Write words see MacTalk

Name	Value
Servo_1:0	{...}
Servo_1:0.Data	{...}
Servo_1:0.Data[0]	200000
Servo_1:0.Data[1]	8000
Servo_1:0.Data[2]	2
Servo_1:0.Data[3]	512
Servo_1:0.Data[4]	0

Name	Value
Servo_1:1	{...}
Servo_1:1.Data	{...}
Servo_1:1.Data[0]	2
Servo_1:1.Data[1]	200000
Servo_1:1.Data[2]	0
Servo_1:1.Data[3]	0
Servo_1:1.Data[4]	524304

4.9 Examples of applications

MacTalk IO assembly setup, seen in the controller tag list and read from the PLC when the connection has been established.

Cyclic data setup

Read Word 1: 2 - Operating mode

Read Word 2: 10 - Actual position

Read Word 3: 12 - Actual velocity

Read Word 4: 169 - Actual torque

Read Word 5: 35 - Error status

Write Word 1: 3 - Requested position

Write Word 2: 5 - Velocity

Write Word 3: 6 - Acceleration

Write Word 4: 7 - Torque

Write Word 5: 0 - No Selection

Servo_1:1		{...}
Servo_1:1.Data		{...}
+ Servo_1:1.Data[0]		2
+ Servo_1:1.Data[1]		200000
+ Servo_1:1.Data[2]		0
+ Servo_1:1.Data[3]		0
+ Servo_1:1.Data[4]		524304

Servo_1:0		{...}
Servo_1:0.Data		{...}
+ Servo_1:0.Data[0]		200000
+ Servo_1:0.Data[1]		8000
+ Servo_1:0.Data[2]		2
+ Servo_1:0.Data[3]		512
+ Servo_1:0.Data[4]		0

Explanation

2 - Operating Mode = 2 (position mode)

10 - Actual Position = 200000

12 - Actual Velocity = 0 Cnt/s

169 - Actual Torque = 0 (1024 = 300%)

35 - Error Status = 524304 (no errors)

Explanation

3 - Requested position = 200000

5 - Velocity = 8000 (8000 = 2820 RPM)

6 - Acceleration = 2 Cnt/s² (2 = 543 RPM/s²)

7 - Torque = 512 (512 = 150%)

0 - No Selection (value is not updated)

TT3028GB

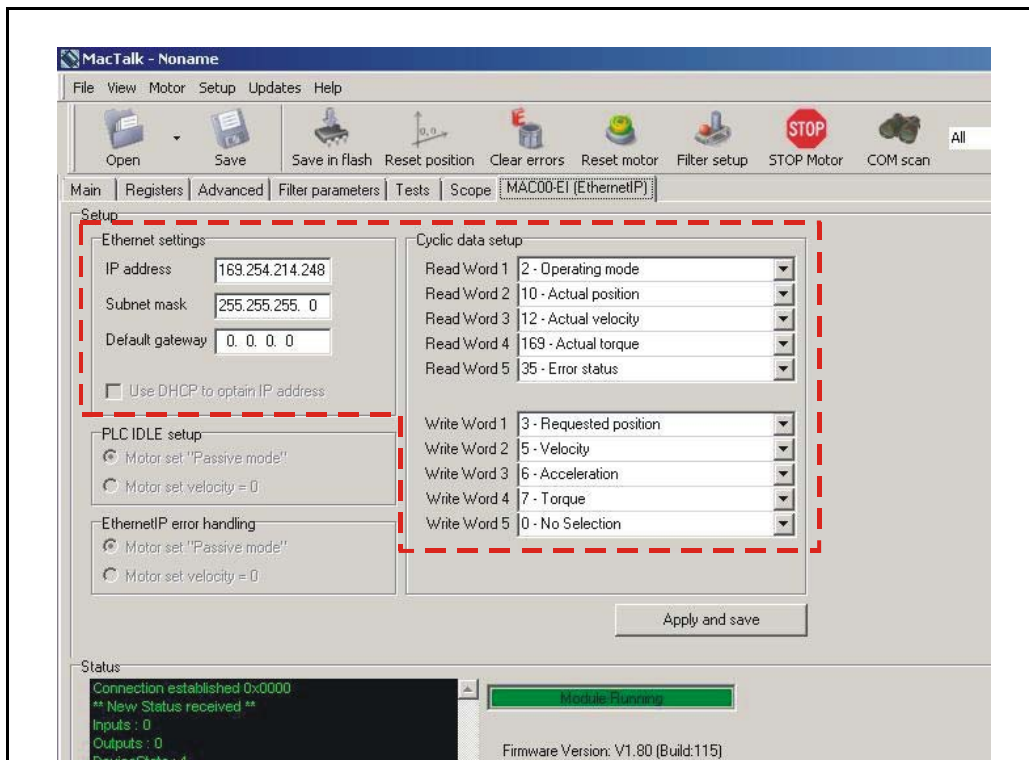
4.9 Examples of applications

4.9.6 How to use the Example Step by Step.

Setting up IP addresses and general usage of the Rockwell CompactLogix PLC with the software package Logix5000 is beyond the scope of this example.

The following guideline is based on the JVL MAC400 motor with the factory setup.

1. Apply 24V, open MacTalk and setup the ethernet settings as required and the IO assembly (cyclic data setup) according to the following:



2. Press the "Apply and save" -button for permanent storage of the EthernetIP -settings.
3. Switch off the 24V supply while connecting the Ethernet cable to the switch/PLC.
4. Re-apply 24V set the PLC into "RUN" -mode. Now we should be able to control the motor.
5. Start by setting the profile data such as, Velocity, acceleration and Torque. According to the following:

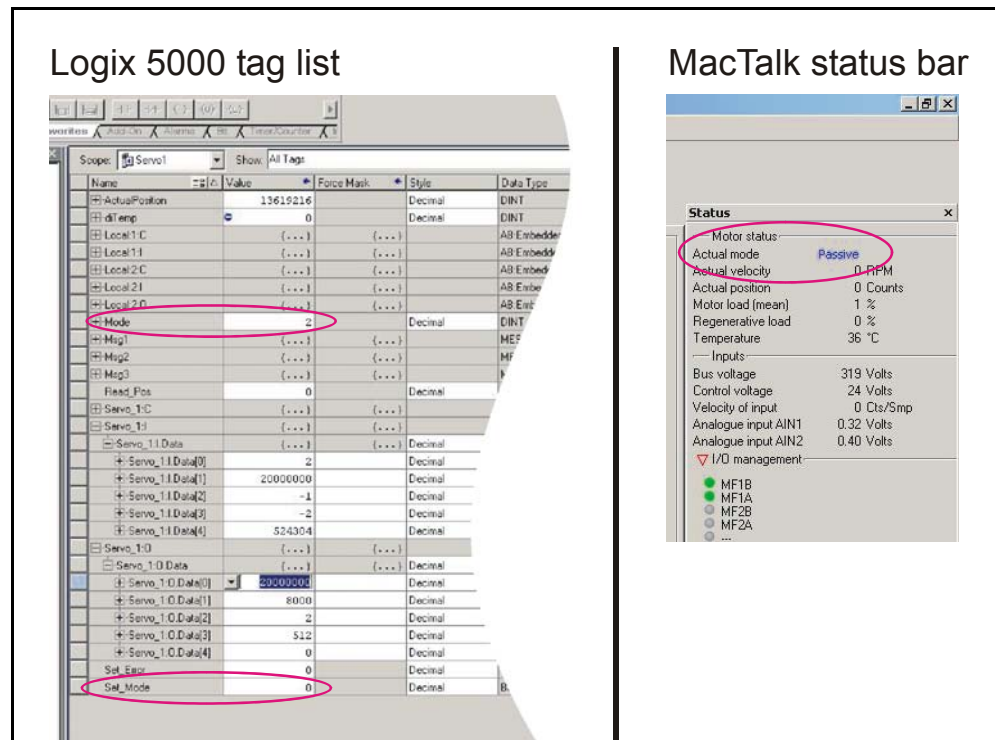
		Explanation
[-] Servo_1:0	{...}	
[-] Servo_1:0.Data	{...}	
[+] Servo_1:0.Data[0]	200000	3 - Requested position = 200000
[+] Servo_1:0.Data[1]	8000	5 - Velocity = 8000 (8000 = 2820 RPM)
[+] Servo_1:0.Data[2]	2	6 - Acceleration = 2 Cnt/s ² (2 = 543 RPM/s ²)
[+] Servo_1:0.Data[3]	512	7 - Torque = 512 (512 = 150%)
[+] Servo_1:0.Data[4]	0	0 - No Selection (value is not updated)

TT3031GB

4.9

Examples of applications

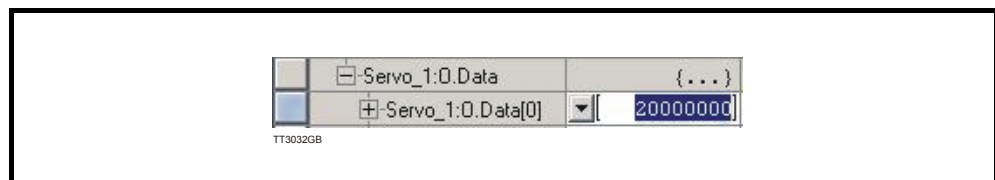
- Now we will set the motor into an active mode (position mode), find the Controller tag "Mode" enter 2, find the tag "Set_Mode" enter 1. Now the motor is active and will start moving to the entered position in the "Servo_1:O_Data[0]" which is assigned to the requested position register in the motor. When the motor reaches the position it will stop and hold this position.
From MacTalk the actual mode (see the status-panel) is changed from "Passive" to Position and the motion progress can be followed. Remember to change the "Set_Mode" tag back to 0 to stop the sending of Msg2 -messages.



Changing the "Servo_1:O_Data[0]"-tag will result in an immediate repositioning of the axle in the motor. This value is defined in the IO assembly and is interchanged cyclic.

To stop the motor set "Mode" = 0 and set "Set_Mode" = 1 to apply the mode setting. Reset "Set_Mode" to 0 again to stop sending Msg2. -messages.

- To activate the explicit message Msg1 set the commanded position to a far greater value. For example 200000000 as illustrated below.



- Find the "Read_Pos" -tag and set this to 1. Now the current position of the motor is seen in the "Actual Position" -tag.

5.1

Technical Data

5.1.1 **MAC00-EI4 EthernetIP - Technical specifications**

Galvanic isolated, 100MBit, 100Base-Tx, no termination necessary.

Network topology:

Max. 100 m cable between slaves.

Connectors: "PWR" (power) M12 connector 5pin male

"I/O" M12 connector 8pin female

"L/A IN" and "L/A OUT" (Ethernet) M12 connector 4pin D-coded female.

Supply voltage (CV): 10-25V

Current rating (CV): typical 150mA, max. 250mA

User inputs:

Input impedance: 4.7k

Input current @24V: 5.1mA

MAC00-EC4 EtherCAT - Technical specifications

Galvanic isolated, 100MBit, 100Base-Tx, no termination necessary.

Network topology: Line and tree possibly (line recommended)

Max. 100 m cable between slaves.

Maximum number of slaves: 65535

Pass trough delay: < 4 μ s.

Connectors: "PWR" (power) M12 connector 5pin male

"I/O" M12 connector 8pin female

"L/A IN" and "L/A OUT" (Ethernet) M12 connector 4pin D-coded female.

Supply voltage (CV): 10-25V

Current rating @ 24V DC (CV): typical 150mA, max. 250mA

User inputs:

Input impedance: 4.7k

Input current @24V: 5.1mA

5.2

Motor registers

Only MAC400 & 800

When using the RS232 or RS422 serial links, it is possible to access all the internal registers in the motor.

This gives the same possibilities as using the general installation and monitoring program MacTalk.

In addition to these features, many more are accessible. In total, the MAC motor contains more than 200 internal registers such as nominal velocity, actual position, etc.

Important note:

All registers can be read without any risk but please note that several registers are not for the normal user and damage may occur if the contents of these registers is changed. These registers are marked in grey in the table below.

Main Control					
Reg. no.	Name	Width	Unit	Description	MacTalk name
1	PROG_VER.	Long int	-	Shows the actual version of the firmware. Bit0-5: Minor version Bit6-12: Major version Bit13: (if set) Beta version Bit14: Reserved Bit15: (if set) MAC400 or MAC800	(status bar)
2	MODE_REG	Long int	-	The current MAC motor mode: (see also register 37 - "Start mode") 0: Passive 1: Velocity 2: Position 3: Gear Mode 4: Analog Torque (direct) 5: Analog Velocity 6: Analog Velocity/Gear. 7-11: Reserved for special purposes 12: Torque zero search 13: Sensor type 1 zero search 14: Sensor type 2 zero search 16: Analogue velocity (with deadband) 17: Velocity/analogue torque 18: Analogue gear 19: Coil 20: Air cylinder 21: Analogue to position	Mode
3	P_SOLL	Long Int	Encoder counts	The commanded position	Position
4	P_NEW	Long Int	Encoder counts	Offset position for position change	-
5	V_SOLL	Long Int	Counts/sample/16	Desired velocity 1 RPM=2.77056 counts/sample. Example: To obtain 100 RPM, V_SOLL must be set to 277.	Max velocity
6	A_SOLL	Long Int	Counts/sample ² /16	The desired nominal acceleration. 1000 RPM/s = 3.598133 counts/Sample ² Example: To obtain 100000 RPM/s, A_SOLL must be set to 360.	Acceleration
7	T_SOLL	Long Int	-	The maximum allowed torque. 0-1023. 1023 = 300% (full peak torque).	Torque
8	P_FUNC	Long Int	Encoder counts	-	-
9	INDEX_OFFSET	Long Int	Encoder counts	Distance from encoder index to ext. sensor	-
10	P_IST	Long Int	Encoder counts	The actual motor position	Actual position
11	V_IST_16	Long Int	Counts/sample/16	V_IST (actual speed) measured over 16 samples Same unit as V_SOLL (register 5).	Actual velocity
12	V_IST	Long Int	Counts/sample	Actual velocity. 1RPM=0.17316 counts/sample.	-
13	KVOUT	Fixed 16	-	Overall servo filter inertia factor.	Load
14	GEARF_1	Long Int	-	Gear output factor. Used in gear mode	Input
15	GEARF_2	Long Int	-	Gear input factor. Used in gear mode	Output

5.2

Motor registers

Only MAC400 & 800

Error Handling					
Reg. no.	Name	Width	Unit	Description	MacTalk name
16	I2T	Long Int	-	Motor temperature calculated. The value is integrated during motor operation. If it reaches 100% the overload bit in reg 35 (ERR_STAT bit 0) is set indicating that the motor torque has passed the allowable continues rating = nominal torque.	Motor load (mean)
17	I2TLIM	Long Int	-	Error trip level used for I2T register.	-
18	UIT	Long Int	-	Returned energy from the motor (load). If the value passes 100% the UIT bit in register 35 (ERR_STAT bit 3) is set indicating that too much energy has been returned from the motor (load). Connect an external dump resistor or decrease deceleration.	Regenerative load
19	UITLIM	Long Int	-	Error trip level used for UIT register	-
20	FLWERR	Long Int	Encoder counts	Actual follow error	Follow error
21	U_24V	Long Int	VDC/74.4713	Logic supply voltage measurement. Logic supply voltage [VDC] = U_24V x 0.013428	Logic supply
22	FLWERRMAX	Long Int	Encoder counts	Follow error limit. If the follow error passes this limit the motor will be stopped and the FLW_ERR in register 35 will be set.	EH:Follow error
23	UV_HANDLE	Long Int	-	Register to specify action when undervoltage is detected. Bit 0: (SET_UV_ERR) Error if under voltage Bit 1: (UV_GO_PASSIVE) Go to passive mode Bit 2: (UV_VSOLL0) Set speed=0 if u.volt.	Set error bit Go to passive Set velocity to 0
24	FNCERR	Long Int	-	Actual function error	Function error
26	FNCERRMAX	Long Int	-	Function error limit. If the function error passes this limit the motor will be stopped and the FNC_ERR in register 35 will be set.	EH:Function error
27	UVMIN	Long Int	-	Register not used	-
28	MIN_P_IST	Long Int.	Encoder counts	Software position limit - positive	Position limit max
29	DEGC	Long Int.	-	Actual temperature. Degree celcius=DEGC x 0.12207	Temperature
30	MAX_P_IST	Long int.	Encoder counts	Software position limit - negative	Position limit min
31	DEGCMAX	Long int.	-	Temperature limit. Same scale as DEGC (reg 29). If temperature gets higher than this limit the DEGC_ERR in register 35 is set	-
32	ACC_EMERG	Long Int	Counts/sample ² /16	The maximum allowed deceleration when a fatal error has occurred. 1000 RPM/s = 3.598133 counts/Sample ² . Example: To obtain 100000 RPM/s, ACC_EMERG must be set to 360.	Error acceleration
33	INPOSWIN	Long Int	Encoder counts or encoder counts/sample	If the target position or velocity is reached within the tolerance specified in this window, the motor is in position or at the velocity.	In pos. window / At vel. window
34	INPOSCNT	Long Int	Samples	The number of samples the motor has to be within the pos. interval spec. in INPOSWIN.	In pos. count
35	ERR_STAT	Long Int		Motor error status: Bit 0: (I2T_ERR) Overload Bit 1: (FLW_ERR) Follow error Bit 2: (FNC_ERR) Function error Bit 3: (UIT_ERR) Regenerative error Bit 4: (IN_POS) In position Bit 5: (ACC_FLAG) Accelerating Bit 6: (DEC_FLAG) Decelerating Bit 7: (PLIM_ERR) Position limits error Bit 8: (DEGC_ERR) Temperature error (>DEGCMAX) Bit 9: (UV_ERR) Under voltage error Bit 10: (UV_DETECT) Low voltage at the high volt bus Bit 11: (OV_ERR) Overvoltage error. UB>450V Bit 12: (IPEAK_ERR) Motor over current Bit 13: (SPEED_ERR) Overspeed - >3600RPM Bit 14: (DIS_P_LIM) Software position limits disabled Bit 15: (INDEX_ERR) Internal encoder error Bit 16: (OLD_FILT_ERR) Filter setting not valid Bit 17: (U24V_ERR) Control supply has been too low Bit 18: (SHORT_CIRC) M. Current has been too high Bit 19: AC (>90VAC) supply applied Bit 20: -	Overload Follow Error Function Error Regen. Overload In position Accelerating Decelerating Position Limit Temp. too high Under bus volt. Low bus voltage Over bus voltage Peak Error Overspeed - Internal error 1 Internal error 2 Cntr. Volt unstab. Short circuit -

5.2

Motor registers

Only MAC400 & 800

Power + zero search handling					
Reg. no.	Name	Width	Unit	Description	MacTalk name
36	CNTRL_BITS	Long Int	-	Internal special bits: Bit 0: (RECORDBIT) : Controls the samplebuffer Bit 1: (REWINDBIT) : Controls the samplebuffer Bit 2: (RECINNERBIT) : Controls the samplebuffer Bit 3: (RELPOSPSOLL) : Relative move using P_SOLL Bit 4: (RELPOSPFNC) : Relative move using P_FNC Bit 5: (SYNCPOSAUTO) : Synchronize int. Position regs Bit 6: (SYNCPOSAMAN) : Same as bit 5 but manually Bit 7: (MAN_NO_BRAKE) : Disables the brake if set Bit 8: (SYNCPOSREL) : Offset P_IST with P_NEW Bit 9: (INDEX_HOME) : Use index after zero search	- - - - Reg move type Reg move type Resync pos.... - Disable brake - Use index aft...
37	START_MODE	Long Int	-	The mode used after power up. See also register 2.	(Mode)
38	P_HOME	Long Int	Encoder counts	Motor position after zero search	Zero search position
39	HW_SETUP	Long Int	-	Hardware setup bits: Bit 0: (DIRAWR) Bit 1: (DIRBWR) Bit 2: (PULSEOUT) Bit 3: (XSEL1) Bit 4: (XPRINP) Bit 5: (NOFILT) Bit 6: (INVXDIR)	-
40	V_HOME	Long Int	Counts/sample/16	Speed used during zero search. Speed defined as register 5	Zero search speed
41	T_HOME	Long Int	-	Torque used for Torque zero search. The sign defines polarity of the zero search sensor.	Zero search torque
42	HOME_MODE	Long Int	-	Zero search mode. The type of zero search. Bit 16: (Home_Done) bit 16.	Zero search mode

5.2

Motor registers

Only MAC400 & 800

Registers (P0-7, V0-7 etc.)					
Reg. no.	Name	Width	Unit	Description	MacTalk name
43	P_REG_P	Long Int	-	-	-
44	V_REG_P	Long Int	-	-	-
45	A_REG_P	Long Int	-	-	-
46	T_REG_P	Long Int	-	-	-
47	L_REG_P	Long Int	-	-	-
48	Z_REG_P	Long Int	-	-	-
49	POS0	Long Int	Encoder counts	Position register P1. Used with the fastmac protocol or by the MAC00-R1/3/4 nanoPLC module. See also P_SOLL (register 3)	P1
51	POS1	Long Int	Encoder counts	Position register P2 - see also register 49.	P2
53	POS2	Long Int	Encoder counts	Position register P3 - see also register 49.	P3
55	POS3	Long Int	Encoder counts	Position register P4 - see also register 49.	P4
57	POS4	Long Int	Encoder counts	Position register P5 - see also register 49.	P5
59	POS5	Long Int	Encoder counts	Position register P6 - see also register 49.	P6
61	POS6	Long Int	Encoder counts	Position register P7 - see also register 49.	P7
63	POS7	Long Int	Encoder counts	Position register P8 - see also register 49.	P8
65	VEL0	Long Int	Counts/sample/16	Velocity register V1. Used with the fastmac protocol or by the MAC00-R1/3/4 nanoPLC module. See also V_SOLL (register 5)	V1
66	VEL1	Long Int	Counts/sample/16	Velocity register V2 - see also register 65.	V2
67	VEL2	Long Int	Counts/sample/16	Velocity register V3 - see also register 65.	V3
68	VEL3	Long Int	Counts/sample/16	Velocity register V4 - see also register 65.	V4
69	VEL4	Long Int	Counts/sample/16	Velocity register V5 - see also register 65.	V5
70	VEL5	Long Int	Counts/sample/16	Velocity register V6 - see also register 65.	V6
71	VEL6	Long Int	Counts/sample/16	Velocity register V7 - see also register 65.	V7
72	VEL7	Long Int	Counts/sample/16	Velocity register V8 - see also register 65.	V8
73	ACC0	Long Int	Counts/sample ² /16	Acceleration register A1. Used with the fastmac protocol or by the MAC00-R1/3/4 nanoPLC module. See also A_SOLL (register 6)	A1
74	ACC1	Long Int	Counts/sample ² /16	Acceleration register A2 - see also register 73.	A2
75	ACC2	Long Int	Counts/sample ² /16	Acceleration register A3 - see also register 73.	A3
76	ACC3	Long Int	Counts/sample ² /16	Acceleration register A4 - see also register 73.	A4
77	TQ0	Long Int	-	Torque register T1. Used with the fastmac protocol or by the MAC00-R1/3/4 nanoPLC module. See also T_SOLL (register 7)	T1
78	TQ1	Long Int	-	Torque register T2 - see also register 77.	T2
79	TQ2	Long Int	-	Torque register T3 - see also register 77.	T3
80	TQ3	Long Int	-	Torque register T4 - see also register 77.	T4
81	LOAD0	Fixed16	-	Load register L1. Used with the fastmac protocol or by the MAC00-R1/3/4 nanoPLC module. See also KVOUT (register 13)	L1
82	LOAD1	Fixed16	-	Load register L2 - see also register 81.	L2
83	LOAD2	Fixed16	-	Load register L3 - see also register 81.	L3
84	LOAD3	Fixed16	-	Load register L4 - see also register 81.	L4
85	ZERO0	Long Int	-	In position register Z1. Used with the fastmac protocol or by the MAC00-R1/3/4 nanoPLC module. See also INPOSWIN (register 33)	Z1
86	ZERO1	Long Int	-	In position register Z2 - see also register 81.	Z2
87	ZERO2	Long Int	-	In position register Z3 - see also register 81.	Z3
88	ZERO3	Long Int	-	In position register Z4 - see also register 81.	Z4

Registers 89 to 120 are reserved for future purposes.

5.2

Motor registers

Only MAC400 & 800

Filters (main 6.th. order servo filter)					
Reg. no.	Name	Width	Unit	Description	MacTalk name
121	KFF5	Fixed 24	-	-	-
122	KFF4	Fixed 24	-	-	-
123	KFF3	Fixed 24	-	-	-
124	KFF2	Fixed 24	-	-	-
125	KFF1	Fixed 24	-	-	-
126	KFF0	Fixed 24	-	-	-
127	KVFX6	Fixed 16	-	-	-
128	KVFX5	Fixed 16	-	-	-
129	KVFX4	Fixed 16	-	-	-
130	KVFX3	Fixed 16	-	-	-
131	KVFX2	Fixed 16	-	-	-
132	KVFX1	Fixed 16	-	-	-
133	KVFX0	Fixed 16	-	-	-
134	KVFX0	Fixed 16	-	-	-
135	KVFX0	Fixed 16	-	-	-
136	KVFX0	Fixed 16	-	-	-
137	KVFX0	Fixed 16	-	-	-
138	KVFX0	Fixed 16	-	-	-
139	KVFX0	Fixed 16	-	-	-
140	KVFX0	Fixed 16	-	-	-
141	KVFX0	Fixed 16	-	-	-
142	KVFX0	Fixed 16	-	-	-
143	KVFX0	Fixed 16	-	-	-
144	KVFX0	Fixed 16	-	-	-
145	KVFX0	Fixed 16	-	-	-
146	KVFX0	Fixed 16	-	-	-
147	KVFX0	Fixed 16	-	-	-
148	KVFX0	Fixed 16	-	-	-
149	KVFX0	Fixed 16	-	-	-
154	MODEL_POT	Long Int	-	-	-
156	S_ORDER	Long Int	-	-	-
157	OUTLOOPDIV	Long Int	-	-	-

Sample registers					
Reg. no.	Name	Width	Unit	Description	MacTalk name
158	SAMPLE1	Long Int	-	-	-
159	SAMPLE2	Long Int	-	-	-
160	SAMPLE3	Long Int	-	-	-
161	SAMPLE4	Long Int	-	-	-
162	REC_CNT	Long Int	-	-	-

5.2

Motor registers

Only MAC400 & 800

Outer loop registers					
Reg. no.	Name	Width	Unit	Description	MacTalk name
163	V_EXT	Long Int	-	Speed at the external pulseinput (if used)	Velocity of input
164	GV_EXT	Long Int	-	-	-
165	G_FNC	Long Int	-	-	-
166	FNC_OUT	Fixed 16	-	-	-
167	FF_OUT	Long Int	-	-	-
168	VB_OUT	Long Int	-	-	-
169	VF_OUT	Long Int	-	Actual motor torque. See also T_SOLL (register 7)	Actual motor torque
170	ANINP	Long Int	-	Analogue input voltage. VDC = ANINP x 0.0048828	Analogue input
171	ANINP_OFFSET	Long Int	-	Analogue input offset. Same scale as ANINP (170)	Analogue input offset

Inner loop registers					
Reg. no.	Name	Width	Unit	Description	MacTalk name
172	ELDEG_OFFSET	Long Int	-	-	-
173	PHASE_COMP	Long Int	-	-	-
174	AMPLITUDE	Long Int	-	-	-
175	MAN_I_NOM	Fixed 16	-	-	-
176	MAN_ALPHA	Long Int	-	-	-
177	UMEAS	Long Int	-	-	-
178	I_NOM	Long Int	-	-	-
179	PHI_SOLL	Long Int	-	-	-
180	IA_SOLL	Long Int	-	-	-
181	IB_SOLL	Long Int	-	-	-
182	IC_SOLL	Long Int	-	-	-
183	IA_IST	Long Int	-	-	-
184	IB_IST	Long Int	-	-	-
185	IC_IST	Long Int	-	-	-
186	IA_OFFSET	Long Int	-	-	-
187	IB_OFFSET	Long Int	-	-	-
188	KIA	Long Int	-	-	-
189	KIB	Long Int	-	-	-
190	ELDEG_IST	Long Int	-	-	-
191	V_ELDEG	Long Int	-	-	-
192	UA_VAL	Long Int	-	-	-
193	UB_VAL	Long Int	-	-	-
194	UC_VAL	Long Int	-	-	-
195	EMK_A	Long Int	-	-	-
196	EMK_B	Long Int	-	-	-
197	EMK_C	Long Int	-	-	-
198	U_BUS	Long Int	-	Internal busvoltage. 1VDC = 0.888798. Example: U_BUS = 366 is equal to 325VDC at the internal bus.	Bus voltage
199	U_BUS_OFFSET	Long Int	-	-	-
200	TC0_CV1	Long Int	-	-	-
201	TC0_CV2	Long Int	-	-	-

5.2

Motor registers

Only MAC400 & 800

Diverse					
Reg. no.	Name	Width	Unit	Description	MacTalk name
202	MY_ADDR	Long Int	-	Motor adress	Motor address
203	MOTOR_TYPE	Long Int	-	Type of the MAC motor	-
204	SERIAL_NUMBER	Long Int	-	The serial number of the MAC motor	-
205	HW_VERSION	Long Int	-	Hardware version	-
206	CHKSUM	Long Int	-	Firmware checksum	-

A

AIN 14
Air Cylinder mode 14
Analogue Input
 AIN 14

C

Cables 18
Connectors 15–18
 M12 16–18

E

Error output 8
Expansion modules
 MAC00-B1/B2/B4 12–18

F

Features 8

G

GND 15, 17
Grounding 15–16

I

In position output 8
Inputs
 See also AIN
 Multifunction I/O 12, 17
 Pulse inputs 12

Introduction
 Features 8

IP67 18

M

M12 16–18
MAC00-B1/B2/B4 Expansion Modules 12–18
 General analogue input (AIN) 14
 General hardware aspects 10
 MAC00-B4 cables 18
 Power supply 13
 RS232 15

MacTalk 15

Main Features 8

R

RS232
 MAC00-B1/B2/B4 15

Z

Zero search 14, 17

