Smart encoders & actuators

User's guide

RD6

EtherCAT®

Modbus

RS-232 version

- Rotary actuator with EtherCAT interface in compliance with ETG.1000 specifications
- Includes CAN Application protocol over EtherCAT (CoE) and EtherCAT State Machine
- BLDC motor sizes: P_N 157 W, M_N 0.5 Nm, n_N 3000 rpm
  P_N 250 W, M_N 0.8 Nm, n_N 3000 rpm
- Real multiturn absolute encoder 4096 cpr x 65536 rev.
- For change-over operations and automated positioning systems

Suitable for the following models:

- RD6-P8-157-EC-...
- RD6-P8-250-EC-...

General Contents

lika

General contents

Subject Index

Typographic and iconographic conventions

In this guide, to make it easier to understand and read the text the following typographic and iconographic conventions are used:

- parameters and objects both of Lika device and interface are coloured in GREEN;
- alarms are coloured in RED;
- states are coloured in FUCSIA.

When scrolling through the text some icons can be found on the side of the page: they are expressly designed to highlight the parts of the text which are of great interest and significance for the user. Sometimes they are used to warn against dangers or potential sources of danger arising from the use of the device. You are advised to follow strictly the instructions given in this guide in order to guarantee the safety of the user and ensure the performance of the device. In this guide the following symbols are used:

	This icon, followed by the word **WARNING**, is meant to highlight the parts of the text where information of great significance for the user can be found: user must pay the greatest attention to them! Instructions must be followed strictly in order to guarantee the safety of the user and a correct use of the device. Failure to heed a warning or comply with instructions could lead to personal injury and/or damage to the unit or other equipment.
	This icon, followed by the word **NOTE**, is meant to highlight the parts of the text where important notes needful for a correct and reliable use of the device can be found. User must pay attention to them! Failure to comply with instructions could cause the equipment to be set wrongly: hence a faulty and improper working of the device could be the consequence.
	This icon is meant to highlight the parts of the text where suggestions useful for making it easier to set the device and optimize performance and reliability can be found. Sometimes this symbol is followed by the word **EXAMPLE** when instructions for setting parameters are accompanied by examples to clarify the explanation.

Preliminary information

This guide is designed to provide the most complete information the operator needs to correctly and safely install and operate the **DRIVECOD rotary actuators RD6 model with EtherCAT interface**.

RD6 units are positioning devices which integrate into one system a brushless motor, a drive, a multiturn absolute encoder and a position controller. They are not equipped with gears. The 70 mm (2.76") size square flange and the 14 mm (0.55") shaft are designed to be coupled with planetary gearboxes available in the market. Thus the units can be easily integrated into custom applications to meet specific torque and speed requirements. RD6 are designed to drive positioning systems, change-over applications and linear guides. Typical uses are packaging lines, food processing and pharmaceutical industries, wood & metalworking machinery, paper machinery, material handling equipment, bending machines, filling and bottling plants, printing machines, mold changers, mobile stops, tool changers, spindle positioning devices, among others.

RD6 rotary actuators can be equipped with the following interfaces:

- RD6-x-xxx-**CB**-... = CANopen DS301 interface;
- RD6-x-xxx-**EC**-... = EtherCAT interface;
- RD6-x-xxx-**EP**-... = EtherNet/ IP interface;
- RD6-x-xxx-**MB**-... = Modbus RTU (RS-485) interface;
- RD6-x-xxx-**PB**-... = Profibus-DP interface;
- RD6-x-xxx-**PL**-... = POWERLINK interface.

The present manual is specifically designed to describe the EtherCAT interface model. For information on the actuators designed for the integration into other fieldbus/Ethernet networks, please refer to the specific documentation.

In the Modbus version the configuration of the DRIVECOD unit can be done through a software expressly developed and released by Lika Electronic in order to allow an easy set up of the device. The program is supplied for free and can be installed in any PC fitted with a Windows operating system (Windows XP or later). It allows the operator to set the working parameters of the device; control manually some movements and functions; and monitor whether the device is running properly. In the EtherCAT version configuration can be done using the same program through a **service RS-232 serial interface, in compliance with Modbus protocol**.

To make it easier to read the text, this guide can be divided into two main sections.
In the first section general information concerning the safety, the mechanical installation and the electrical connection as well as tips for setting up and running properly and efficiently the unit are provided.
In the second section, entitled **EtherCAT Interface**, both general and specific information is given on the EtherCAT interface. In this section the interface features and the objects implemented in the unit are fully described.
In the third section, entitled **Modbus Interface**, both general and specific information is given on the Modbus interface. As previously stated, EtherCAT version is equipped with a service RS-232 serial interface, in compliance with Modbus protocol. Using a software expressly developed and released by

Lika Electronic for free it allows the operator to configure the ROTADRIVE unit before installation in the EtherCAT network. In the **Modbus Interface** section the interface features and the registers implemented in the unit are fully described.

Glossary of EtherCAT terms

EtherCAT, like many other networking systems, has a set of unique terminology. Table below contains a few of the technical terms used in this guide to describe the EtherCAT interface. They are listed in alphabetical order.

Acknowledge telegram (AT)	Telegram, in which each Slave inserts its data.
Actual value	Value of a variable at a given instant.
Algorithm	Completely determined finite sequence of operations by which the values of the output data can be calculated from the values of the input data.
Application	Function or data structure for which data is consumed or produced. Software functional element specific to the solution of a problem in industrial-process measurement and control.
Application class	Configuration of a Drive Object with a set of functional objects and supported by standard telegrams.
Application mode	Type of application that can be requested from a PDS.
Application objects	Multiple object classes that manage and provide a run time exchange of messages across the network and within the network device.
Application process	part of a distributed application on a network, which is located on one device and unambiguously addressed.
Application relationship	cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation. This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities.
Attribute	Description of an externally visible characteristic or feature of an object, property or characteristic of an entity. The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behaviour of an object. Attributes are divided into class attributes and instance attributes.
Axis	Logical element inside an automation system (e.g. a motion control system) that represents some form of movement.
Basic Slave	Slave device that supports only physical addressing of data.
Behaviour	Indication of how an object responds to particular events.
Bit	Unit of information consisting of a 1 or a 0. This is the smallest data unit that can be transmitted.

CANopen	Application layer protocol as defined in EN 50325-4.
Channel	Representation of a single physical or logical management object of a Slave to control conveyance of data.
CIP™	Common Industrial Protocol (see IEC 61158 Type 2, IEC 61784-1 and IEC 61784-2 CPF2).
Class	Description of a set of objects that share the same attributes, operations, methods, relationships, and semantics.
Client	Object which uses the services of another (Server) object to perform a task. Initiator of a message to which a Server reacts.
Clock synchronization	Representation of a sequence of interactions to synchronize the clocks of all time receivers by a time Master.
Commands	Set of commands from the application control program to the PDS to control the behaviour of the PDS or functional elements of the PDS.
Communication cycle	Accumulation of all telegrams between two Master synchronization telegrams.
Communication object	Component that manages and provides a run time exchange of messages across the network.
Connection	Logical binding between two application objects within the same or different devices.
Consume	Act of receiving data from a provider.
Consumer	Node or sink receiving data from a provider.
Control	Purposeful action on or in a process to meet specified objectives.
Control device	Physical unit that contains – in a module/subassembly or device – an application program to control the PDS.
Control unit	Control device.
Control word	Two adjacent bytes inside the Master data telegram containing commands for the addressed drive.
Controller	Controlling device which is associated with one or more drives (axes) a host for the overall automation.
Conveyance path	Unidirectional flow of APDUs across an application relationship.
Cycle time	Time span between two consecutive cyclically recurring events.
Cyclic	Events which repeat in a regular and repetitive manner.
Cyclic data	Part of the telegram which does not change its meaning during cyclic operation of the interface. High priority real-time data that is transferred by a CIP Motion connection on a periodic basis.
Data	Generic term used to refer to any information carried over a

	fieldbus.
Data consistency	Means for coherent transmission and access of the input-or output-data object between and within Client and Server.
Data exchange	Demand dependent; non cyclic transmission (service channel).
Data type	Relation between values and encoding for data of that type according to the definitions of IEC 61131-3. Set of values together with a set of permitted operations.
Data type object	Entry in the object dictionary indicating a data type.
Default gateway	Device with at least two interfaces in two different IP subnets acting as router for a subnet.
Device	Field device. Networked independent physical entity of an industrial automation system capable of performing specified functions in a particular context and delimited by its interfaces. Entity that performs control, actuating and/or sensing functions and interfaces to other such entities within an automation system. Physical entity connected to the fieldbus composed of at least one communication element (the network element) and which may have a control element and/or a final element (transducer, actuator, etc.).
Device profile	Collection of device dependent information and functionality providing consistency between similar devices of the same device type. Representation of a device in terms of its parameters and behaviour according to a device model that describes the device's data and behaviour as viewed through a network, independent from any network technology.
Diagnosis information	All data available at the Server for maintenance purposes.
Distributed clocks	Method to synchronize Slaves and maintain a global time base.
DL	Data-link-layer.
DLPDU	Data-link-protocol-data-unit.
Drive Object	Functional element of a Drive Unit.
Drive Unit	Logical device which comprises all functional elements related to one central processing unit.
Error	Discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition.
Error class	General grouping for related error definitions and corresponding error codes.
Error code	Identification of a specific type of error within an error class.
EtherCAt State Machine	EtherCAT Slave is a state machine; communication and operating characteristics depend on the current state of the

	device.
Event	Instance of a change of conditions.
Event data	Medium priority real-time data that is transferred by a CIP Motion connection only after a specified event occurs.
Feed forward	Command value used to compensate the lag in the control loop.
Feedback variable	Variable which represents a controlled variable and which is returned to a comparing element.
Fieldbus memory management unit	Function that establishes one or several correspondences between logical addresses and physical memory.
Fieldbus memory management unit entity	Single element of the fieldbus memory management unit: one correspondence between a coherent logical address space and a coherent physical memory location.
Frame	Denigrated synonym for DLPDU.
FreeRun	Asynchronous communication mode.
Full Slave	Slave device that supports both physical and logical addressing of data.
Functional element	Entity of software or software combined with hardware, capable of accomplishing a specified function of a device.
HMI	Human Machine Interface.
Host	Device that covers the automation functionality of an automation device.
I/O data	Input data and output data that would typically need to be updated on a regular basis (e.g. periodic change of state), such as commands, set-points, status and actual values.
Identification number (IDN)	Designation of operating data under which a data block is preserved with its attribute, name, unit, minimum and maximum input values, and the data.
Index	Address of an object within an application process.
Input data	Data transferred from an external source into a device, resource or functional element.
Interface	Shared boundary between two entities defined by functional characteristics, signal characteristics, or other characteristics as appropriate.
Little endian	Data representation of multi-octet fields where the least significant octet is transmitted first.
Logical power drive system	Model which includes PDS and communication network accessible through the generic PDS interface.
Mapping	Correspondence between two objects in that way that one object is part of the other object.
Mapping parameters	Set of values defining the correspondence between application objects and process data objects.

Master	Device that controls the data transfer on the network and initiates the media access of the Slaves by sending messages and that constitutes the interface to the control system. Node, which assigns the other nodes the right to transmit.
Master data telegram (MDT)	Telegram, in which the Master inserts its data.
Medium	Cable, optical fibre or other means by which communication signals are transmitted between two or more points.
Message	Ordered series of octets intended to convey information. Normally used to convey information between peers at the application layer.
Model	Mathematical or physical representation of a system or a process, based with sufficient precision upon known laws, identification or specified suppositions.
Motion	Any aspect of the dynamics of an axis.
Motion Axis Object	Object that defines the attributes, services, and behaviour of a motion device based axis (or PDS) according to the CIP Motion specification, including Communications, Device Control, and Basic Drive FE elements as defined in IEC 61800-7.
Network	Set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways.
Node	Single DL-entity as it appears on one local link. End-point of a link in a network or a point at which two or more links meet [derived from IEC 61158-2].
Object	Abstract representation of a particular component within a device. An object can be: 1. an abstract representation of the capabilities of a device. Objects can be composed of any or all of the following components: ○ data (information which changes with time); ○ configuration (parameters for behaviour); ○ methods (things that can be done using data and configuration); 2. a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behaviour.
Object dictionary	Data structure addressed by Index and Sub-index that contains description of data type objects, communication objects and application objects. List of objects with unique 16-bit index and 8-bit sub-index as defined in EN 50325-4.
Operating cycle	Period of the control loop within the drive or the control unit.

Operating mode	Characterization of the way and the extent to which the human operator intervenes in the control equipment.
Output data	Data originating in a device, resource or functional element and transferred from them to external systems.
P-Device	Field device and the host for the Drive Objects.
Parameter	Data element that represents device information that can be read from or written to a device, for example through the network or a local HMI.
PDO	Process Data Object.
PDS	Power Drive System.
Process data	Collection of application objects designated to be transferred cyclically or acyclically for the purpose of measurement and control.
Process Data Object (PDO)	Communication object with real-time capability. Structure described by mapping parameters containing one or several process data entities.
Producer	Node or source sending data to one or many consumers.
Profile	Representation of a PDS interface in terms of its parameters, parameter assemblies and behaviour according to a communication profile and a device profile.
Protocol	Convention about the data formats, time sequences, and error correction in the data exchange of communication systems.
Reference variable	Input variable to a comparing element in a controlling system which sets the desired value of the controlled variable and is deducted from the command variable.
Resource	Processing or information capability.
Segment	Collection of one real Master with one or more Slaves.
Server	Object which provides services to another (Client) object.
Service	Operation or function than an object and/or object class performs upon request from another object and/or object class.
Service data	Lower priority real-time data associated with a service message from the controller that is transferred by a CIP Motion connection on a periodic basis.
Set-point	Value or variable used as output data of the application control program to control the PDS.
Slave	DL-entity accessing the medium only after being initiated by the preceding Slave or the Master. Node, which is assigned the right to transmit by the Master.
Standard telegram	Set of input data and output data for an application mode.
Status	Set of information from the PDS to the application control program reflecting the state or mode of the PDS or a functional element of the PDS.

Status word	Two adjacent bytes inside the drive telegram containing status information.
Subindex	Sub-address of an object within the object dictionary.
Supervisor	Engineering device which manages provisions of configuration data (parameter sets) and collections of diagnosis data from P-Devices and/or controllers.
Switch	MAC bridge as defined in IEEE 802.1D.
Sync Manager	Sync Manager has the task of synchronizing data transfer between Master and Slave and prevents the same memory area from being written by different events. Collection of control elements to coordinate access to concurrently used objects.
Sync manager channel	Single control elements to coordinate access to concurrently used objects.
Synchronised	Condition where the local clock value on the drive is locked onto the Master clock of the distributed System Time.
Synchronous with DC SYNC0	In this operating mode data is sampled and then copied into Sync Manager buffer simultaneously at SYNC0 event generated by the ESC capture/compare unit.
Synchronous with SM3	In this mode data is sampled and then copied into Sync Manager buffer as soon as previous data was read from the Master (SM event); in this way new sampled data is synchronous with Master readings.
System Time	Absolute time value as defined in the CIP Sync specification in the context of a distributed time system where all devices have a local clock that is synchronised with a common Master clock.
Telegram	Message.
Time stamp	System Time stamp value associated with the CIP Motion connection data that conveys the absolute time when the associated data was captured, or that can also be used to determine when the associated data shall be applied.
Topology	Physical network architecture with respect to the connection between the stations of the communication system.
Type	Hardware or software element which specifies the common attributes shared by all instances of the type.
Use case	Class specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.
Variable	Software entity that may take different values, one at a time.

Glossary of MODBUS terms

MODBUS, like many other networking systems, has a set of unique terminology. Table below contains a few of the technical terms used in this guide to describe the MODBUS interface. They are listed in alphabetical order.

Address field	It contains the Slave address.
Application Process	The Application Process is the task on the Application Layer.
Application protocol	MODBUS is an application protocol or messaging structure that defines rules for organizing and interpreting data independent of the data transmission medium.
ASCII transmission mode	When devices are setup to communicate on a MODBUS serial line using ASCII (American Standard Code for Information Interchange) mode, each 8-bit byte in a message is sent as two ASCII characters. This mode is used when the physical communication link or the capabilities of the device does not allow the conformance with RTU mode requirements regarding timers management.
Bus	A bus is a communication medium connecting several nodes. Data can be transferred via serial or parallel circuits, that is, via electrical conductors or fibre optic.
Client	A Client is any network device that sends data requests to servers. MODBUS follows the Client/Server model. MODBUS Masters are referred to as Clients, while MODBUS Slaves are Servers.
Cyclic Redundancy Check (CRC)	Error-checking technique in which the frame recipient calculates a remainder by dividing frame contents by a prime binary divisor and compares the calculated remainder to a value stored in the frame by the sending node.
Data encoding	MODBUS uses a 'big-Endian' representation for addresses and data items. This means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first.
Exception code	Code to be returned by Slaves in the event of problems. All exceptions are signalled by adding 0x80 to the function code of the request.
Exception response	MODBUS operates according to the common client/server (Master/Slave) model: the Client (Master) sends a request telegram (service request) to the Server (Slave), and the Server replies with a response telegram. If the Server cannot process a request, it will instead return a error function code (exception response) that is the original function code plus 80H (i.e. with its most significant bit set to 1).
Function code	MODBUS is a request/reply protocol and offers services

	specified by function codes. The function code is sent from a Client to the Server and indicates which kind of action the Server must perform. MODBUS function codes are elements of MODBUS request/reply PDUs. The function code field of a MODBUS data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (the range 128 – 255 is reserved and used for exception responses). Function code "0" is not valid. Lika actuators only implement public function codes.
Holding register	In the MODBUS data model, a Holding register is the output data. A Holding register has a 16-bit quantity, is alterable by an application program, and allows either read-write or read-only access.
IEEE 1588	This standard defines a protocol enabling synchronisation of clocks in distributed networked devices (e.g. connected via Ethernet).
Input register	In the MODBUS data model, an Input register is the input data. An Input register has a 16-bit quantity, is provided by an I/O system, and allows read-only access.
LRC Checking	In ASCII mode, messages include an error-checking field that is based on a Longitudinal Redundancy Checking (LRC) calculation that is performed on the message contents, exclusive of the beginning 'colon' and terminating CRLF pair characters. It is applied regardless of any parity checking method used for the individual characters of the message.
Master	A Master is any network device that sends data requests to Slaves.
Message	The MODBUS messaging service provides a Client/Server communication between devices connected on the network. The Client / Server model is based on four types of messages: • MODBUS Request • MODBUS Confirmation • MODBUS Indication • MODBUS Response The MODBUS messaging services are used for information exchange.
MODBUS Confirmation	A MODBUS Confirmation is the Response Message received on the Client side.
MODBUS Indication	A MODBUS Indication is the Request message received on the Server side.
MODBUS Request	A MODBUS Request is the message sent on the network by the Client to initiate a transaction.
MODBUS Response	A MODBUS Response is the Response message sent by the Server.

Network	Network is a group of computers on a single physical network segment.
PDU	The Protocol Data Unit (PDU) is the MODBUS function code and data field. It is packed together with the Address Field and the CRC (or LRC) to form the Modbus Serial Line PDU. The MODBUS protocol defines three PDUs. They are: • MODBUS Request PDU, mb_req_pdu • MODBUS Response PDU, mb_rsp_pdu • MODBUS Exception Response PDU, mb_excep_rsp_pdu
Read Holding Registers (03, 0003hex)	This function code is used to READ the contents of a contiguous block of holding registers in a remote device; in other words, it allows to read the values set ina group of work parameters placed in order.
Read Input Register (04, 0004hex)	This function code is used to READ from 1 to 125 contiguous input registers in a remote device; in other words, it allows to read some result values and state / alarm messages in a remote device.
Register	MODBUS functions operate on memory registers to configure, monitor, and control device I/O.
RTU transmission mode	Remote Terminal Unit. When devices communicate on a MODBUS serial line using the RTU mode, each 8–bit byte in a message contains two 4–bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII mode for the same baud rate. Each message must be transmitted in a continuous stream of characters.
Server	A Server is any program that awaits data requests to be sent to it. Servers do no initiate contacts with Clients, but only respond to them. MODBUS follows the Client/Server model. MODBUS Masters are referred to as clients, while MODBUS Slaves are servers.
Service request	It is the MODBUS Request, i.e. the message sent on the network by the Client to initiate a transaction.
Slave	A Slave is any program that awaits data requests to be sent to it. Slaves do no initiate contacts with Masters, but only respond to them.
Transmission rate	Data transfer rate (in bps).
Write Multiple Registers (16, 0010hex)	This function code is used to WRITE a block of contiguous registers (1 to 123 registers) in a remote device.
Write Single Register (06, 0006hex)	This function code is used to WRITE a single holding register in a remote device.

1 Safety summary

1.1 Safety

- Always adhere to the professional safety and accident prevention regulations applicable to your country during device installation and operation;
- installation and maintenance operations have to be carried out by qualified personnel only, with power supply disconnected and stationary mechanical parts;
- device must be used only for the purpose appropriate to its design: use for purposes other than those for which it has been designed could result in serious personal and/or the environment damage;
- high current, voltage and moving mechanical parts can cause serious or fatal injury;
- warning ! Do not use in explosive or flammable areas;
- failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the equipment;
- Lika Electronic assumes no liability for the customer's failure to comply with these requirements.

1.2 Electrical safety

- Turn OFF power supply before connecting the device;
- connect according to explanation in the "Electrical connections" section;
- a safety push-button for emergency power off must be installed to shut off the motor power supply in case of emergency situations;
- 
in compliance with 2014/30/EU norm on electromagnetic compatibility, following precautions must be taken:
  - before handling and installing the equipment, discharge electrical charge from your body and tools which may come in touch with the device;
  - power supply must be stabilized without noise; install EMC filters on device power supply if needed;
  - always use shielded cables (twisted pair cables whenever possible);
  - avoid cables runs longer than necessary;

- avoid running the signal cable near high voltage power cables;
- mount the device as far as possible from any capacitive or inductive noise source; shield the device from noise source if needed;
- to guarantee a correct working of the device, avoid using strong magnets on or near by the unit;
- minimize noise by connecting the shield and/or the connector housing and/or the frame to ground. Make sure that ground is not affected by noise. The connection point to ground can be situated both on the device side and on user's side. The best solution to minimize the interference must be carried out by the user.

1.3 Mechanical safety

- Install the device following strictly the information in the "Mechanical installation" section;
- mechanical installation has to be carried out with stationary mechanical parts;
- do not disassemble the unit;
- do not tool the unit or its shaft;
- delicate electronic equipment: handle with care; do not subject the device and the shaft to knocks or shocks;
- respect the environmental characteristics of the product;
- the actuator can be mounted directly on the drive shaft or coupled with planetary gearboxes. An adapting flange can be further interposed.

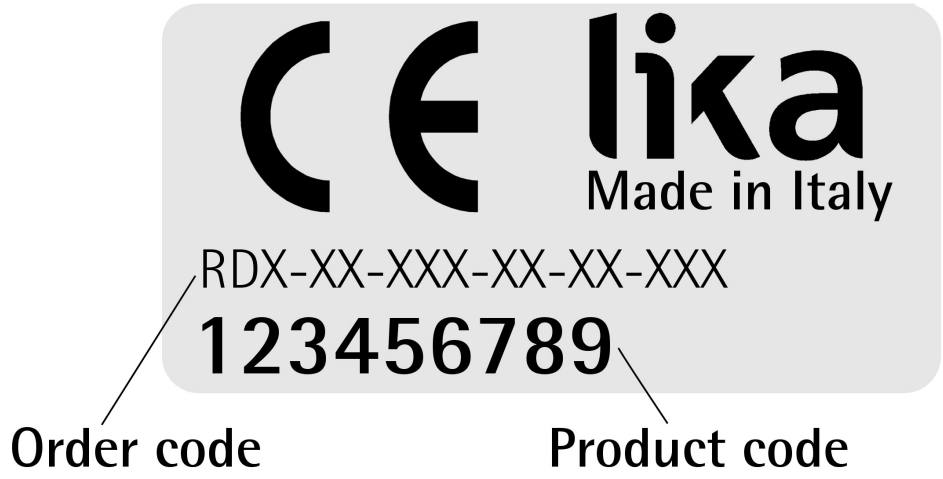
WARNING
The unit has been adjusted by performing a full-load mechanical running test; thence default values which has been set refer to a device running in such condition. Furthermore they are intended to ensure a standard and safe operation which not necessarily results in a smooth running and an optimum performance. Thus to suit the specific application requirements it may be advisable and even necessary to enter new parameters instead of the factory default settings; in particular it may be necessary to change velocity, acceleration, deceleration and gain values.

WARNING
The counter-electromotive force (back EMF) generated by the motor in case the shaft is forced to spin due to a manual external force can cause irreparable damages to the internal circuitry.

WARNING
Please evaluate attentively the characteristics of the 24V power supply pack as the counter-electromotive force (back EMF) generated by the motor flows back and directly discharge through the capacitor module of the 24V power supply pack.

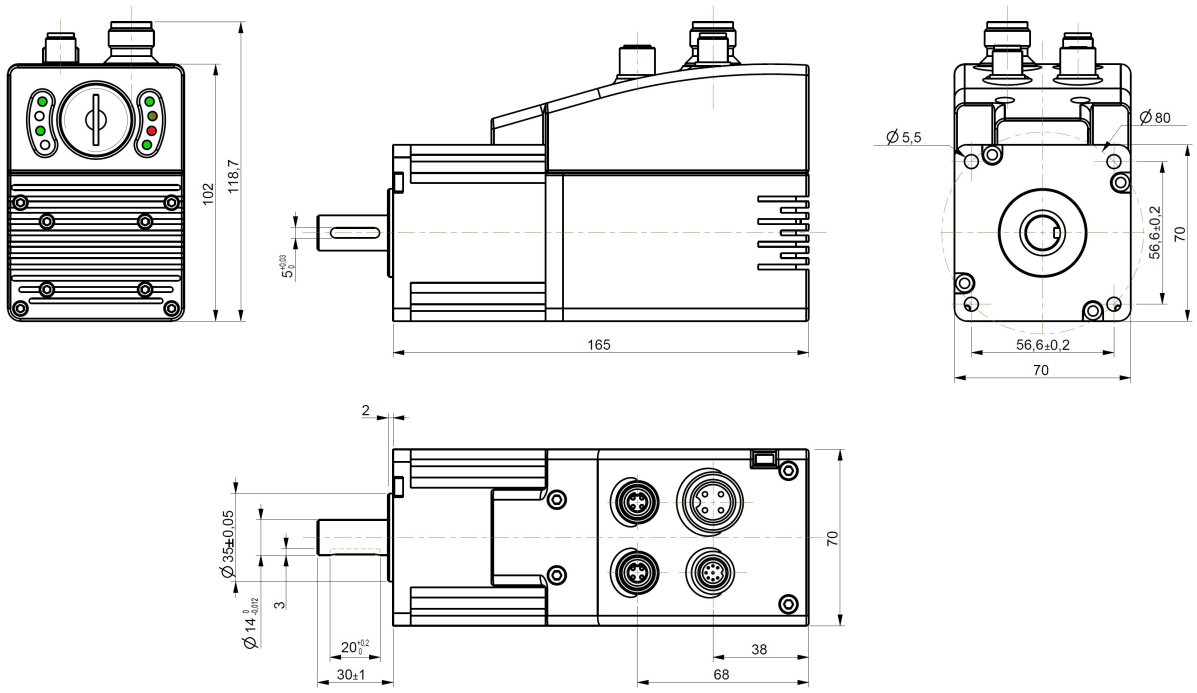
2 Identification

Device can be identified through the **order code** and the **serial number** printed on the label applied to its body. Information is listed in the delivery document too. Please always quote the order code and the serial number when reaching Lika Electronic for purchasing spare parts or needing assistance. For any information on the technical characteristics of the product refer to the technical catalogue.



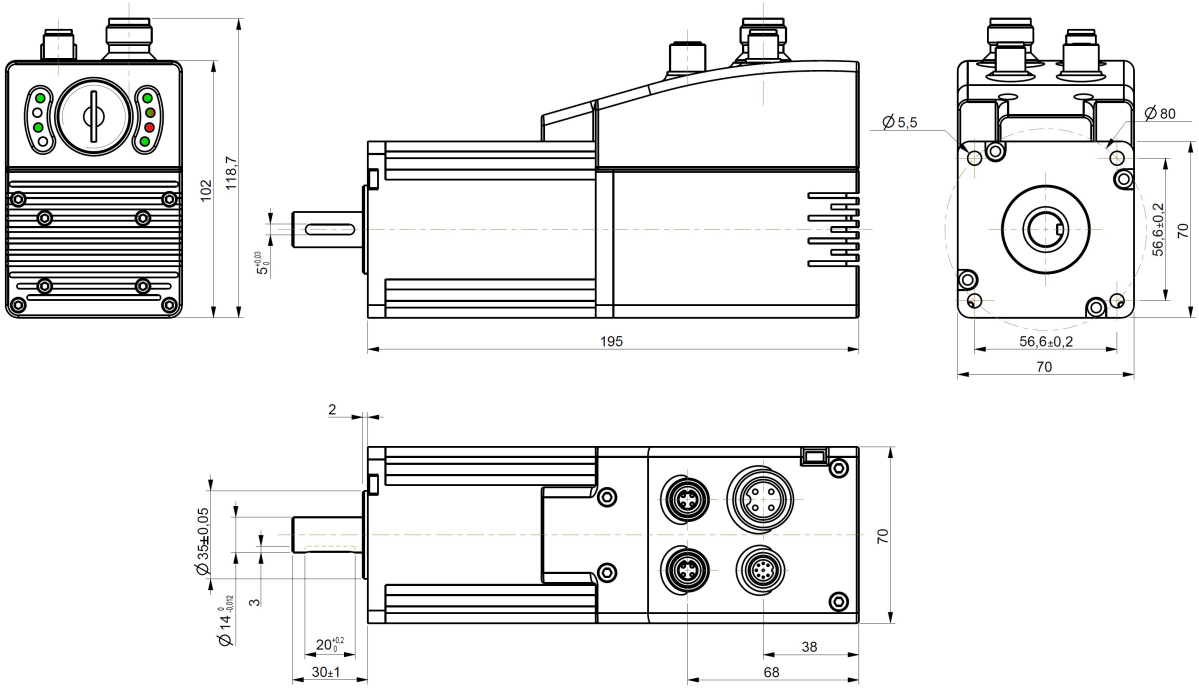
3 Mechanical installation

WARNING
Installation and maintenance operations have to be carried out by qualified personnel only, with power supply disconnected. Motor and shaft must be in stop.



(values are expressed in mm)

Figure 1 - RD6-P8-157-... unit – Overall dimensions



(values are expressed in mm)

Figure 2 - RD6-P8-250-... unit – Overall dimensions

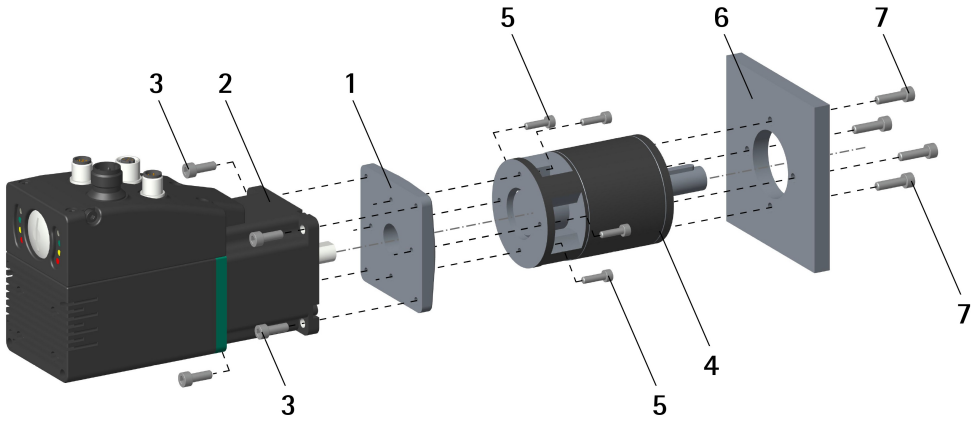
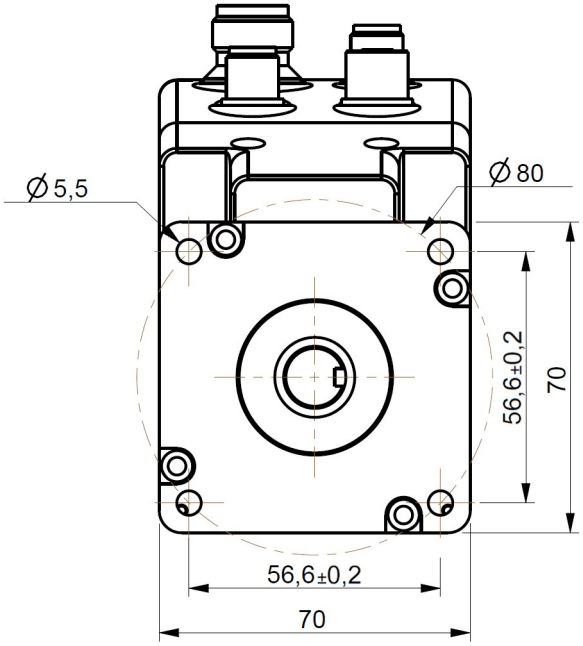


Figure 3 - Installation example of an RD6 unit

To install the DRIVECOD unit properly please read carefully and follow the instructions below; note anyway that the unit can be installed in several manners and according to the specific user's application.

- If required, mount an adapting flange **1**; the adapting flange **1** can be either fixed to the actuator's flange **2** first and then to the planetary gearbox **4**; or on the contrary it can be fixed to the planetary gearbox **4** first and then to the actuator's flange **2**, according to the mounting configuration;



  - use M5 type screws **3** to fix the adapting flange **1** to the actuator's flange **2**;
  - use the screws **5** to fix the planetary gearbox **4** to the adapting flange **1**;
  - couple and fasten the actuator and the planetary gearbox **4** together using the screws **3** or **5** according to the mounting configuration; properly secure the shaft of the actuator and the shaft of the planetary gearbox **4**;
- it could be advisable to install a coupling between the actuator and the planetary gearbox **4**;
- mount the shaft of the planetary gearbox **4** on the drive's shaft and properly secure them together; then fasten the planetary gearbox **4** to the flange or the mounting support **6** by means of the screws **7**.

WARNING

Never force manually the rotation of the shaft not to cause permanent damages! The counter-electromotive force (back EMF) generated by the motor in case the shaft is forced to spin due to a manual external force may cause irreparable damages to the internal circuitry.

4 Electrical connections

WARNING

When you send the **Start**, **Jog +** or **Jog -** commands, the unit and the shaft start moving! Before operating please make sure that there are no risks of personal injury and mechanical damages.
Each **Start** routine has to be checked carefully in advance!
Never force manually the rotation of the shaft not to cause permanent damages!

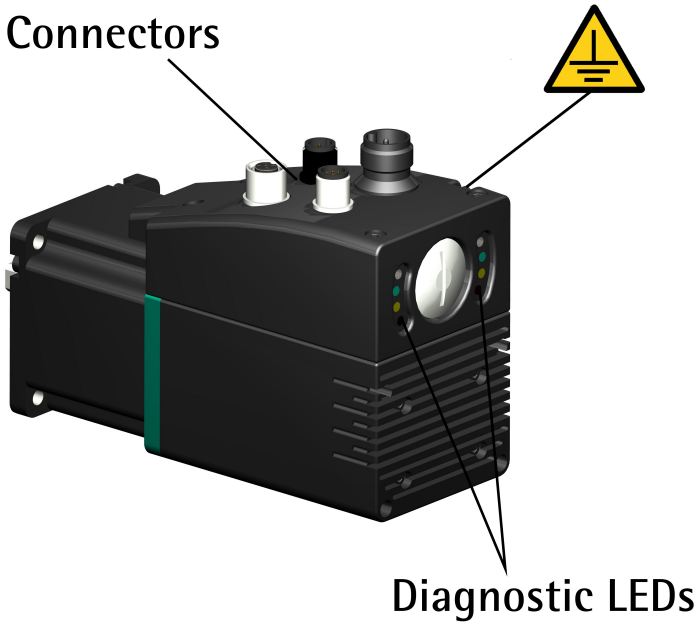
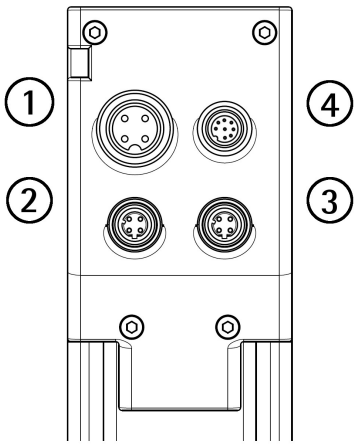


Figure 4: Connectors and diagnostic LEDs

4.1 Ground connection (Figure 5)

To minimize noise connect properly the frame to ground; we suggest using the ground screw provided in the frame (see the Figure above). Connect properly the cable shield to ground on user's side. Lika EC- pre-assembled cables are fitted with shield connection to the connector ring nut in order to allow grounding through the body of the device. Lika E- connectors have a plastic gland, thus grounding is not possible. If metal connectors are used, connect the cable shield properly as recommended by the manufacturer. See also the note in the next paragraph. Anyway make sure that ground is not affected by noise. It is recommended to provide the ground connection as close as possible to the device.

4.2 Connectors (Figure 4 and Figure 5)



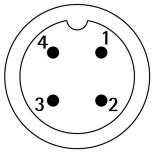
1	M18 4-pin male connector power supply
2	M12 4-pin female connector PORT OUT (ECAT OUT)
3	M12 4-pin female connector PORT IN (ECAT IN)
4	M12 8-pin male connector, service serial port Modbus RS-232

Figure 5: Connectors

4.2.1 Power supply connector

Power supply
M18 4-pin male connector

(frontal side)



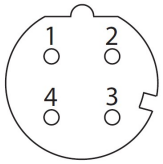
Pin	Description
1	**motor +24Vdc ±10% power supply**
2	**controller +24Vdc ±10% power supply**
3	**motor 0Vdc supply voltage**
4	**controller 0Vdc supply voltage**

NOTE
Wire gauge of the mating connector cable: 1.50 mm^2 max. Please consider the consumption of both the motor and the controller to evaluate the better configuration, see the datasheet.

4.2.2 EtherCAT interface connectors (PORT IN and PORT OUT)

Two M12 4-pin female connectors with D coding are used for Ethernet connection through PORT IN (ECAT IN) and PORT OUT (ECAT OUT).

Interface
M12 4-pin connectors
D coding, female
(frontal side)



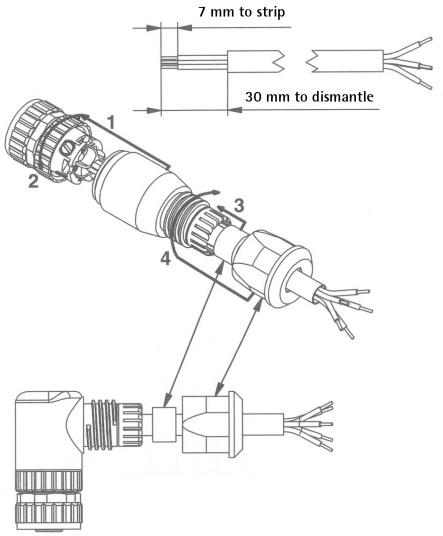
Pin	Description
1	**Tx Data +**
2	**Rx Data +**
3	**Tx Data -**
4	**Rx Data -**
Case	**Shielding** [1]

[1] Lika EC- pre-assembled cables only

The Ethernet interface supports 100 Mbit/s, fast Ethernet, full duplex operation.

PORT OUT **2** and PORT IN **3** M12 connectors have pin-out in compliance with the EtherCAT® standard. Therefore you can use standard EtherCAT cables commercially available.
Input (PORT IN **3**) and output (PORT OUT **2**) connectors are not interchangeable! PORT IN connector must be networked towards the EtherCAT Master.

NOTE
We suggest always connecting the cable shield to ground on user's side.
Lika EC- pre-assembled cables are fitted with shield connection to the connector ring nut in order to allow grounding through the body of the device. Lika E-connectors have a plastic gland, thus grounding is not possible (see Figure below). If metal connectors are used, connect the cable shield properly as recommended by the manufacturer.



4.2.3 Network configuration: topologies, cables, hubs, switches - Recommendations

Cables and connectors comply with the EtherCAT specifications. Cables are CAT-5 shielded cables.
Line, tree or star: EtherCAT supports almost any topology. The bus or line structure known from the fieldbuses thus also becomes available for Ethernet, without the quantity limitations implied by cascaded switches or hubs.
The Fast Ethernet physics (100BASE-TX) enables a cable length of 100 m (328 ft) between two devices. Since up to 65,535 devices can be connected, the size of the network is almost unlimited.
The Ethernet protocol according to IEEE 802.3 remains intact right up to the individual device; no sub-bus is required. In order to meet the requirements of a modular device like an electronic terminal block, the physical layer in the coupling device can be converted from twisted pair or optical fiber to LVDS (alternative Ethernet physical layer, standardized in [4,5]). A modular device can thus be extended very cost-efficiently. Subsequent conversion from the backplane physical layer LVDS to the 100BASE-TX physical layer is possible at any time – as usual with Ethernet.

For a complete list of the available cordsets and connection kits please refer to the product datasheet ("Accessories" list).

4.2.4 Addressing

It is not necessary to assign a physical address to the device because the addressing of the Slave is automatic at power-on during the initial scanning of the hardware configuration.
The field for addressing is 32-bit long, the Auto Increment Addressing is supported.

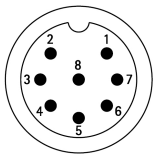
- Auto Increment Addressing = Position Addressing: 16 bits indicate the physical position of the Slave inside the network while 16 bits are scheduled for local memory addressing; when the Slave receives the frame then it increments the position address and the Slave receiving address 0 is the addressed device.

4.2.5 Line Termination

EtherCAT network needs no line termination because the line is terminated automatically; in fact every Slave is able to detect the presence of the downstream Slaves.

4.2.6 Modbus RS-232 service port

Modbus RS-232 service port
M12 8-pin connector
A coding, male
(frontal side)



Pin	Description
1	**n.c.**
2	**n.c.**
3	**n.c.**
4	**n.c.**
5	**n.c.**
6	**TD (RS-232)**
7	**RD (RS-232)**
8	**0Vdc (RS-232)**

The configuration parameters of the Modbus service serial port have fixed values so the user cannot change them.
They are:

RS-232 Modbus service serial port settings	**Default value**
Baud rate	9600
Byte size	8
Parity	Even
Stop bits	1

The MODBUS address is 1. It cannot be changed. See the "8.2 "Serial configuration" page" section on page 108.

For any further information on configuring and using the RS-232 service serial port refer to the "Modbus® interface" section on page 106.

4.3 Diagnostic LEDs (Figure 4 and Figure 6)

Eight LEDs located in the back of the actuator's enclosure are meant to show visually the operating or fault status of both the EtherCAT and MODBUS interfaces and the device. The meaning of each LED is explained in the following tables.

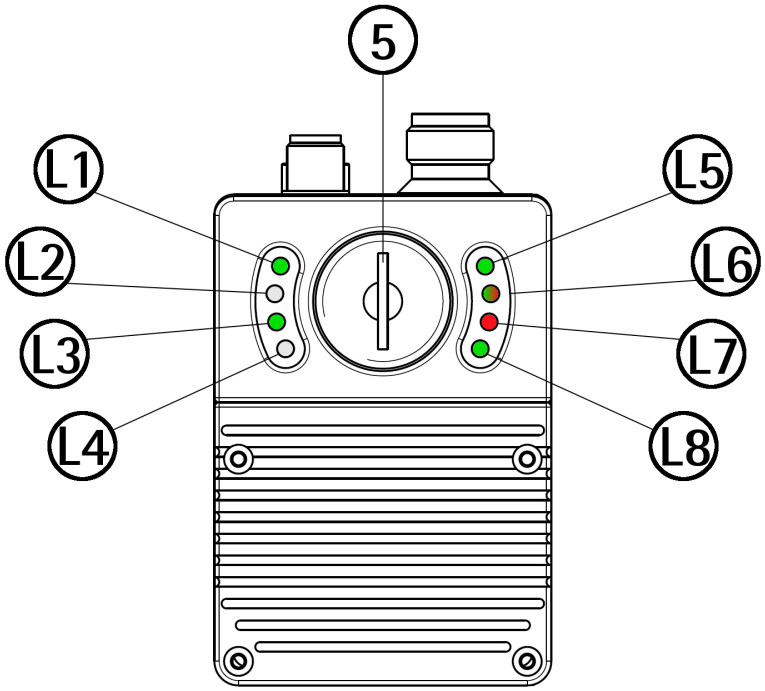


Figure 6: Diagnostic LEDs

5	Screw plug not used, do not open

L1	Link / Activity in PORT IN	**L5**	Controller power supply information
L2	Not used	**L6**	Fieldbus interface status information
L3	Link / Activity in PORT OUT	**L7**	Active errors / faults information
L4	Not used	**L8**	Motor enabling information

NOTE
Please note that the LEDs have different meanings depending on the active interface.

The LEDs operation is according to the EtherCAT specifications, see ETG1300_S_R_V1i1i0_IndicatorLabelingSpecification.pdf.

LED states	Definition
ON	The indicator shall be constantly ON.
OFF	The indicator shall be constantly OFF.
Flickering	The indicator shall turn ON and OFF iso-phase with a frequency of 10 Hz: ON for 50 ms and OFF for 50 ms.
Blinking	The indicator shall turn ON and OFF iso-phase with a frequency of 2.5 Hz: ON for 200 ms followed by OFF for 200 ms.
Single flash	The indicator shall show one short flash (200 ms) followed by a long OFF phase (1000 ms).
Double flash	The indicator shall show a sequence of two short flashes (200 ms), separated by an OFF phase (200 ms), and followed by a long OFF phase (1000 ms).

LED L1 GREEN	Description
It shows the state of the physical link and the activity in PORT IN (connector 3)	
OFF	There is neither network link nor activity
ON	The network link has been detected, but there is no activity
FLASHING	Activity (data exchange)

LED L3 GREEN	Description
It shows the state of the physical link and the activity in PORT OUT (connector 2)	
OFF	There is neither network link nor activity
ON	The network link has been detected, but there is no activity
FLASHING	Activity (data exchange)

LED L5 GREEN	Description
It shows whether the power supply of the controller is switched on	
ON	It indicates that the power supply of the controller is turned on
OFF	It indicates that the power supply of the controller is turned off

LED L6 GREEN / RED	Description
EtherCAT RUN LED, it shows the state of the EtherCAT communication	
OFF	The actuator is in INIT state
Blinking GREEN	The actuator is in PRE-OPERATIONAL state

Single flash GREEN	The actuator is in SAFE-OPERATIONAL state
ON GREEN	The actuator is in OPERATIONAL state
Flickering GREEN	The actuator is in BOOT state. Please do not use, the unit could be irreparably damaged
ON RED	Fatal event. If RUN and ERROR LEDs turn red, this indicates a fatal event, forcing the bus interface to a physically passive state. If such a condition arises, please contact Lika Electronic After Sales Service.

LED L7 RED	**Description**
EtherCAT ERROR LED, it shows the EtherCAT communication errors	
OFF	No error
Blinking RED	Invalid configuration. State change received from the Master is not possible due to invalid register or object settings
Single flash RED	Unsolicited state change. Slave device application has changed the EtherCAT state autonomously
Double flash RED	Application watchdog timeout, Sync manager watchdog timeout, communication with Master cut off
ON RED	Fatal event. If RUN and ERROR LEDs turn red, this indicates a fatal event, forcing the bus interface to a physically passive state. If such a condition arises, please contact Lika Electronic After Sale Service.
Flickering RED	Booting error detected, e.g. due to firmware download failure

LED L8 GREEN	**Description**
It shows the enabling state of the motor	
ON	It indicates that the motor is enabled (control loop activated)
OFF	It indicates that the motor is disabled (control loop deactivated)

LED L5 GREEN	**Description**
It shows whether the power supply of the controller is switched on	
ON	It indicates that the power supply of the controller is turned on
OFF	It indicates that the power supply of the controller is turned off

LED L8 GREEN	Description
It shows the enabling state of the motor	
ON	It indicates that the motor is enabled (control loop activated)
OFF	It indicates that the motor is disabled (control loop deactivated)

During initialisation, the system checks the controller diagnostic LED L5 for proper operation; therefore it blinks for a while.

4.4 Screw plug for internal access (Figure 4 and Figure 6)

WARNING
Do not remove the screw plug **5**, there are no serviceable parts inside.

5 Quick reference

The following instructions are given to allow the operator to set up the device for standard operation in a quick and safe mode.

- Mechanically install the device;
- execute the electrical connections;
- switch on the +24Vdc power supply (in both the motor and the controller);
- check the operating condition shown through the LEDs;
- to resume the normal work condition reset the active emergency: switch high ("=1") the **Emergency** bit 7 of the **Control Word** (see on page 90 -EtherCAT interface; see on page 148 -MODBUS interface); reset the active alarms: switch high ("=1") the **Alarm reset** bit 3 of the **Control Word** (see on page 90 -EtherCAT interface; see on page 148 -MODBUS interface). Check the operating condition shown through the LEDs;
- set a proper value next to the **Distance per revolution** item (see on page 97 -EtherCAT interface; see on page 141 -MODBUS interface);
- set a proper value next to the **Jog speed** item (see on page 101 -EtherCAT interface; see on page 145 - MODBUS interface);
- set a proper value next to the **Work speed** item (see on page 101 -EtherCAT interface; see on page 145 -MODBUS interface);
- if required, set a proper value next to the **Preset** item (see on page 102 -EtherCAT interface; see on page 146 -MODBUS interface);
- set the limit switch values next to the **Max delta pos** / **Positive delta** and **Max delta neg** / **Negative delta** items (see on page 99 ff -EtherCAT interface; see on page 143 ff -MODBUS interface);
- set the commanded position next to the **Target position** item (see on page 94 -EtherCAT interface; see on page 152 -MODBUS interface);
- save the new setting values (**Save parameters** command; see on page 92 -EtherCAT interface; see on page 150 -MODBUS interface).

Use the **Jog+**, **Jog-**, **Start** and **Stop** commands in the **Control Word** (see on page 90 -EtherCAT interface; see on page 148 -MODBUS interface) to move the axis and reach the commanded position.

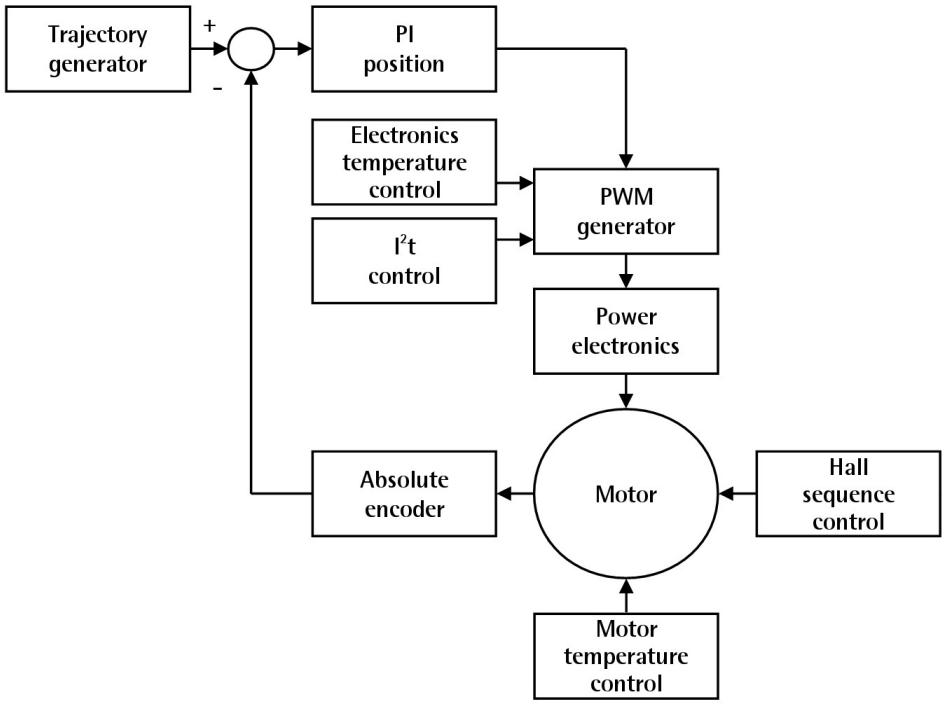
NOTE

The parameters **Distance per revolution**, **Jog speed**, **Work speed**, **Preset**, **Max delta pos** / **Positive delta** and **Max delta neg** / **Negative delta** are closely related, hence you have to be very attentive when you need to change the value in any of them. For any further information please refer to page 45.

6 Functions

6.1 Working principle

The following scheme is intended to show schematically the working principle of the system control logic.



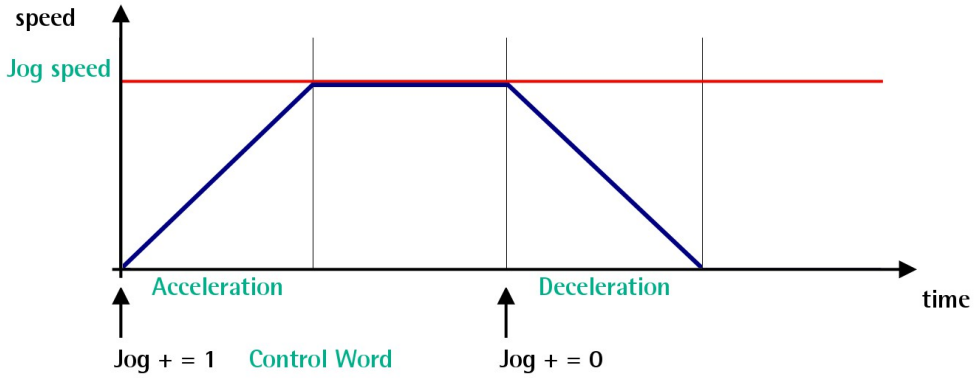
6.2 Movements: jog and positioning

Two kinds of movement are available in the DRIVECOD positioning unit, they are:

- Jog: speed control;
- Positioning: position and speed control.

Jog: speed control

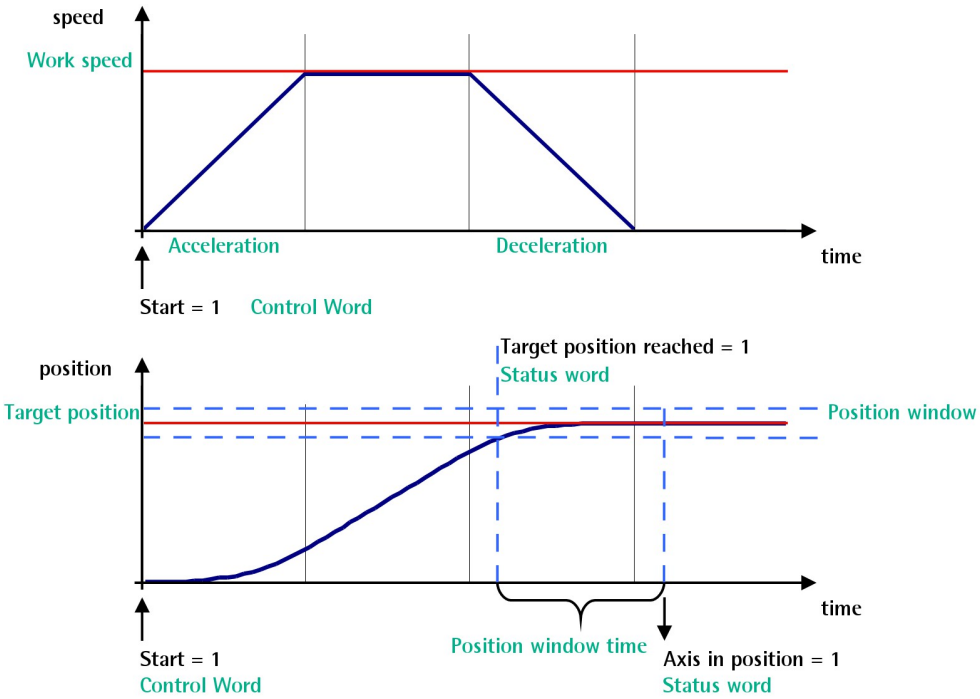
This kind of control is intended to generate a speed trajectory which allows the rotation speed of the DRIVECOD unit shaft to be equal to the value set in the **220E-00 Jog speed** / **Jog speed [0x0D]** parameter.



When the bit 0 **Jog +** in the **2200 Control Word** / **Control Word [0x2A]** is "=1", the motor accelerates toward the positive direction according to the value set next to the **220A-00 Acceleration** / **Acceleration [0x07]** item; if the available travel is long enough it reaches the speed set next to the **220E-00 Jog speed** / **Jog speed [0x0D]** item. As soon as the bit 0 **Jog +** in the **2200 Control Word** / **Control Word [0x2A]** goes low ("=0"), the motor decelerates according to the value set next to the**220B-00 Deceleration** / **Deceleration [0x08]** item until it stops.
Setting the bit 1 **Jog -** in the **2200 Control Word** / **Control Word [0x2A]** to "=1" causes the motor to run in the opposite direction (negative direction) respecting the work phases already described above.

Positioning: position and speed control

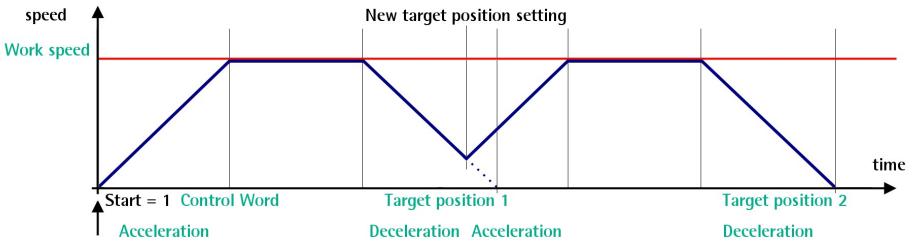
This kind of control is a point-to-point movement and the maximum reachable speed is equal to the value set in the 220F-00 Work speed / Work speed [0x0E] parameter; the set speed can be reached only if the available travel is long enough.



When the bit 6 **Start** in the2200 Control Word / Control Word [0x2A] is "=1", the motor starts moving and accelerates according to the value set next to the 220A-00 Acceleration / Acceleration [0x07] item in order to reach the target position as set next to the 2201-00 Target Position / Target position [0x2B-0x2C] item. If the available travel is long enough it reaches the speed set next to the 220F-00 Work speed / Work speed [0x0E] item. The movement direction can be either positive or negative according to the target position to reach. As soon as the axis is within the tolerance window limits set next to the 2205-00 Position tolerance / Position window [0x01] item, the bit 8 **Target position reached** in the 2202 Status Word / Status word [0x01] goes high ("=1"). When the position is within the tolerance window limits set next to the 2205-00 Position tolerance / Position window [0x01] item, after the delay set next to the 2206-00 Settling time / Position window time [0x02] item, the bit 0 **Axis in position** in the 2202 Status Word / Status word [0x01] goes high ("=1"). The motor decelerates according to the value set next to the

220B-00 Deceleration / Deceleration [0x08] item in order to reach the halt position according to the set target position.

NOTE
Position override function
It is possible to change the target position value even on the fly, while the device is still reaching a previously commanded target position and without sending a new **Start** command. To do this, just set a new target value in the 2201-00 Target Position / Target position [0x2B-0x2C] item.



6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta

The variables Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta are closely related, hence you have to be very attentive every time you need to change the value in any of them.
Should a new setting be necessary, please comply with the following procedure:

- set a proper value next to the Distance per revolution item (see on page 97 -EtherCAT interface; see on page 141 -MODBUS interface);
- set a proper value next to the Jog speed item (see on page 101 -EtherCAT interface; see on page 145 -MODBUS interface);
- set a proper value next to the Work speed item (see on page 101 -EtherCAT interface; see on page 145 -MODBUS interface);
- set a proper value next to the Preset item (see on page 93 -EtherCAT interface; see on page 146 -MODBUS interface);
- check the value next to the Max delta pos / Positive delta item (see on page 99 -EtherCAT interface; see on page 143 -MODBUS interface);
- check the value next to the Max delta neg / Negative delta item (see on page 100 -EtherCAT interface; see on page 144 -MODBUS interface);

- save the new values (**Save parameters** command, bit 9 in the 2200 Control Word / Control Word [0x2A] item, see on page 90 -EtherCAT interface; see on page 150 -MODBUS interface).

Each time you change the value in Distance per revolution then you must update the value in Preset in order to define the zero of the axis as the system reference has changed.
After having changed the parameter in the Preset item it is not necessary to set new values for the travel limits as the Preset function calculates them automatically and initializes again the positive and negative limits according to the values set in Max delta pos / Positive delta and Max delta neg / Negative delta.

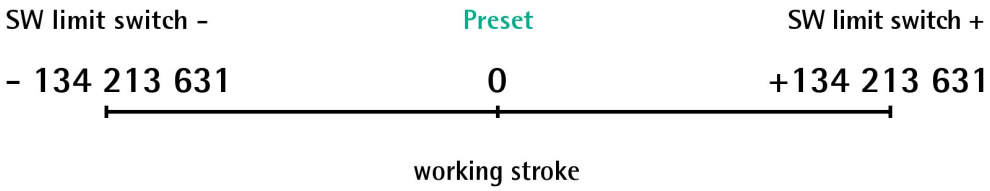
The number of revolutions managed by the system is 32,768 in negative direction and 32,768 in positive direction assuming the Preset value as a reference.
The value set next to the Max delta pos / Positive delta item plus the value set in the Preset parameter is the maximum forward travel (positive travel) starting from the preset (the value is expressed in pulses).
The value set next to the Max delta neg / Negative delta item subtracted from the value set in the Preset parameter is the maximum backward travel (negative travel) starting from the preset (the value is expressed in pulses).

WARNING
Please note that the parameters listed hereafter are closely related to the Distance per revolution parameter; hence when you change the value in Distance per revolution also the value in each of them necessarily changes. They are: Position tolerance / Position window, Max following error, Max delta pos / Positive delta, Max delta neg / Negative delta, Target position, Real position / Current position and Following error [pulse] / Position following error.

EXAMPLE 1
Default values:
Distance per revolution = 4096 steps per revolution
Max. Work speed: 2000 rpm
Preset = 0

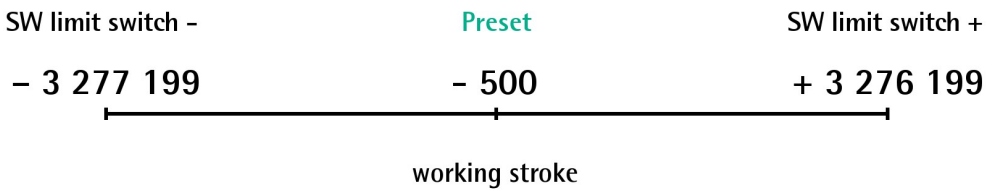
Max delta pos / Positive delta and Max delta neg / Negative delta max. values = 134 213 631 = (4,096 steps per revolution x 32,768 revolutions) - 1 – 4,096 steps (i.e. 1 revolution for safety reasons) when Preset = 0

Max. **SW limit switch +** = 0 + 134 213 631 = + 134 213 631 pulses (forward travel)
Max. **SW limit switch -** = 0 - 134 213 631 = - 134 213 631 pulses (backward travel)
Therefore, when **Preset** = 0, the working stroke of the axis will span the overall positive and negative limits range, that is max. **SW limit switch +** = + 134 213 631 and max. **SW limit switch -** = – 134 213 631.



EXAMPLE 2
DRIVECOD RD6 positioning unit is joined to a worm screw having 1 mm (0.039") pitch and you need to have a hundredth of a millimetre resolution.

Distance per revolution = 100 steps per revolution
Max. **Work speed** = 73 rpm (100 * 3000 / 4096 = 73)
Preset = -500 (ex. thickness of the tool)
Max. delta pos / **Positive delta** and **Max delta neg** / **Negative delta** = (100 steps per revolution * 32,768 revolutions) - 1 – 100 steps (i.e. 1 revolution for safety reasons) = 3 276 699 pulses
Max. **SW limit switch +** = (-500) + 3 276 699 = 3 276 199 pulses (forward travel)
Max. **SW limit switch -** = (-500) – 3 276 699 = -3 277 199 pulses (backward travel)
Therefore, when **Preset** = - 500, the working stroke of the axis will span the following positive and negative limits range, that is max. **SW limit switch +** = + 3 276 199 and max. **SW limit switch -** = – 3 277 199.



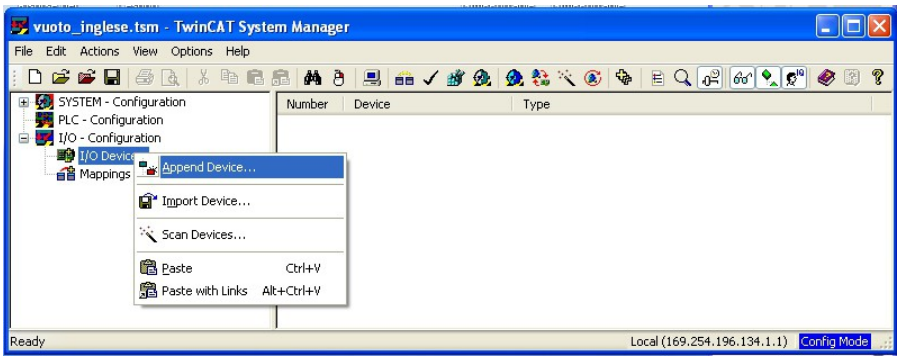
7 EtherCAT® interface

Lika actuators are Slave devices and support "CANopen Over EtherCAT" (COE) mode for data transfer. In particular, they support the "CANopen DS301 Communication profile".
For any omitted specification on CANopen® protocol, please refer to the "CiA Draft Standard Proposal 301 CANopen Application layer and communication profile" document available at the address **www.can-cia.org**.
For any omitted specification on EtherCAT® protocol, please refer to "ETG.1000 EtherCAT Specification" document available at the address **www.ethercat.org**.

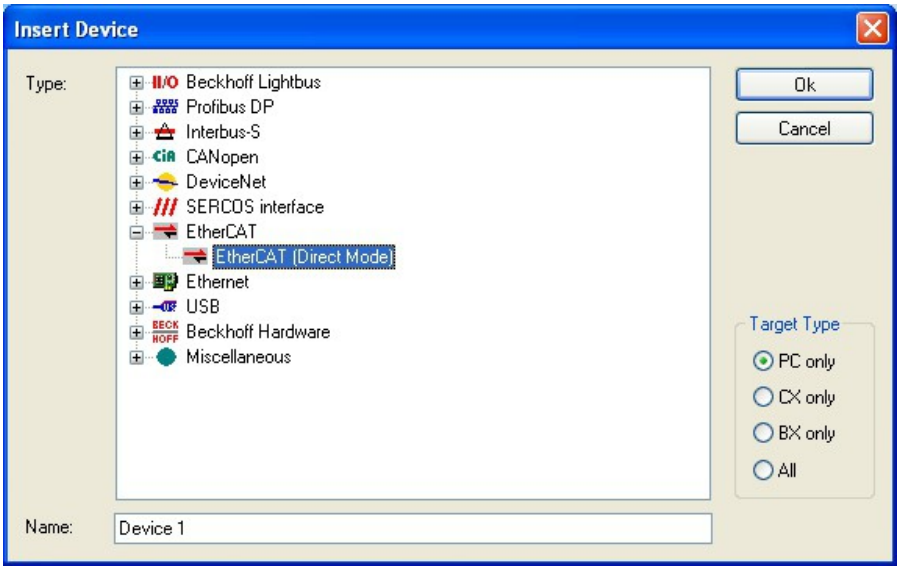
7.1 System configuration using TwinCAT software system from Beckhoff

7.1.1 Setting the Network Card

Launch **TwinCAT System Manager**.
In the left pane of the main window extend the devices tree and select the **I/O Devices** item; right-click the **I/O Devices** item and then press the **Append Device...** command.



In the **Insert Device** window select the **EtherCAT (Direct Mode)** item and confirm pressing the **OK** button.

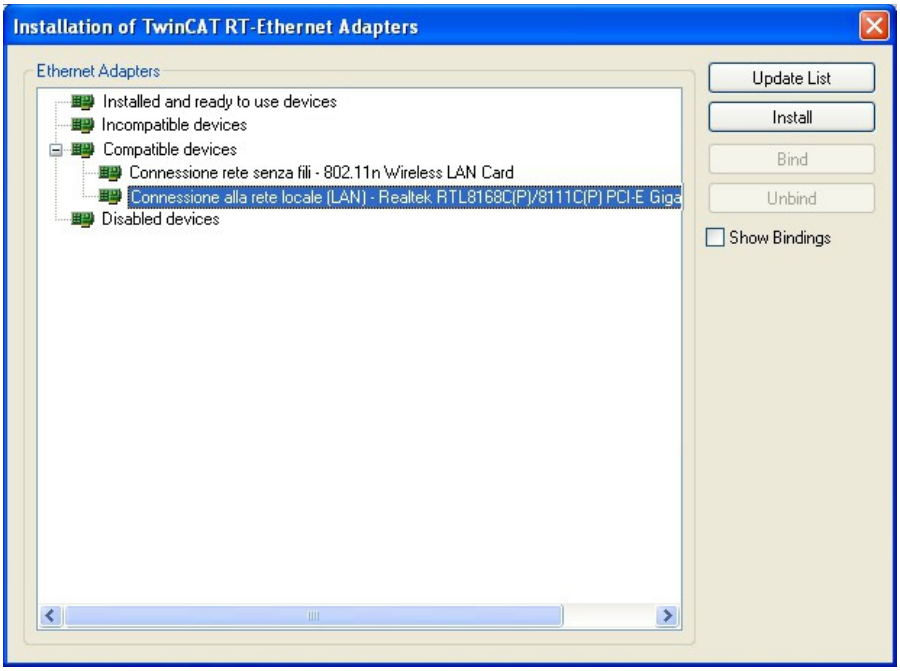


If a network card has been already installed properly, the following window will appear showing the list of the installed devices.

Select the network card you want to use and then confirm the choice pressing the **OK** button.

If there are no network cards installed, you must install one before proceeding.
To do this, on the menu bar of the **TwinCAT System Manager** main window, select the **Options** menu and then press the **Show Real Time Ethernet**

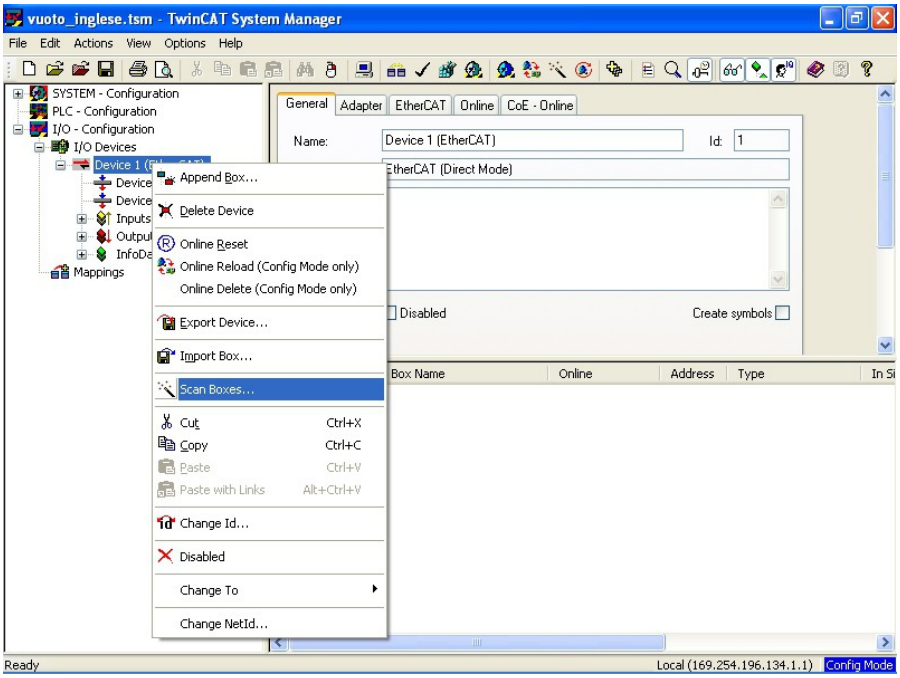
Compatible Devices... command. The **Installation of TwinCAT RT – Ethernet Adapter** window will appear.



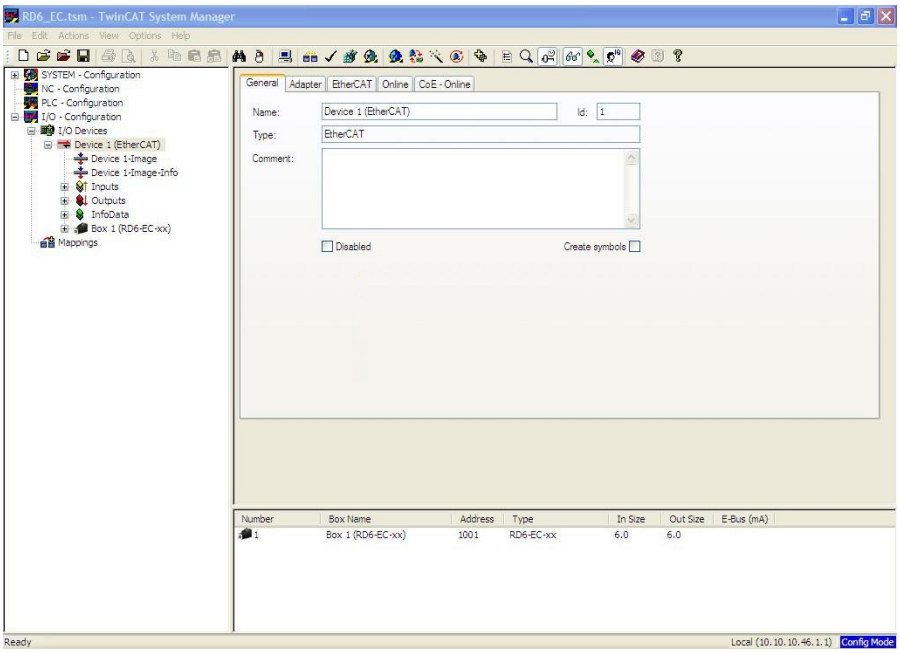
Now select **Compatible Devices** item and choose the network card you want to install; finally press the **Install** button to confirm your choice.

7.1.2 Add new I/O modules (Boxes)

If devices are connected to the network and switched ON, right-click the **Device 1 (EtherCAT)** item in the left pane of the **TwinCAT System Manager** main window and press the **Scan Boxes...** command.

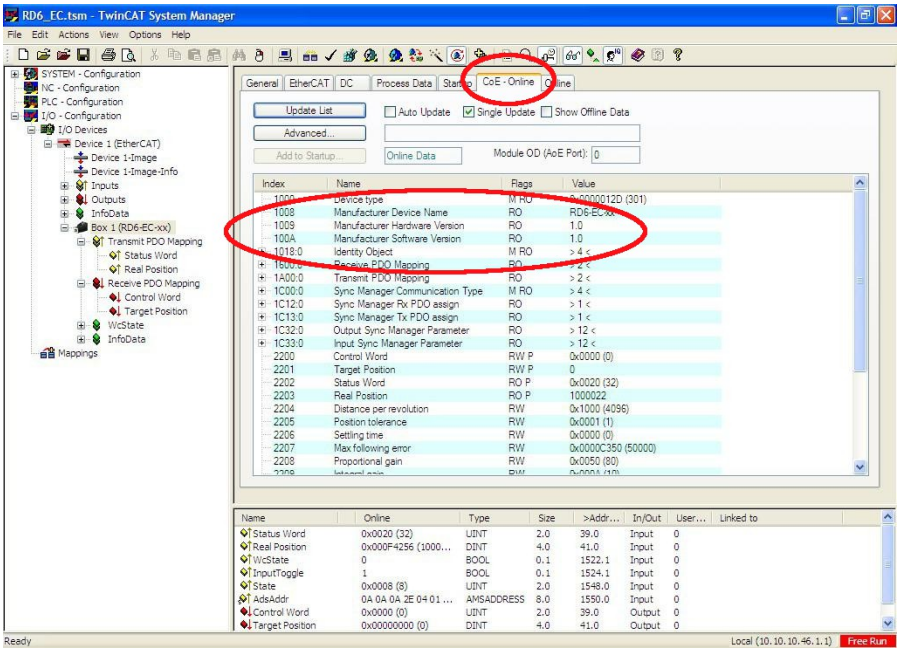


At the end of the process some information will be listed in the right page as in Figure here below.

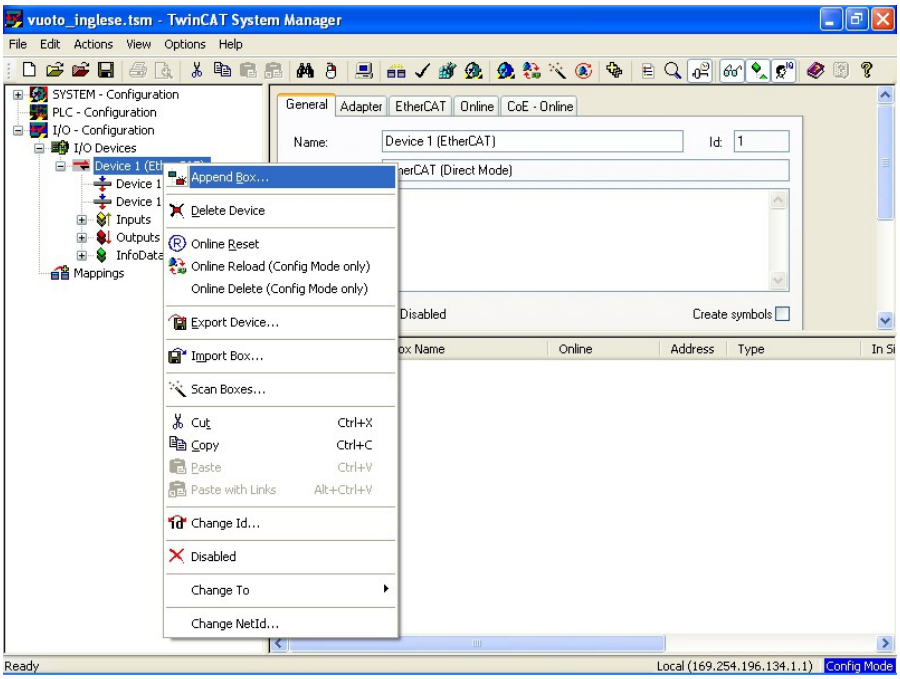


If devices are not already connected to the network it is necessary to use the XML file supplied with the actuator: **Lika_RDxx_EC_Vx.xml** (see at **www.lika.biz > ROTARY ACTUATORS > ROTARY ACTUATORS/POSITIONING UNITS (DRIVECOD**).

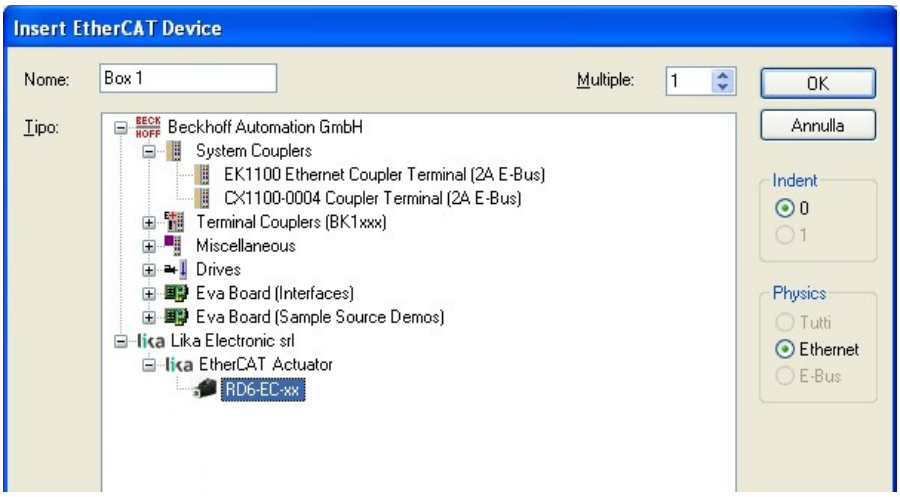
If you want to know the firmware version of a device, press the **Box (RD6-EC-xx)** item in the left pane of the **TwinCAT System Manager** main window: some tabbed pages for configuring and managing the device will appear in the right pane. Enter the **CoE - Online** page and refer to the **1009-00 Manufacturer Hardware version** and **100A-00 Manufacturer Software version** indexes.



Right-click the **Device 1 (EtherCAT)** item in the left pane of the **TwinCAT System Manager** main window and press the **Append Box...** command.



The **Insert EtherCAT Device** window will appear.



In the **Insert EtherCAT Device** window that appears select **Lika Electronic srl** and then **EtherCAT Actuator** items; now choose from the list the actuator to install (if more actuators are available).

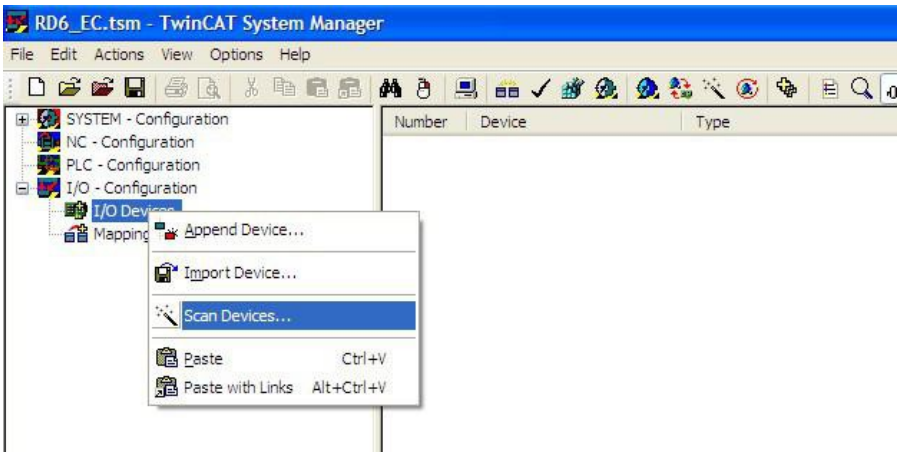
Press the **OK** button to confirm your choice.

7.2 Setting the communication mode

Lika actuators with EtherCAT interface only support the FreeRun communication mode.

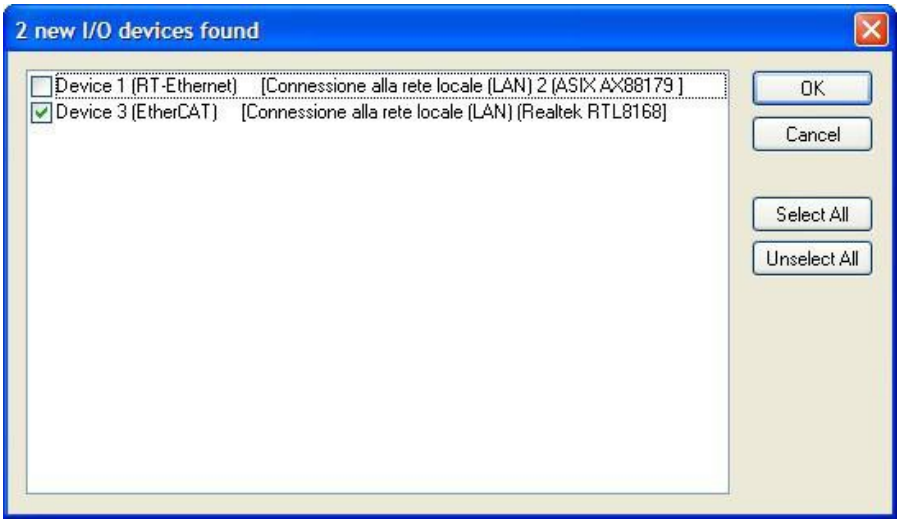
7.2.1 FreeRun communication mode

When you first scan the bus for connected EtherCAT devices, you are requested to activate the FreeRun communication mode.
To do this select **I/O Devices** and then right-click to open the drop-down menu. Select the **Scan Devices...** command.



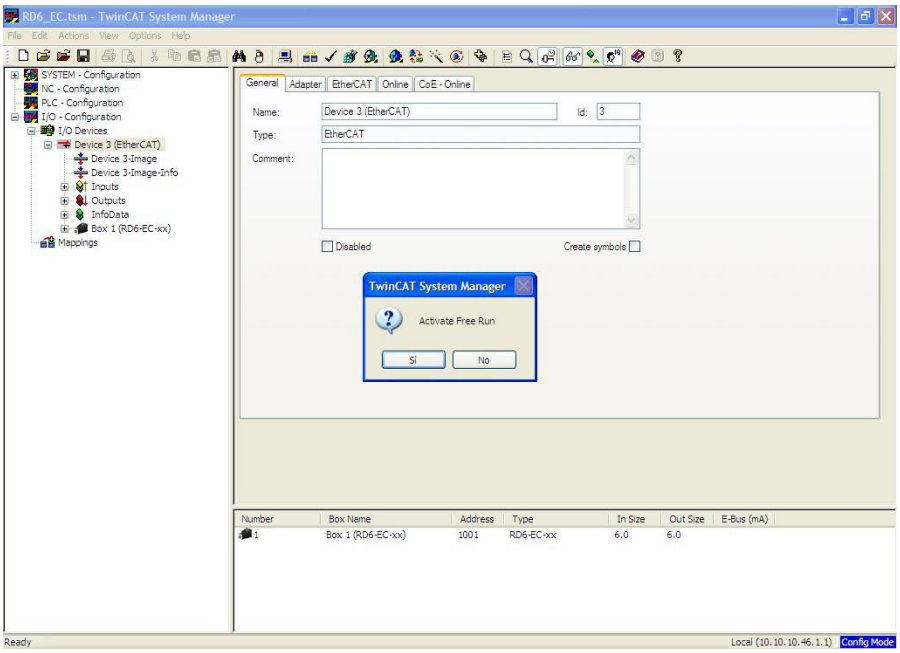
A warning message dialog box will appear on the screen: press **OK** to start scanning for EtherCAT devices.

Select the EtherCAT interface card among the detected ones.

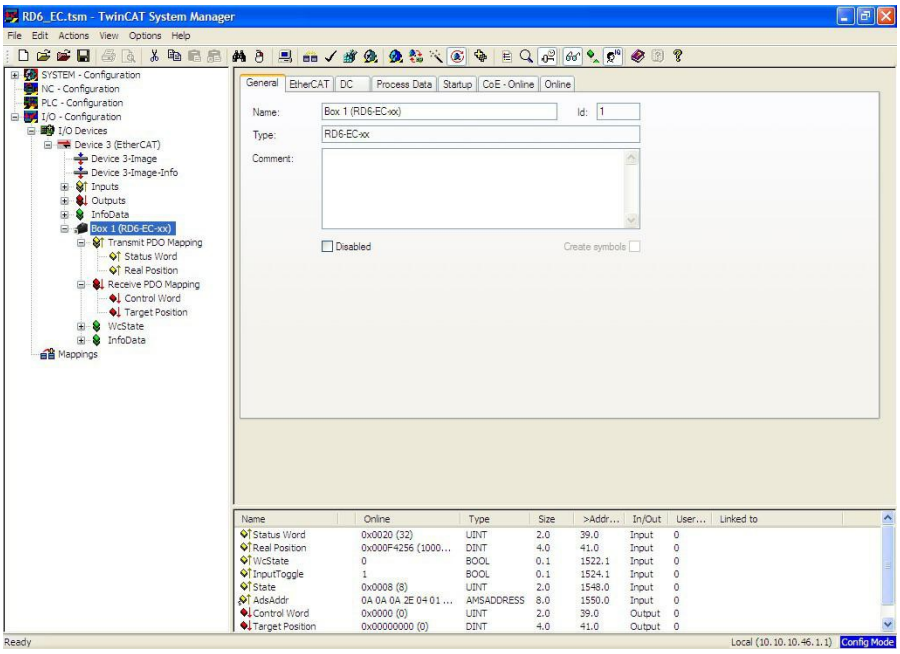


Press **YES** and confirm to start the "Scan for boxes" procedure and search for connected devices.

Press **YES** to activate the FreeRun communication mode.



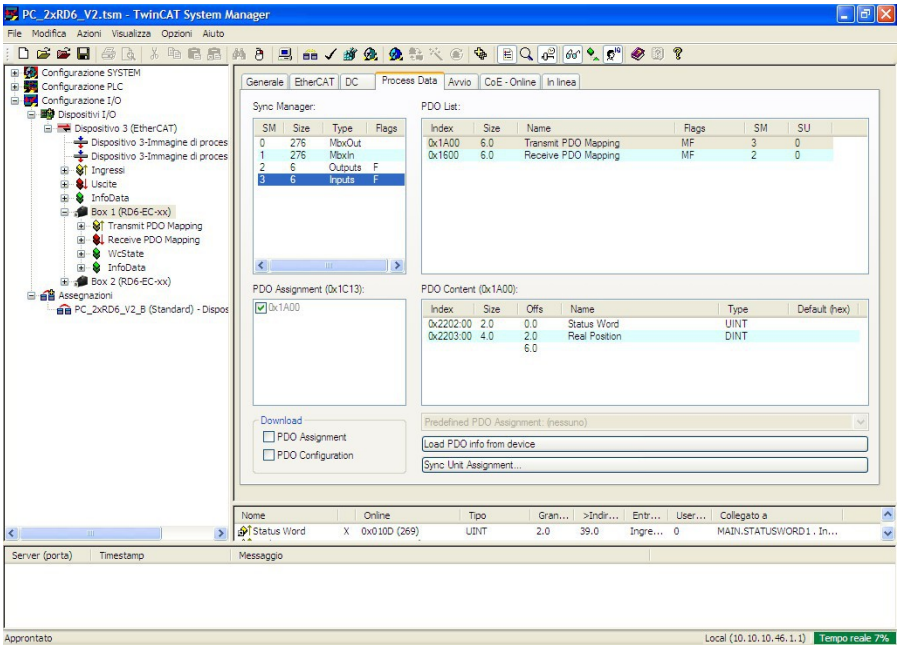
After activating the FreeRun communication mode the page will appear as follows:



For any further information on the FreeRun operation mode please refer to the "FreeRun" section on page 67 and to the **1C33 Input Sync Manager Parameter** object on page 83.

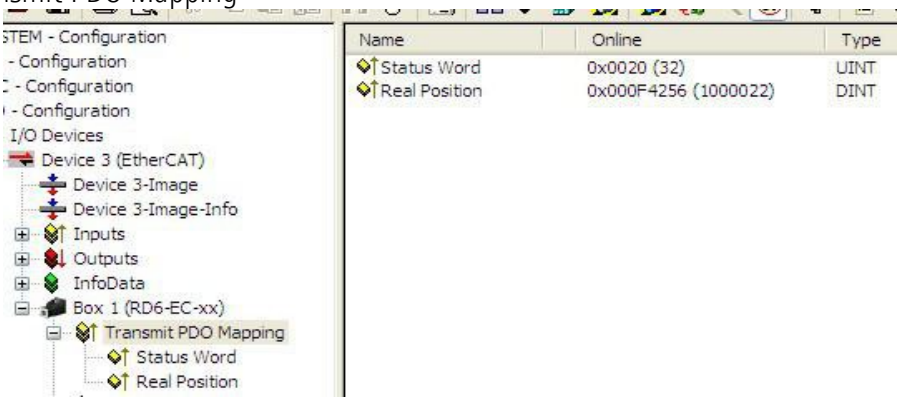
7.3 Process Data Objects

In the left pane of the **TwinCAT System Manager** main window press the **Box (RD6-EC-xx)** item. Expand the box to see Process Data Inputs (PDI) and Outputs (PDO). some tabbed pages for configuring and managing the device will appear in the right pane. Enter the **Process Data** page. In this page process data objects (RxPDO and TxPDO Mapping) are shown.

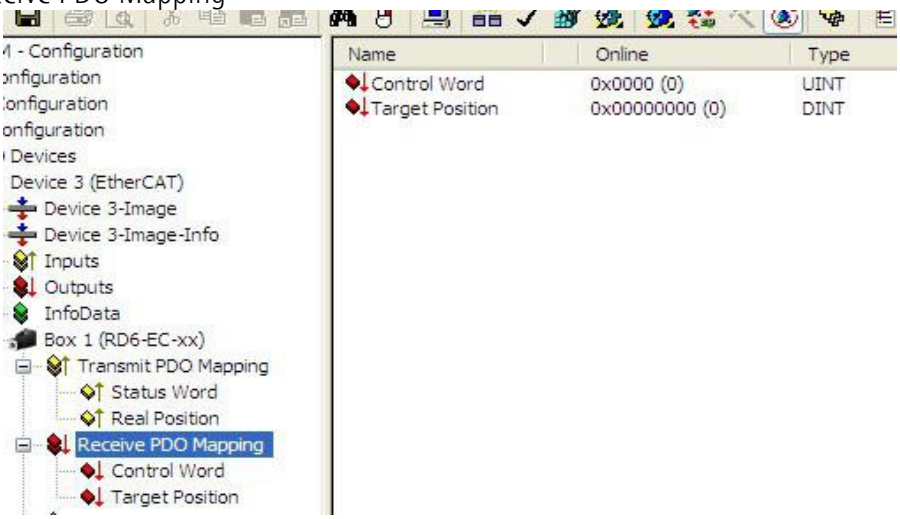


Process data objects can be displayed also by pressing the **Transmit PDO Mapping** and **Receive PDO Mapping** items in the left pane of the **TwinCAT System Manager** main window; data is listed in the right pane.

Transmit PDO Mapping

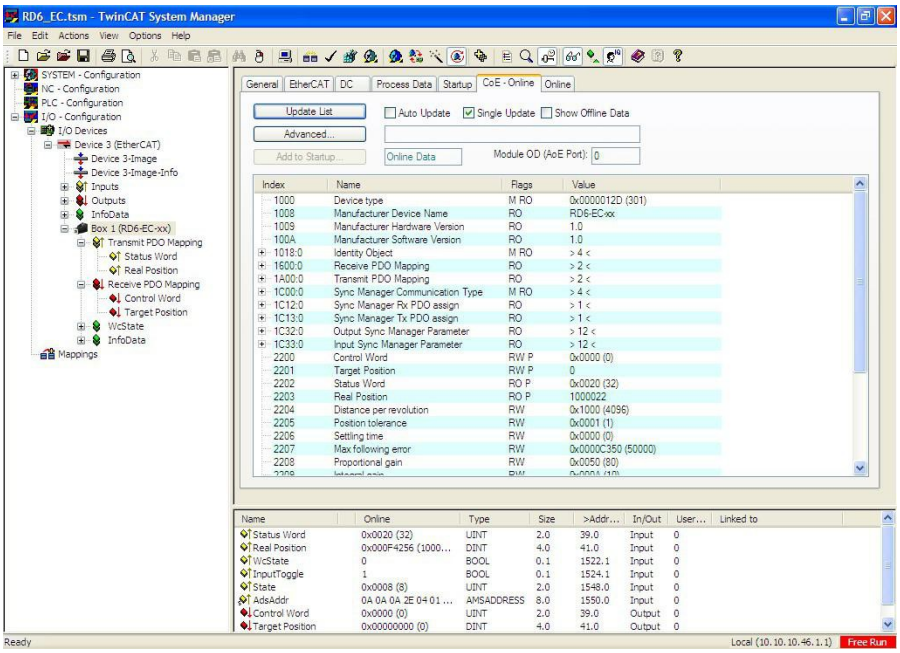


Receive PDO Mapping

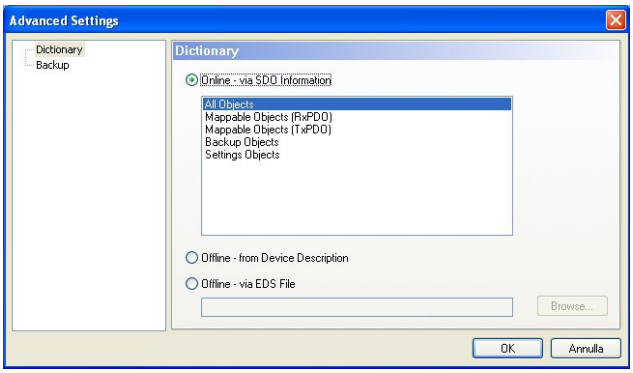


7.4 COE Object Dictionary

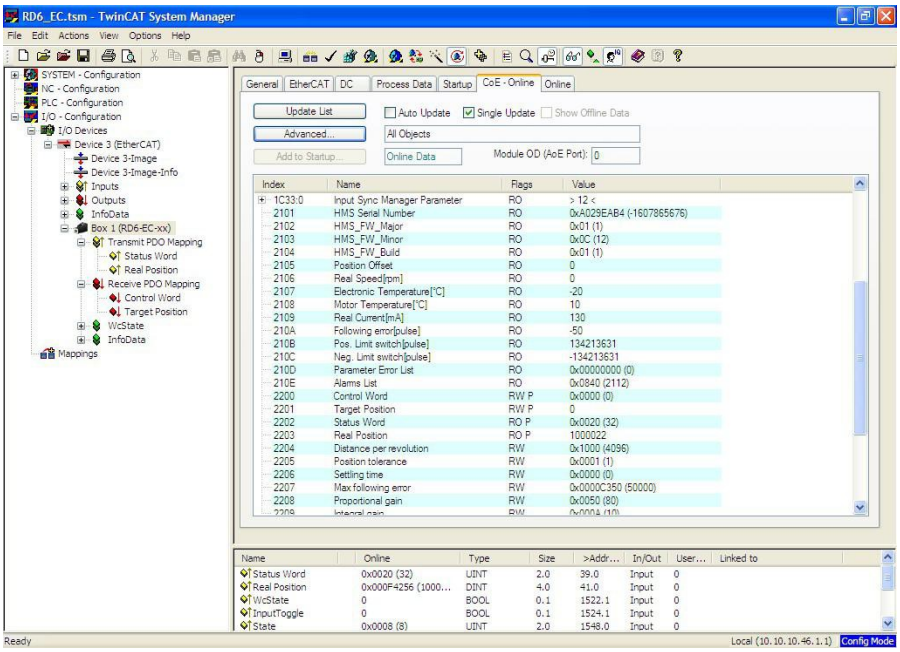
In the left pane of the **TwinCAT System Manager** main window press the **Box (RD6-EC-xx)** item: some tabbed pages for configuring and managing the device will appear in the right pane. Enter the **CoE - Online** page. In this page the objects dictionary is shown. This is the offline version of the objects dictionary as read from the xml file.



Objects can also be read directly from the actuator; to do this click the **Advanced...** button: the **Advanced Settings** window will appear.

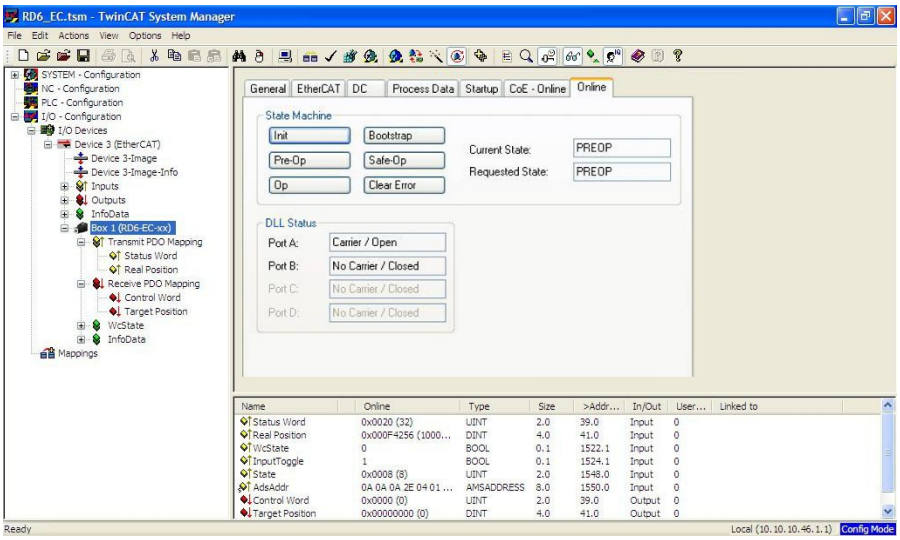


Select the **Dictionary** item in the left pane and then choose the **Online - via SDO Information** option in the **Dictionary** page; press the **OK** button to confirm.
Online version of the objects dictionary as read from the actuator.



7.5 Online Data

In the left pane of the **TwinCAT System Manager** main window press the **Box (RD6-EC-xx)** item: some tabbed pages for configuring and managing the device will appear in the right pane. Enter the **Online** page to check the actuator status.



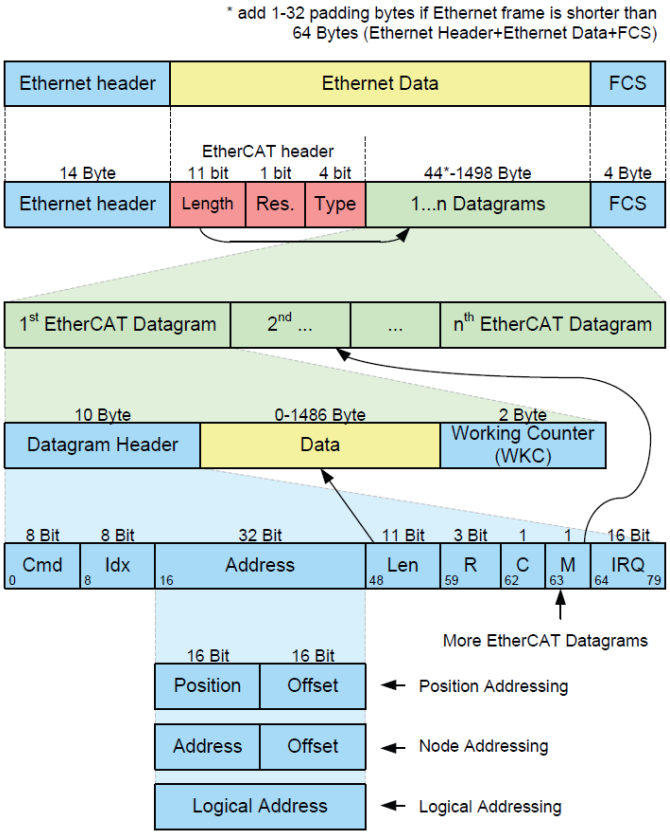
To display the actuator process data in real time, click the **Safe-OP** button if you want to display inputs only; click the **OP** button if you want to display both inputs and outputs.

WARNING

The structure of Data Objects (PDO and SDO) requires bytes to be sent from the Least Significant Byte (LSB) to the Most Significant Byte (MSB).
On the contrary, in TwinCAT you must write and read data from MSB to LSB.
Furthermore in TwinCAT also strings must be entered in the reverse order.

7.6 Basic Information on the EtherCAT® Protocol

The EtherCAT protocol is designed to use the standard Ethernet dataframes for issuing data; in addition, and as regards the hardware, it is not necessary to install dedicated Masters for establishing and managing the EtherCAT communication because standard Ethernet network cards can be used. This results in a great advantage in terms of lower costs and simplicity of use because Ethernet network cards are used in standard personal computers and are easily commercially available.
An EtherCAT bus can be viewed as a single and large Ethernet device that receives and sends Ethernet telegrams; it can be considered an Ethernet subnet supported by an Ethernet dataframes structure.
However this "subnet" must be fitted with one only EtherCAT Master controller and several EtherCAT Slaves, but no Ethernet controller with downstream microprocessor must be present.
Here follows an Ethernet frame structure with EtherCAT:

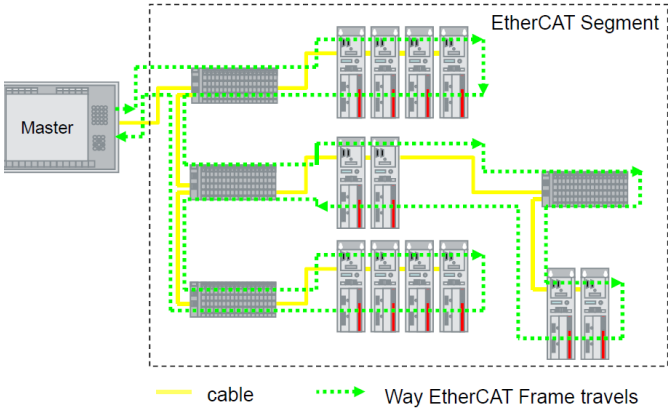


Inside the Ethernet frames, data are transmitted among Master and Slaves using PDO (Process Data Objects) protocol. Each PDO message has inside one or more addresses for issuing data to the Slaves; data + address/es (and additional elements such as a validation checksum) joined together form an EtherCAT telegram (Datagram).

An EtherCAT frame can contain several telegrams and a complete control cycle often requires more than one frame.

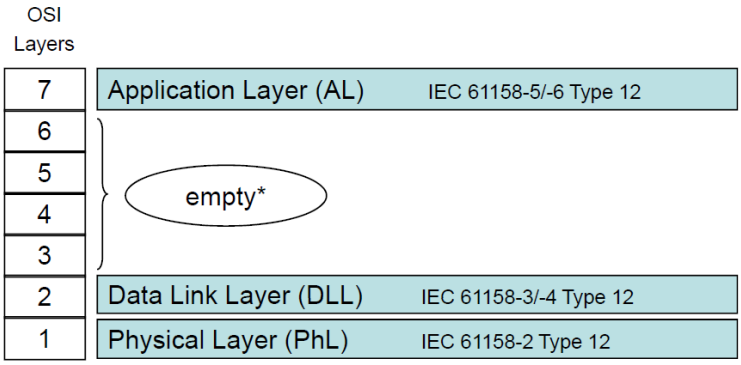
7.6.1 Data transfer

Usually, in a data bus system, Master controller sends online a data request and then waits for data to be processed and sent back from each Slave node; this does not comply with a real-time system because the Master receives data from the Slaves in different moments and the whole system cannot be synchronized.
In EtherCAT the real-time characteristic of the system is quite improved because data are processed "on the fly", using one single frame to acquire all data from all Slaves.
In fact the frame sent by the Master is read by each Slave node the data is addressed to while the telegram passes through the device; similarly, input data is inserted while the telegram passes through. Then the telegram is forwarded to the next device. Telegrams are only delayed by a few nanoseconds.
The last Slave issues back the complete frame to the Master with all the requested data (again passing through all the Slaves).



This efficient data flow is guaranteed by the 100BASE-TX full-duplex structure of the EtherCAT bus which is fitted with two separate lines for transmitting and receiving data.
Moreover the protocols exchange takes place inside the hardware and it is thus independent from CPU and software processing.

7.6.2 ISO/OSI Layer model

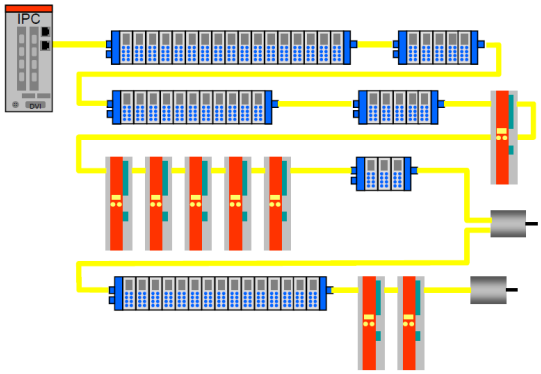


* "empty" means that the layer behaviour exists, but is not shown explicitly.

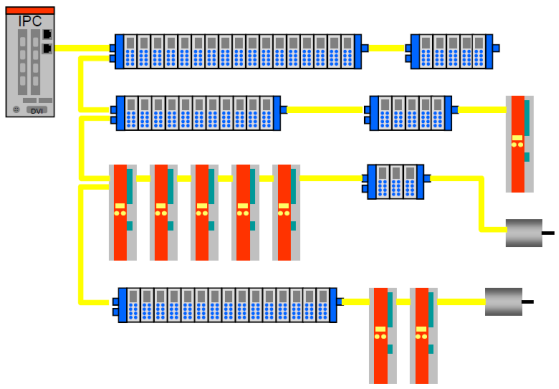
7.6.3 Topology

Several topologies of connection are supported by EtherCAT networks: line, tree, daisy-chain, star, ...). EtherCAT networks can be configured in almost any topology in the same structure. The maximum length of the cable between two Slaves is 100 m (328 ft); standard EtherCAT cables commercially available can be used.
The choice of the topology depends on the structural characteristics of the plant and it is made in order to reduce the complexity and time for cabling.
Inside an EtherCAT network up to 65,535 devices can be connected.
Some topology examples are shown in the Figures below:

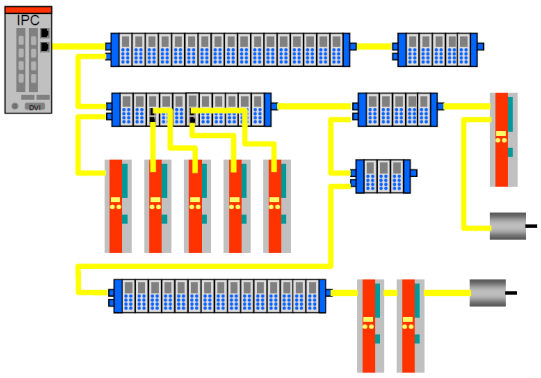
LINE topology:



TREE topology:



DAISY CHAIN with drop lines topology:

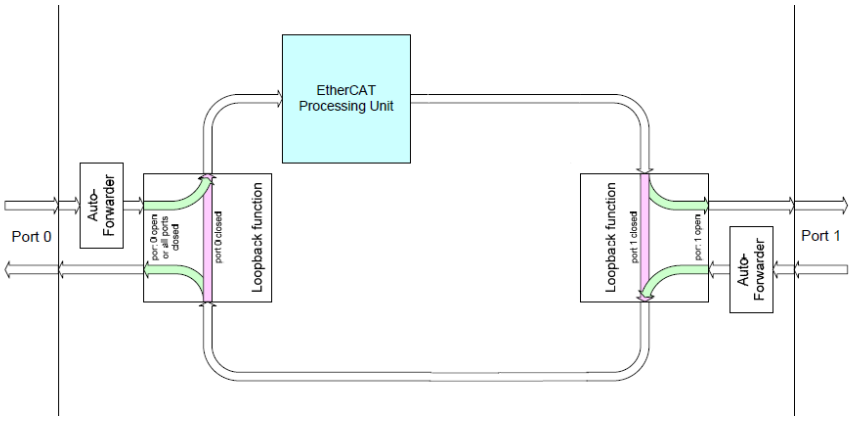


7.6.4 Line Termination

EtherCAT network needs no line termination because the line is terminated automatically; in fact every Slave is able to detect the presence of the downstream Slaves.
An EtherCAT Slave is able to detect the presence of the signal in the outgoing line (Port 0) or in the return line (Port 1).
If the Slave is not able to detect the signal in its return line then it closes the communication ring by short-circuiting the TX signal of its outgoing line with the RX signal of its return line; in this way a telegram received through the outgoing line is processed and sent back through the TX of the return line.

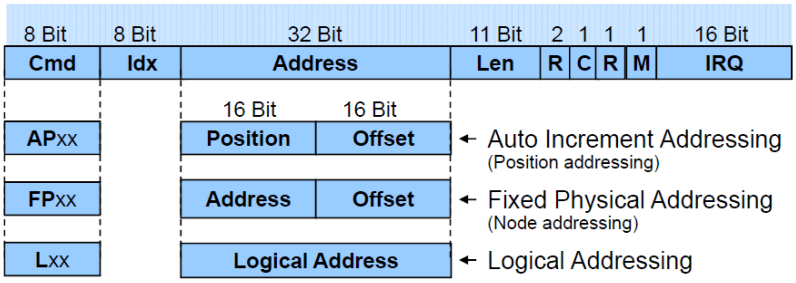
The Slave sends a "carrier signal" or a telegram on TX of the outgoing line continuously and, once the next Slave is connected again, a signal on RX of the

return line is detected again; so the short circuit is removed and the telegrams are sent on TX of the outgoing line.



7.6.5 Addressing

It is not necessary to assign a physical address to the device (for instance using a dip-switch) because the addressing of the Slave is automatic at power-on during the initial scanning of the hardware configuration.



The field for addressing is 32 bit-long; EtherCAT allows three kinds of addressing:

- Auto Increment Addressing = Position Addressing: 16 bits indicate the physical position of the Slave inside the network while 16 bits are scheduled for local memory addressing; when the Slave receives the frame then it increments the position address and the Slave receiving address 0 is the addressed device;
- Fixed Addressing = 16 bits indicate the physical address of the Slave inside the network while 16 bits are scheduled for addressing the local memory - not used;
- Logical Address = the Slave is not provided with its own individual address, but it can read and write data in a section of the total memory space available (4 Gigabytes) – not used.

7.6.6 Communication mode

EtherCAT interface allows three kinds of operating mode:

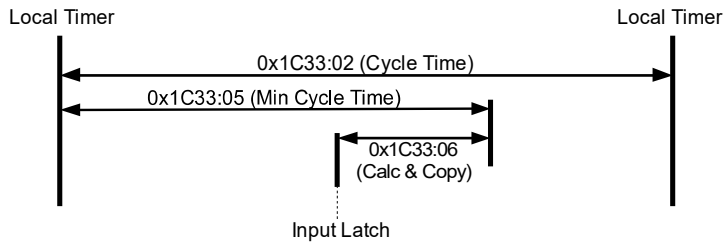
- FreeRun: asynchronous mode;
- SM3 event: synchronous mode;
- DC: distributed clock synchronization mode (synchronous mode).

Lika actuators with EtherCAT interface only support the FreeRun operating mode.

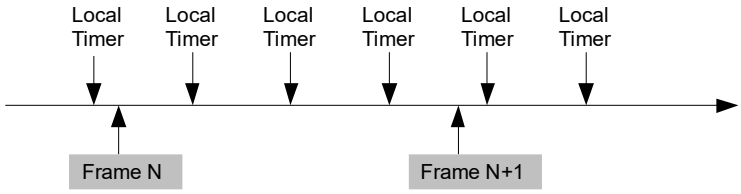
For a system that requires high performances in real time (closed-loop applications) DC mode can be used; if real time requirements are not so mandatory SM3 or Freerun modes can be used instead.
A reference parameter is the "Jitter": it represents the temporal fluctuation of the instant data sampling; in other words data sampled by the micro-controller is available in ECAT DPRAM memory after a certain time and the measure of the variability over time is the "jitter".

FreeRun

Asynchronous mode; TxPDOs and RxPDOs are sampled directly from the EtherCAT frame sent by the Master; their update is performed by an internal timer of the controller every 1 millisecond.



This operating mode has a high sampling jitter (up to 1 millisecond) and can be chosen only when cycle times are quite longer than the jitter if we want to ensure a sufficient real-time system performance.



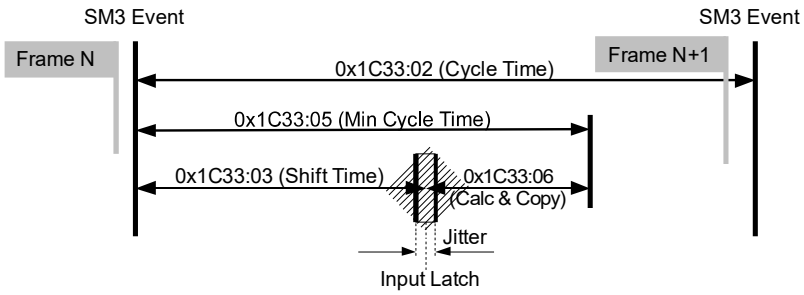
Description	Min	Typ	Max	
Jitter	0		1	msec
Cycle Time	1		64	msec

See the 1C33 Input Sync Manager Parameter entry on page 83.

Synchronous with SM3

This operating mode is not supported by Lika actuators.

In this mode data is sampled and then copied into the Sync Manager buffer as soon as previous data was read from the Master (SM event); in this way new sampled data is synchronous with Master readings.



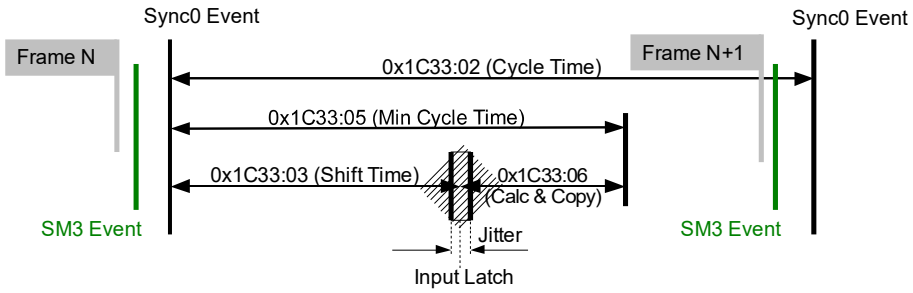
New data will be read by the Master at next cycle (following SM3 event), so if the cycle time is too long, data could be relatively old for a real-time system.
The main advantage is that data is updated exactly when the Master is reading (synchronous mode).

Description	Min	Typ	Max	
Jitter	0		7.2	ns
Cycle Time	62.5		64000	µs

Synchronous with DC SYNC0

This operating mode is not supported by Lika actuators.

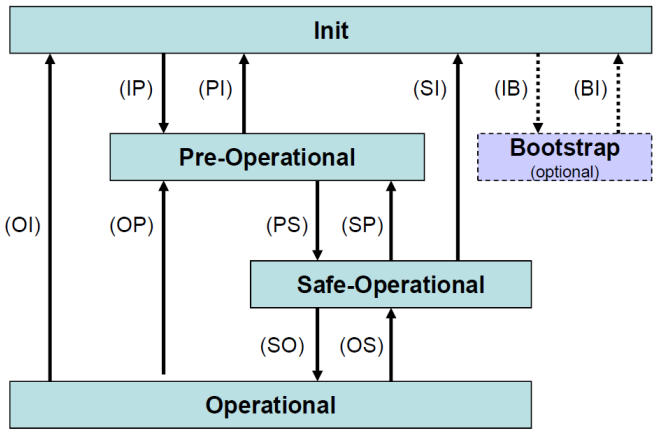
In this operating mode data is sampled and then copied into the Sync Manager buffer simultaneously at SYNC0 event generated by the ESC capture/compare unit.
Time required for accomplishing these operations is set next to the 1C33 Input Sync Manager Parameter object; in particular object **03 Shift Time** (1C33hex, sub3) and object **06 Calc and Copy Time** (1C33hex, sub6).



In this operating mode "Jitter" is a fundamental parameter in the sampling of two consecutive data. The main advantage of this mode is that there is a direct relation between the sampling instant and the absolute time of the system; in this way, if we know the shift times of the Slaves, we can have an exact image of the system at a given moment (with a tolerance equal to jitter).

Description	Min	Typ	Max	
Jitter	0	100	200	µsec
Cycle Time	62.5		64000	µsec

7.6.7 EtherCAT State Machine (ESM)



EtherCAT Slave is a state machine; communication and operating characteristics depend on the current state of the device:

- INIT: it is the default state after power-on; in this state there is not direct communication between the Master and the Slave on the Application Layer;

some configuration registers are initialized and the Sync Managers are configured;

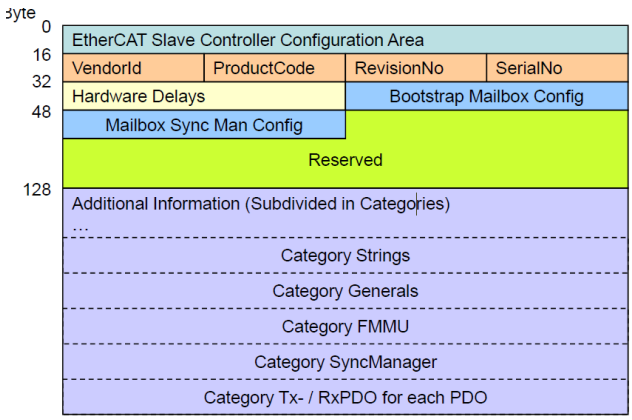
- PRE-OPERATIONAL (PREOP): in this state mailbox is active; both the Master and the Slave can use the mailbox and its protocols for exchanging specific initialization parameters of the application. Exchange of Process Data (PDO) is forbidden;
- SAFE-OPERATIONAL (SAFEOP): in this state the Master and the Slave can issue only input process data, while output process data are still in SAFE-OPERATIONAL state;
- OPERATIONAL (OP): in this state the Master and the Slave are enabled to send both input process data and output process data.
- BOOTSTRAP (BOOT). Optional. No process data communication. Communication only via mailbox on Application Layer available. Special mailbox configuration is possible, e.g. larger mailbox size. In this state usually the FoE protocol is used for firmware download. **Please do not use, the unit could be irreparably damaged.**

The current state of the Slave device is signalled through the LED L6, see on page 36.

7.6.8 Slave configuration

The configuration of the Slave communication characteristics (Sync Manager configuration, addresses, synchronization modes, PDO mapping, ...) can be made both using the XML file (EtherCat Slave Information - ESI) or loading data directly from EEPROM (Slave Information Interface SII).

EEPROM content (SII):



7.6.9 Timing and Synchronization

The main feature of EtherCAT is its almost ideal representation of a real-time system.
Hence the Master has to synchronize all the Slaves at the same time in order to build a system where all nodes have the same reference time, this goal can be achieved by using "distributed clocks".
The Master downloads its clock into one of the Slaves (customarily the first Slave) which becomes the reference clock for all the Slaves in the network; so it has the task of synchronizing the other Slaves. The Master controller periodically sends a special synchronization-telegram where the reference Slave writes its own "current time". This telegram is then sent to all the other Slaves that, in this way, provide for a new re-synchronization of their own clock in order to avoid possible drifts.
This synchronization of the reference time is very important in order to have a snapshot of the system and accordingly to take simultaneous actions in high sensitive applications such as the coordination in axis control operations.
Besides, the EtherCAT Slave Controller (ESC) is fitted with a capture/compare unit that provides accurate synchronization signals (SYNC0 or interrupts): they are sent to the local micro-controller so that it is able to synchronize its own clock to the Slaves clock.

Sync Manager

Sync Manager has the task of synchronizing data transfer between the Master and the Slave and prevents the same memory area from being written by different events.
There are two synchronization modes:

- 3-Buffer Mode;
- 1-Buffer Mode.

Synchronisation mode is initialized through the XML file or by loading data directly from EEPROM (SII).

Buffered Mode (3-Buffer Mode)

In this mode new data can be accessed at any time by both the EtherCAT Master and the ESC controllers; no timing restrictions are imposed.
Three buffers are necessary (three consecutive memory areas); one buffer is always available to the ESC controller for writing and one buffer always contains updated data to be read by the Master.
The buffered mode is typically used for cyclic data exchange, i.e. process data.

Mailbox Mode (1-Buffer Mode)

In this mode a "handshake" between the Master and the Slave must be used; in fact one only memory buffer is available to both the Master and the Slave for writing and reading; the Master (or the Slave) is enabled to write only when the buffer is empty, that is when the Slave (or the Master) has finished reading the

data buffer. And vice versa: the Master (or the Slave) is enabled to read only when the buffer is empty, that is when the Slave (or the Master) has finished writing the data buffer. The mailbox mode is typically used for application layer protocols and exchange of acyclic data (e.g. parameter settings).

The actuator features four Sync Managers, see the **1C00 Sync Manager Communication Type** object:

- **Sync Manager 0 (SM MailBox Receive, SM0)**
  Used for mailbox write transfers (Master to Slave).
  The module has a configurable write mailbox size with default size of 276 bytes, corresponding to 255 bytes plus relevant protocol headers and padding.
- **Sync Manager 1 (SM MailBox Send, SM1)**
  Used for mailbox read transfers (Slave to Master).
  The module has a configurable read mailbox size with default size of 276 bytes, corresponding to 255 bytes plus relevant protocol headers and padding.
- **Sync Manager 2 (SM PDO output, SM2)**
  It contains the RxPDOs (i.e., Sync Manager 2 holds the Read Process Data).
- **Sync Manager 3 (SM PDO input, SM3)**
  It contains the TxPDOs (i.e., Sync Manager 3 holds the Write Process Data).

7.6.10 Watchdog

RD6 EtherCAT actuators are equipped with a safety feature (watchdog) that switches off the outputs after the set time e.g. in the event of an interruption of the process data traffic, depending on the device and settings, e.g. in OFF state. The unit is recovered to the **SAFE-OPERATIONAL** state. Apart from the standard watchdog functionality, three types of watchdog are mainly implemented:

- SM watchdog (default: 100 ms)
- Output I/O watchdog (default: 100 ms)
- PDI watchdog (default: 100 ms)

SM watchdog (SyncManager Watchdog)

The SyncManager watchdog is reset after each successful EtherCAT process data communication with the terminal. If no EtherCAT process data communication takes place with the actuator for longer than the set and activated SM watchdog time, e.g. in the event of a line interruption, the watchdog is triggered

and the outputs are set to FALSE. The OP state of the terminal is unaffected. The watchdog is only reset after a successful EtherCAT process data access.
The SyncManager watchdog monitors correct and timely process data communication with the ESC from the EtherCAT side.

Output I/O Sync Manager Watchdog
If enabled, this watchdog monitors the PDO communication towards the module. If the Master does not update the Read Process Data within the specified time period, this will trigger a timeout condition in the module, causing it to shift from OPERATIONAL to SAFE-OPERATIONAL. The supervision-bit (SUP) is also affected by this.

PDI watchdog (Process Data Watchdog)
If no PDI communication with the EtherCAT Slave Controller (ESC) takes place for longer than the set and activated PDI watchdog time, this watchdog is triggered. PDI (Process Data Interface) is the internal interface between the ESC and local processors in the EtherCAT Slave, for example. The PDI watchdog can be used to monitor this communication for failure.
The PDI watchdog monitors correct and timely process data communication with the ESC from the application side.
The settings of the SM- and PDI-watchdog must be done for each Slave separately in the TwinCAT System Manager.

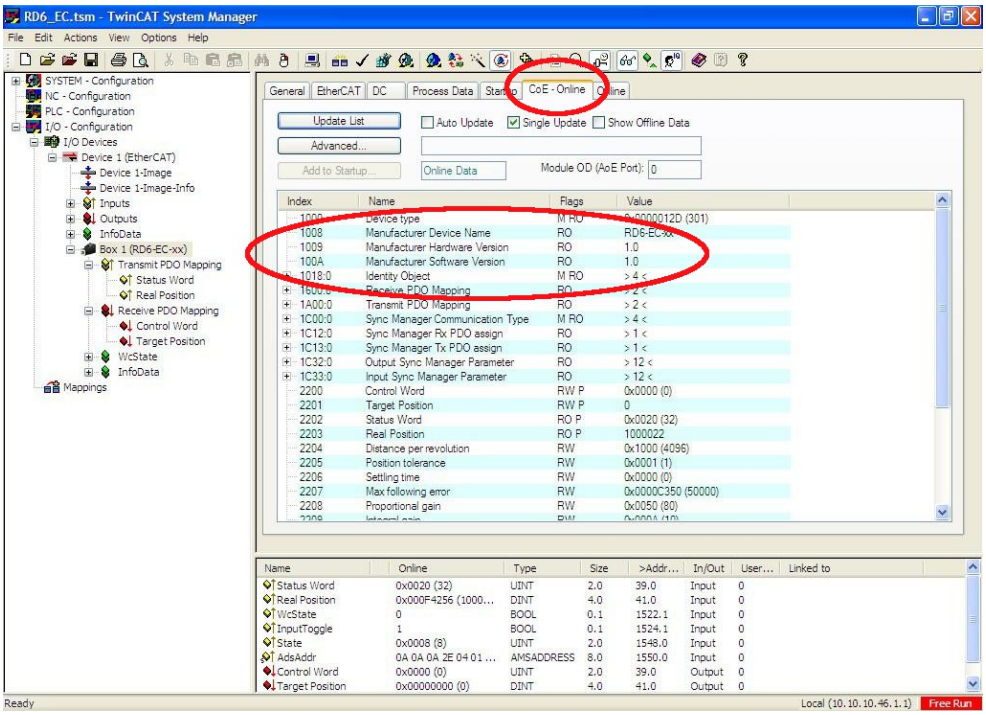
7.7 CANopen Over EtherCAT (CoE)

Lika actuators are Slave devices and support "CanOpen Over EtherCAT" (COE) mode for data transfer. In particular, they support the "CANopen DS301 Communication profile".
For any omitted specification on CANopen® protocol, please refer to the "CiA Draft Standard Proposal 301 CANopen Application layer and communication profile" document available at the address **www.can-cia.org**.
For any omitted specification on EtherCAT® protocoll, please refer to "ETG.1000 EtherCAT Specification" document available at the address **www.ethercat.org**.

7.7.1 XML file

EtherCAT® actuators are supplied with their own XML file **Lika_RDxx_EC_Vx.xml** (see at **www.lika.biz > ROTARY ACTUATORS > ROTARY ACTUATORS/POSITIONING UNITS (DRIVECOD)**.
The XML file must be installed in the EtherCAT® Master device.

If you want to know the firmware version of a device, press the **Box (RD6-EC-xx)** item in the left pane of the **TwinCAT System Manager** main window: some tabbed pages for configuring and managing the device will appear in the right pane. Enter the **CoE - Online** page and refer to the **1009-00 Manufacturer Hardware version** and **100A-00 Manufacturer Software version** indexes.



7.7.2 Communication messages

EtherCAT Datagram of CoE mode has the following structure:

Mbx Header	CoE Cmd			Cmd specific data
type = 3	Number	res	Type	
6 bytes	9 bits	3 bits	3 bits	0 ... 1478 bytes

Mbx Header = 3 CoE mode

Number = 0 in case of SDO messages
≠ 0 in case of PDO messages, it defines the type of service

res reserved bits

Type = 0 reserved
= 1 Emergency message
= 2 SDO request
= 3 SDO response
= 4 Transmitted PDO (TxPDO)
= 5 Received PDO (RxPDO)
= 6 Remote transmission request of TxPDO
= 7 Remote transmission request of RxPDO
= 8 SDO information
= 9 ... 15 reserved

Cmd specific data PDO messages: are the process data, e.g. position value
SDO messages: standard CANopen frame

Transmit (tx) or receive (rx) "Type" is viewed from the Slave.

7.7.3 Process Data Objects (PDO)

PDO messages are used for transmitting or receiving process data in real time; data to be transmitted or received is defined in PDO Mapping and managed by Sync Manager PDO Mapping.

7.7.4 Service Data Objects (SDO)

SDO messages are issued via Mailbox (low priority data); Segmented SDO Service and SDO Complete Access are not supported (transfer of low size data and one sub-index at a time).

"CoE Cmd type" = 2 or 3

Structure of "Cmd specific data":

Cmd specific data				
SDO control	Index	Sub index	Data	Data optional
8 bits	16 bits	8 bits	32 bits	1 ... 1470 bytes

SDO control	standard CANopen SDO Service
Index	parameter index
Sub index	parameter sub-index
Data	parameter value
Data optional	optionally, more then 4 bytes of data can be sent in one frame. Full mailbox size usable.

Index and sub-index values are described in the "Object dictionary".

7.8 Object dictionary

The most important part of a device profile is the Object Dictionary. The Object Dictionary is essentially a grouping of objects accessible via the network in an ordered, pre-defined mode. Each object within the dictionary is addressed using a 16-bit index.
The Object Dictionary can contain a maximum of 65,536 entries.
The user-related objects are grouped in two main areas: the Communication Profile Area and the Manufacturer Specific Profile Area. The objects are all described in the XML file.

The **Communication Profile Area** at indexes from 1000h to 1FFFh contains the communication specific parameters for the EtherCAT network. These entries are common to all devices. PDO objects and SDO objects are described in this section. The Communication Profile Area objects comply with the "CiA Draft Standard Proposal 301 CANopen Application layer and communication profile". Refer to the "7.8.1 Communication Profile Area objects (DS 301)" section on page 78.

The **Manufacturer Specific Profile Area** at indexes from 2000h to 22FFh is free to add manufacturer-specific functionality. Refer to the "7.8.2 Manufacturer Specific Profile Area objects" section on page 85.

In the following pages the objects implemented are listed and described as follows:

Index-subindex Object name
[data types, attribute]

- Index and sub-index are expressed in hexadecimal notation.
- Attribute:

ro = read only access
rw = read and write access

Signed8 / Unsigned8 data type:

Process data bytes							
byte 4							
7	6	5	4	3	2	1	0
MSbit			...				LSbit

Signed16 / Unsigned16 data type:

Process data bytes					
byte 4			byte 5		
7	...	0	15	...	8
LSByte			MSByte		

Signed32 / Unsigned32 data type:

Process data bytes											
byte 4			byte 5			byte 6			byte 7		
7	...	0	15	...	8	23	...	16	31	...	24
LSByte			...			...			MSByte		

NOTE
Always save the new values after setting in order to store them in the non-volatile memory permanently. Use the **Save parameters** function available in the **2200 Control Word** object, see on page 90.
Should the power supply be turned off all data that has not been saved previously will be lost!

7.8.1 Communication Profile Area objects (DS 301)

1000-00 Device type

[Unsigned32, ro]
It contains information about the device type. The object describes the type of device and its functionality.
Default = 0000 012Dh = rotary actuator, DS 301

1008-00 Manufacturer Device Name

[String, ro]
It shows the manufacturer device name, expressed in ASCII code.
Default = 5244362D45432D7878 = "RD6-EC-xx" = RD6 rotary actuator

1009-00 Manufacturer Hardware version

[String, ro]
It shows the hardware version of the device, expressed in ASCII code.
Default = 312E30 = "1.0" = hardware version 1.0

100A-00 Manufacturer Software version

[String, ro]
It shows the software version of the device, expressed in ASCII code.
Default = 312E30 = "1.0" = software version 1.0

1018 Identity Object

[Unsigned8, ro]
This object contains general information about the device. Sub-Index 00 contains the number of entries.
Default = 4

01 Vendor ID

[Unsigned32, ro]
It provides the manufacturer-specific vendor ID. The EtherCAT vendor ID is equal to the CANopen vendor ID.
Default = 0000 012Eh

02 Product code

[Unsigned32, ro]
The manufacturer-specific product code identifies a specific device version.
Default = 0000 2000h = RD6 rotary actuator

03 Revision number

[Unsigned32, ro]
The manufacturer-specific revision number consists of a major revision number and a minor revision number. The major revision number identifies a specific

device behaviour. The minor revision number identifies different versions with the same device behaviour.
Default = device dependent

7	...	0	15	...	8	23	...	16	31	...	24
Minor revision number						Major revision number					
LSB			...			...			MSB		

04 Serial number
[Unsigned32, ro]
It provides the Serial Number of the device. It is 0 if no serial number is provided.
Default = 0000 0000h

1600 Receive PDO Mapping

[Unsigned8, ro]
This object contains the mapping parameters for the PDOs the EtherCAT device is able to receive. Sub-Index 00 contains the number of entries.

01 Mapped Object 001
[Unsigned32, ro]
Sub-Index 01 contains the information of the mapped application object 001. The object describes the content of the PDO by its index, sub-index and length. The length contains the length of the application object in bits. This may be used to verify the mapping.

7 0	15 8	23 16	31 24
Length	Sub-Index	Index	
LSB			MSB

Default = 2200 0010h = **2200 Control Word** object, length 16 bits

02 Mapped Object 002
[Unsigned32, ro]
Sub-Index 02 contains the information of the mapped application object 002. The object describes the content of the PDO by its index, sub-index and length. The length contains the length of the application object in bits. This may be used to verify the mapping.

7 0	15 8	23 16	31 24
Length	Sub-Index	Index	
LSB			MSB

Default = 2201 0020h = **2201-00 Target Position** object, length 32 bits

1A00 Transmit PDO Mapping

[Unsigned8, ro]
This object contains the mapping parameters for the PDOs the EtherCAT device is able to transmit. Sub-Index 00 contains the number of entries.

01 Mapped Object 001

[Unsigned32, ro]
Sub-Index 01 contains the information of the mapped application object 001. The object describes the content of the PDO by its index, sub-index and length. The length contains the length of the application object in bits. This may be used to verify the mapping.

7 0	15 8	23 16	31 24
Length	Sub-Index	Index	
LSB			MSB

Default = 2202 0010h = **2202 Status Word** object, length 16 bits

02 Mapped Object 002

[Unsigned32, ro]
Sub-Index 02 contains the information of the mapped application object 002. The object describes the content of the PDO by its index, sub-index and length. The length contains the length of the application object in bits. This may be used to verify the mapping.

7 0	15 8	23 16	31 24
Length	Sub-Index	Index	
LSB			MSB

Default = 2203 0020h = **2203-00 Real Position** object, length 32 bits

1C00 Sync Manager Communication Type

[Unsigned8, ro]
This object contains the number and type of Sync Manager Communication Types supported by the actuator. Sub-Index 00 specifies the number of Sync Manager channels. Refer also to the "Sync Manager" section on page 71.

01 SM MailBox Receive (SM0)
[Unsigned8, ro]
Used for mailbox write transfers (Master to Slave).
Default = 01

02 SM MailBox Send (SM1)
[Unsigned8, ro]
Used for mailbox read transfers (Slave to Master).
Default = 02

03 SM PDO output (SM2)
[Unsigned8, ro]
It contains the RxPDOs (i.e., Sync Manager 2 holds the Read Process Data).
Default = 03

04 SM PDO input (SM3)
[Unsigned8, ro]
It contains the TxPDOs (i.e., Sync Manager 3 holds the Write Process Data).
Default = 04

1C12-00 Sync Manager RxPDO assign

[Unsigned8, ro]
This object specifies whether the device uses Receive PDO messages. Sub-Index 00 specifies the number of entries, i.e. the number of assigned RxPDOs.

01 SubIndex 001
[Unsigned16, ro]
Default = 1600h = **1600 Receive PDO Mapping** object

1C13-01 Sync Manager TxPDO assign

[Unsigned8, ro]
This object specifies whether the device uses Transmit PDO messages. Sub-Index 00 specifies the number of entries, i.e. the number of assigned TxPDOs.

01 SubIndex 001
[Unsigned16, ro]
Default = 1A00h = **1A00 Transmit PDO Mapping** object

1C32 Output Sync Manager Parameter

[Unsigned8, ro]

The **1C32 Output Sync Manager Parameter** object contains the output synchronization parameters. Some of them are calculated dynamically and depend on the actuator configuration (programmed resolution, counting direction, ...) and the selected synchronization mode (if several modes are supported). Sub-Index 00 contains the number of entries.

01 Sync Type

[Unsigned16, ro]

It allows to select the synchronization mode. Lika actuators only support the FreeRun communication mode. For more information see on page 67.

Default = 0 = FreeRun

02 Cycle time

[Unsigned32, rw]

Application cycle time, i.e. interval between two position samplings (internal timer). The value is expressed in nanoseconds.

Default = 1000000 = 1 ms

03 Shift Time

[Unsigned32, rw]

Interval between the synchronization event and the moment of inputs latching from hardware. This parameter is calculated dynamically and expressed in nanoseconds.

Default = 0

04 Synchronization Types supported

[Unsigned16, ro]

It shows the list of the supported synchronization modes.

Bit 0 = 1 = FreeRun supported

Default = 0001h

05 Minimum Cycle Time

[Unsigned32, ro]

Min. duration of the actuator internal cycle time. This parameter is calculated dynamically and depends on the operating parameters and the position value. It is expressed in nanoseconds.

Default = 1000000 = 1 ms

06 Calc and Copy Time

[Unsigned32, ro]

Time the internal micro-controller (DSP) needs to make calculations on latched optical reading of position and then copy updated data from local memory to ESC memory (Sync Manager) before they are available to EtherCAT. This parameter is calculated dynamically and depends on the operating parameters and the position value. It is expressed in nanoseconds.

Default = 0000 0000

09 Delay Time

[Unsigned32, ro]

Delay time expressed in nanoseconds. It is always set to 0.

Default = 0000 0000

0C Cycle Time Too Small

[Unsigned16, ro]

Error counter for cycle times that are too small.

Default = 0000

1C33 Input Sync Manager Parameter

[Unsigned8, ro]

The **1C33 Input Sync Manager Parameter** object contains the input synchronization parameters. Some of them are calculated dynamically and depend on the actuator configuration (programmed resolution, counting direction, ...) and the selected synchronization mode (if several modes are supported). Sub-Index 00 contains the number of entries.

01 Sync Type

[Unsigned16, ro]

It allows to select the synchronization mode. Lika actuators only support the FreeRun communication mode. For more information see on page 67.

Default = 0 = FreeRun

02 Cycle time

[Unsigned32, rw]

Application cycle time, i.e. interval between two position samplings (internal timer). The value is expressed in nanoseconds.

Default = 1000000 = 1 ms

03 Shift Time

[Unsigned32, rw]

Interval between the synchronization event and the moment of inputs latching from hardware. This parameter is calculated dynamically and expressed in nanoseconds.

Default = 0

04 Synchronization Types supported

[Unsigned16, ro]

It shows the list of the supported synchronization modes.

Bit 0 = 1 = FreeRun supported

Default = 0001h

05 Minimum Cycle Time

[Unsigned32, ro]

Min. duration of the actuator internal cycle time. This parameter is calculated dynamically and depends on the operating parameters and the position value. It is expressed in nanoseconds.

Default = 1000000 = 1 ms

06 Calc and Copy Time

[Unsigned32, ro]

Time the internal micro-controller (DSP) needs to make calculations on latched optical reading of position and then copy updated data from local memory to

ESC memory (Sync Manager) before they are available to EtherCAT. This parameter is calculated dynamically and depends on the operating parameters and the position value. It is expressed in nanoseconds.
Default = 0000 0000

0C Cycle Time Too Small
[Unsigned16, ro]
Error counter for cycle times that are too small.
Default = 0000

NOTE
Always save the new values after setting in order to store them in the non-volatile memory permanently. Use the **Save parameters** function available in the **2200 Control Word** object, see on page 90.
Should the power supply be turned off all data that has not been saved previously will be lost!

7.8.2 Manufacturer Specific Profile Area objects

2101-00 HMS Serial Number

[Unsigned32, ro]
It shows the serial number of the HMS module.
Value = device dependent

2102-00 HMS_FW_Major

[Unsigned8, ro]
The HMS firmware revision number consists of a major revision number and a minor revision number. In this object the major revision number is shown.
Value = device dependent

2103-00 HMS_FW_Minor

[Unsigned8, ro]
The HMS firmware revision number consists of a major revision number and a minor revision number. In this object the minor revision number is shown.
Value = device dependent

2104-00 HMS_FW_Build

[Unsigned8, ro]
It shows the firmware build number of the HMS module.
Value = device dependent

2105-00 Position Offset

[Signed32, ro]
This variable defines the difference between the position value transmitted by the device and the real position: real position – preset. The value is expressed in pulses.

2106-00 Real Speed [rpm]

[Signed32, ro]
Speed of the device expressed in revolutions per minute [rpm], updated at every second.

2107-00 Electronics Temperature [°C]

[Signed8, ro]

This variable shows the temperature of the electronics as detected by internal probes. The value is expressed in °C (Celsius degrees). The minimum detectable temperature is -20°C.

2108-00 Motor Temperature [°C]

[Signed8, ro]

This variable shows the temperature of the motor as detected by internal probes. The value is expressed in °C (Celsius degrees). The minimum detectable temperature is -20°C.

2109-00 Real Current

[Signed32, ro]

This variable shows the value of the current absorbed by the motor (rated current). The value is expressed in mA (milliamperes).

210A-00 Following error [pulse]

[Signed32, ro]

This variable contains the difference between the target position and the current position step by step. If this value is greater than the one set in the **2207-00 Max following error** parameter, then the **Following error** alarm is triggered and the unit stops. The value is expressed in pulses.

210B-00 Pos. Limit Switch [pulse]

[Signed32, ro]

This is the **SW limit switch +** value (maximum positive limit) calculated according to values set next to the **2211-00 Preset** and **220C-00 Max delta pos** objects. When the maximum forward limit is reached, the condition is signalled through the **SW limit switch +** status bit 3 of the **2202 Status Word**.

SW limit switch + = **2211-00 Preset** + **220C-00 Max delta pos**.

The value is expressed in pulses.

Refer also to the EXAMPLE 1 in the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.

Default = 134 213 631

210C-00 Neg. Limit Switch [pulse]

[Signed32, ro]

This is the **SW limit switch -** value (maximum negative limit) calculated according to values set next to the 2211-00 Preset and 220D-00 Max delta neg objects. When the maximum backward limit is reached, the condition is signalled through the **SW limit switch -** status bit 4 of the 2202 Status Word.

SW limit switch - = 2211-00 Preset – 220D-00 Max delta neg.

The value is expressed in pulses.

Refer also to the EXAMPLE 1 in the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.

Default = - 134 213 631

210D Parameter Error List

[Unsigned32, ro]

The operator has set invalid data and the **Machine data not valid** alarm has been triggered. This variable is meant to show the list of the wrong parameters, according to the information in the following table.

Please note that the normal work status can be restored only after having set proper values.

Bit	Parameter
0	Not used
1	2204-00 Distance per revolution
2	220A-00 Acceleration
3	220B-00 Deceleration
4	220C-00 Max delta pos
5	220D-00 Max delta neg
6	220E-00 Jog speed
7	220F-00 Work speed
8	2210-00 Count direction
9	2211-00 Preset
10	2212-00 Step jog
11	2208-00 Proportional gain
12	2209-00 Integral gain
13	2206-00 Settling time
14	2207-00 Max following error

15	Not used

210E Alarms List

[Unsigned16, ro]
This variable is meant to show the alarms currently active in the device.
Structure of the alarms byte:

byte	LSB			MSB		
bit	7	...	0	15	...	8
	msb		lsb	msb		lsb

The available alarm error codes are listed hereafter:

Byte 0

Machine data not valid
bit 0 — One or more parameters are not valid, set proper values to restore the normal work condition. See the list of the wrong parameters in the **210D Parameter Error List** object.

Flash memory error
bit 1 — Internal error, it cannot be restored.

Counting error
bit 2 — For safety reasons, both the absolute position and the incremental position of the integral encoder are read and saved to two separate registers. If any difference between the values in the registers is found the error is signalled.

Following error
bit 3 — The difference between the real position and the theoretical position is greater than the value set in the **2207-00 Max following error** object; we suggest reducing the work speed.

Axis not synchronized
bit 4 — Internal error, it cannot be restored.

Target not valid
bit 5 — The set target position is over the maximum travel limits.

Emergency
bit 6 — Bit 7 **Emergency** in **2200 Control Word** has been forced to low value (0); or alarms are active in the unit.

Overcurrent
bit 7 — Motor overcurrent.

Byte 1

Electronics Overtemperature

bit 8 — The temperature of the MOSFETs detected by an internal probe is exceeding the maximum ratings (see **2107-00 Electronics Temperature [°C]** on page 86). Please wait some minutes for the actuator to cool down. Ensure that the operating temperature is within the allowed range.

Motor Overtemperature

bit 9 — The temperature of the motor detected by an internal probe is exceeding the maximum ratings (see **2108-00 Motor Temperature [°C]** on page 86). Please wait some minutes for the actuator to cool down. Ensure that the operating temperature is within the allowed range.

Undervoltage

bit 10 — The power supply voltage is under the minimum ratings allowed. Please ensure that the power supply voltage is within the allowed range.

Watch dog

bit 11 — The watchdog is a safety feature that switches off the outputs after the set time e.g. in the event of an interruption of the process data traffic, depending on the device and settings, e.g. in OFF state. When the Watchdog safety feature is enabled in the configuration tool for the Master, if no EtherCAT process data communication takes place with the actuator or no PDI communication with the EtherCAT Slave Controller (ESC) takes place for longer than the set time (100 milliseconds), the system forces an alarm condition (the **Watch dog** alarm bit is activated). When the SyncManager watchdog is triggered, the outputs are set to FALSE while the OP state of the terminal is unaffected. The PDI watchdog can be used to monitor the communication for failure. For more information please refer to the "7.6.10 Watchdog" section on page 72.

bits 12 and 13 — Not used.

Hall sequence

bit 14 — An error has been detected in the Hall sensors commutation sequence.

Overvoltage

bit 15 — The power supply voltage is over the maximum ratings allowed. Please ensure that the power supply voltage is within the allowed range.

If the alarm is triggered during the braking operation, please consider the counter-electromotive force (back EMF). To prevent such situation from arising, decrease the deceleration ramp or evaluate attentively the characteristics of the 24V power supply pack (capacitor module).

To reset a faulty condition use the **Alarm reset** command, bit 3 in the **2200 Control Word**. In a normal work condition the **Alarm reset** bit is set to "0". Setting the bit to "1" causes the normal work status of the device to be restored. The normal work status is resumed by switching this bit from "0" to "1". This command resets the alarm but only if the fault condition has ceased.

Please note that should the alarm be caused by wrong parameter values (see **Machine data not valid** and **210D Parameter Error List**), the normal work status can be restored only after having set proper values. The **Flash memory error** alarm cannot be reset.

2200 Control Word

[Unsigned16, rw]
This variable contains the commands to be sent in real time to the Slave in order to manage it. It is mapped in the **1600 Receive PDO Mapping** object that contains the mapping parameters for the PDOs the EtherCAT device is able to receive, see on page 79. It is updated every 1 msec.
Byte structure of the **2200 Control Word** object:

byte	LSB			MSB		
bit	7	...	0	15	...	8
	msb		lsb	msb		lsb

Byte 0

Jog +

bit 0

If the bit 4 **Incremental jog** = 0, as long as **Jog +** = 1, the Slave moves toward the positive direction; otherwise if the bit 4 **Incremental jog** = 1, the activation of this bit causes a single step toward the positive direction having the length, expressed in pulses, set next to the **2212-00 Step jog** entry to be executed at rising edge; then the actuator stops and waits for another command. Velocity, acceleration and deceleration are performed according to the values set next to the **220E-00 Jog speed**, **220A-00 Acceleration** and

220B-00 Deceleration objects respectively. For a detailed description of the jog control see on page 43.

Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.

Jog -
bit 1

If the bit 4 **Incremental jog** = 0, as long as **Jog -** = 1, the Slave moves toward the negative direction; otherwise if the bit 4 **Incremental jog** = 1, the activation of this bit causes a single step toward the negative direction having the length, expressed in pulses, set next to the 2212-00 Step jog entry to be executed at rising edge; then the actuator stops and waits for another command. Velocity, acceleration and deceleration are performed according to the value set next to the 220E-00 Jog speed, 220A-00 Acceleration and 220B-00 Deceleration objects respectively. For a detailed description of the jog control see on page 43.

Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.

Stop
bit 2

If set to "=1" the Slave is allowed to execute the movements as commanded. If, while the unit is running, this bit switches to "=0" then the Slave must stop executing the deceleration procedure set in 220B-00 Deceleration. For an immediate halt in the movement, use the bit 7 **Emergency**.

Alarm reset
bit 3

This command is used to reset an alarm condition of the Slave but only if the fault condition has ceased. In a normal work condition this bit is set to "0". Setting this bit to "1" causes the normal work status of the device to be restored. The normal work status is resumed by switching this bit from "0" to "1".

Please note that should the alarm be caused by wrong parameter values (see Machine data not valid and 210D Parameter Error List), the normal work status can be

restored only after having set proper values. The **Flash memory error** alarm cannot be reset.

Incremental jog

bit 4 — If set to "=0", the activation of the bits **Jog +** and **Jog -** causes the Slave to move as long as **Jog +** / **Jog -** = 1. Setting this bit to 1 the incremental jog function is enabled, that is: the activation of the bits **Jog +** and **Jog -** causes a single step toward the positive or negative direction having the length, expressed in pulses, set next to the **2212-00 Step jog** object to be executed at rising edge; then the actuator stops and waits for another command.

bit 5 — Not used.

Start

bit 6 — When it is set to "=1" the device moves in order to reach the set target position (see **2201-00 Target Position** on page 94). For a complete description of the position control see on page 44.

Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.

Emergency

bit 7 — This bit has to be normally high ("=1") otherwise it will cause the device to stop immediately. For a normal stop (not immediate) respecting the set deceleration see above the bit 2 **Stop**. At power-on it is forced low (="0") for safety reasons. Switch it high ("=1") to resume normal operation.

Byte 1

bit 8 — Not used.

Save parameters

bit 9 — Data is saved on non-volatile memory at each rising edge of the bit; in other words, save is performed each time this bit is switched from logic level low ("0") to logic level high ("1"). Always save the new values after setting in order to store them in the non-volatile memory permanently. Should the power supply be turned off all data that has not been saved previously will be lost!

Load default parameters

bit 10

The default parameters (they are set at the factory by Lika Electronic engineers to allow the operator to run the device for standard operation in a safe mode) are restored at each rising edge of the bit; in other words, the default parameters loading operation is performed each time this bit is switched from logic level low ("0") to logic level high ("1"). The complete list of machine data and relevant default parameters preset by Lika Electronic engineers is available on page 170.

Always save the new values after setting in order to store them in the non-volatile memory permanently. Should the power supply be turned off all data that has not been saved previously will be lost!

WARNING

The unit has been adjusted by performing a full-load mechanical running test; thence default values which has been set refer to a device running in such condition. Furthermore they are intended to ensure a standard and safe operation which not necessarily results in a smooth running and an optimum performance. Thus to suit the specific application requirements it may be advisable and even necessary to enter new parameters instead of the factory default settings; in particular it may be necessary to change velocity, acceleration, deceleration and gain values.

Setting the preset

bit 11

It sets the current position to the value set next to the 2211-00 Preset object. The operation is performed at each rising edge of the bit, i.e. each time this bit is switched from logic level low ("0") to logic level high ("1"). We suggest activating the preset when the actuator is in stop. For more information refer to page 102.

Release axis torque

bit 12

When the axis has reached the commanded position, it maintains the torque.

If set to "=0", when the axis is in position, the PWM is kept active.

If set to "=1", when the axis is in position, the PWM is deactivated (the torque is released).

bits 13 ... 15

Not used.

2201-00 Target Position

[Signed32, rw]

This object is mapped in the 1600 Receive PDO Mapping object that contains the mapping parameters for the PDOs the EtherCAT device is able to receive, see on page 79. It is updated every 1 msec.

It sets the position to be reached, otherwise referred to as commanded position. When the **Start** command is sent while the **Stop** and **Emergency** bits are "=1" and the alarm condition is off, the device moves in order to reach the target position set next to this object.

As soon as the axis is within the tolerance window limits set next to the 2205-00 Position tolerance object, the bit 8 **Target position reached** in the 2201-00 Target Position goes high ("=1"). When the position is within the tolerance window limits set next to the 2205-00 Position tolerance object, after the delay set next to the 2206-00 Settling time object, the bit 0 **Axis in position** in the 2202 Status Word goes high ("=1").

For more information refer also to the "Positioning: position and speed control" section on page 44.

Default = 0 (min. = 0, max. = within 210B-00 Pos. Limit Switch [pulse] / 210C-00 Neg. Limit Switch [pulse])

NOTE

Position override function

It is possible to change the target position value even on the fly, while the device is still reaching a previously commanded target position and without sending a new **Start** command. To do this, just set a new target value in the 2201-00 Target Position object. See also on page 44.

NOTE

Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.

When the Watchdog function is enabled, should the device be disconnected from the EtherCAT network while it is moving (for instance because of a broken cable or a faulty wiring), the device stops moving immediately and activates the Watch dog alarm bit.

2202 Status Word

[Unsigned16, ro]

This variable provides information about the current state of the device. It is mapped in the **1A00 Transmit PDO Mapping** object that contains the mapping parameters for the PDOs the EtherCAT device is able to transmit, see on page 80. It is updated every 1 msec.

Byte structure of the **2202 Status Word** object:

byte	LSB			MSB		
bit	7	...	0	15	...	8
	msb		lsb	msb		lsb

Byte 0

Axis in position

bit 0 — The value is "=1" when the device reaches and keeps the commanded position (**2201-00 Target Position**) for the time set next to the **2206-00 Settling time** object. It is kept active until the position error is lower than **2205-00 Position tolerance**. For further information please refer to the "Positioning: position and speed control" section on page 44.

bit 1 — Not used.

Axis enabled

bit 2 — It shows the enabling status of the motor. This bit is "=1" when the motor is enabled, that is: PWM is active and the axis is under closed-loop control (while reaching a target position or using a jog, for instance). It is "=0" when the motor is disabled, that is when the controller is off after a positioning or jog movement or because of an alarm condition.

SW limit switch +

bit 3 — The value is "=1" should it happen that the device reaches the maximum positive limit (positive limit switch). For more information see the **220C-00 Max delta pos** object.

SW limit switch -

bit 4 — The value is "=1" should it happen that the device reaches the maximum negative limit (negative limit switch). For more information see the **220D-00 Max delta neg** object.

Alarm

bit 5 — The value is "=1" when an alarm occurs, see details in the **210E Alarms List** variable.

Axis running

bit 6 — The value is "=0" when the device is not moving.
The value is "=1" while the device is moving.

Executing a command

bit 7 — The value is "=0" when the controller is not executing any command.
The value is "=1" while the controller is executing a command.

Byte 1

Target position reached

bit 8 — The value is "=1" when the device reaches the target position set next to the **2201-00 Target Position** object (it is within the limits set next to the **2205-00 Position tolerance**). The bit is kept active until a new **2201-00 Target Position** value or the **Alarm reset** command are sent. For more information refer also to the "Positioning: position and speed control" section on page 44.

bits 9 ... 11 — Not used.

PWM saturation

bit 12 — The current supplied for controlling the motor phases has reached the saturation point and cannot be increased further. The motor operation is affected by excessive dynamics or something is jamming the movement.

bits 13 ... 15 — Not used.

2203-00 Real Position

[Signed32, ro]
Current position of the device expressed in pulses. This object is mapped in the **1A00 Transmit PDO Mapping** object that contains the mapping parameters for the PDOs the EtherCAT device is able to transmit, see on page 80. It is updated every 1 msec.

2204-00 Distance per revolution

[Unsigned16, rw]
This parameter sets the number of pulses per each complete revolution of the shaft. It is useful to relate the revolution of the shaft and a linear measurement. For example: the unit is joined to a warm screw having 5 mm (0.197") pitch; by setting **2204-00 Distance per revolution** = 500, at each shaft revolution the system performs a 5 mm (0.197") pitch with one-hundredth of a millimetre resolution.
Default = 4096 (min. = 1, max. = 4096)

WARNING
After having changed this object you must then set new values also next to the **2211-00 Preset** object. For a detailed explanation see on page 45 and the relevant objects.
Please note that the objects listed hereafter are closely related to the **2204-00 Distance per revolution** object; hence when you change the value in **2204-00 Distance per revolution** also the value in each of them necessarily changes. They are: **2205-00 Position tolerance**, **2207-00 Max following error**, **220C-00 Max delta pos**, **220D-00 Max delta neg**, **2201-00 Target Position**, **2203-00 Real Position** and **210A-00 Following error [pulse]**.

NOTE
If **2204-00 Distance per revolution** is not a power of 2 (2, 4, ..., 2048, 4096), at position control a positioning error could occur having a value equal to one pulse.

2205-00 Position tolerance

[Unsigned16, rw]
This object defines the tolerance window limits for the **2201-00 Target Position** value. As soon as the axis is within the tolerance window limits, the bit 8 **Target position reached** in the **2202 Status Word** goes high ("=1"). When the axis is within the tolerance window limits for the time set in the **2206-00 Settling time** object, the bit 0 **Axis in position** in the **2202 Status Word** goes high ("=1"). The parameter is expressed in pulses. See also the "Positioning: position and speed control" section on page 44.
Default = 1 (min. = 0, max. = 65535)

2206-00 Settling time

[Unsigned16, rw]
It represents the time for which the axis has to be within the tolerance window limits set in the **2205-00 Position tolerance** object before the state is signalled through the **Axis in position** status bit of the **2202 Status Word**. The parameter is expressed in milliseconds. See also the "Positioning: position and speed control" section on page 44.
Default = 0 (min. = 0, max. = 10000)

2207-00 Max following error

[Unsigned32, rw]
This object defines the maximum allowable difference between the real position and the theoretical position of the device. If the device detects a value higher than the one set in this parameter, the **Following error** alarm is triggered and the unit stops. The parameter is expressed in pulses.
Default = 50000 (min. = 0, max. = 1000000)

2208-00 Proportional gain

[Unsigned16, rw]
This parameter contains the proportional gain used by the PI controller for the position loop. The value has been optimized by Lika Electronic according to the technical characteristics of the device.
Default = 80 (min. = 0, max. = 1000)

2209-00 Integral gain

[Unsigned16, rw]
This parameter contains the integral gain used by the PI controller for the position loop. The value has been optimized by Lika Electronic according to the technical characteristics of the device.
Default = 10 (min. = 0, max. = 1000)

220A-00 Acceleration

[Unsigned16, rw]
This parameter defines the acceleration value that has to be used by the device when reaching either the **220E-00 Jog speed** or the **220F-00 Work speed**. The parameter is expressed in revolutions per second^2 [rev/s^2]. See also the "6.2 Movements: jog and positioning" section on page 43.
Default = 10 (min. = 1, max. = 500)

220B-00 Deceleration

[Unsigned16, rw]
This parameter defines the deceleration value that has to be used by the device when stopping. The parameter is expressed in revolutions per second^2 [rev/s^2].
See also the "6.2 Movements: jog and positioning" section on page 43.
Default = 10 (min. = 1, max. = 500)

220C-00 Max delta pos

[Unsigned32, rw]
This value is used to calculate the maximum forward (positive) limit the device is allowed to reach starting from the preset value. As soon as the maximum forward limit is reached, the condition is signalled through the **SW limit switch +** status bit of the **2202 Status Word** (the bit is forced high). The parameter is expressed in pulses.
SW limit switch + = **2211-00 Preset** + **220C-00 Max delta pos**. The maximum positive limit can be read next to the **210B-00 Pos. Limit Switch [pulse]** object.
For further information please refer to the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.
Default = 134 213 631 (min. = 0, max. = 134 213 631)

WARNING
Please mind the maximum acceptable value for this item depends on the set scaling.

EXAMPLE
When **2204-00 Distance per revolution** = 4,096 and **2211-00 Preset** = 0, the maximum acceptable value for **220C-00 Max delta pos** is:
(4,096 steps per revolution * 32,768 revolutions) – 1 step – 4,096 steps (i.e. 1 revolution for safety reasons) = 134 213 631 (see the default value)
When **2204-00 Distance per revolution** = 1,024 and **2211-00 Preset** = 0, the maximum acceptable value for **220C-00 Max delta pos** is:
(1,024 steps per revolution * 32,768 revolutions) – 1 step – 1,024 steps (i.e. 1 revolution for safety reasons) = 33 553 407
When **2204-00 Distance per revolution** = 256 and **2211-00 Preset** = 0, the maximum acceptable value for **220C-00 Max delta pos** is:
(256 steps per revolution * 32,768 revolutions) – 1 step – 256 steps (i.e. 1 revolution for safety reasons) = 8 388 351
See further examples in the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.

WARNING

Every time **2204-00 Distance per revolution** and **2211-00 Preset** objects are changed, **220C-00 Max delta pos** and **220D-00 Max delta neg** values have to be checked carefully. Each time you change the value in **2204-00 Distance per revolution**, then you must update the value in **2211-00 Preset** in order to define the zero of the shaft as the system reference has now changed.
After having changed the parameter in **2211-00 Preset** it is not necessary to set new values for travel limits as the Preset function calculates them automatically and initializes again the positive and negative limits according to the values set in **220C-00 Max delta pos** and **220D-00 Max delta neg** objects. For a detailed explanation see on page 45.

220D-00 Max delta neg

[Unsigned32, rw]
This value is used to calculate the maximum backward (negative) limit the device is allowed to reach starting from the preset value. As soon as the maximum backward limit is reached, the condition is signalled through the **SW limit switch -** status bit of the **2202 Status Word** (the bit is forced high). The parameter is expressed in pulses.
SW limit switch - = **2211-00 Preset** - **220D-00 Max delta neg**. The maximum negative limit can be read next to the **210C-00 Neg. Limit Switch [pulse]** object.
For further information please refer to the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.
Default = 134 213 631 (min. = 0, max. = 134 213 631)

WARNING

Please mind the maximum acceptable value for this item depends on the set scaling.

EXAMPLE

When **2204-00 Distance per revolution** = 4,096 and **2211-00 Preset** = 0, the maximum acceptable value for **220D-00 Max delta neg** is:
(4,096 steps per revolution * 32,768 revolutions) - 1 step – 4,096 steps (i.e. 1 revolution for safety reasons) = 134 213 631 (see the default value)
When **2204-00 Distance per revolution** = 1,024 and **2211-00 Preset** = 0, the maximum acceptable value for **220D-00 Max delta neg** is:
(1,024 steps per revolution * 32,768 revolutions) - 1 step – 1,024 steps (i.e. 1 revolution for safety reasons) = 33 553 407

When 2204-00 Distance per revolution = 256 and 2211-00 Preset = 0, the maximum acceptable value for 220D-00 Max delta neg is:
(256 steps per revolution * 32,768 revolutions) – 1 step – 256 steps (i.e. 1 revolution for safety reasons) = 8 388 351
See further examples in the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.

WARNING
Every time 2204-00 Distance per revolution and 2211-00 Preset objects are changed, 220C-00 Max delta pos and 220D-00 Max delta neg values have to be checked carefully. Each time you change the value in 2204-00 Distance per revolution, then you must update the value in 2211-00 Preset in order to define the zero of the shaft as the system reference has now changed.
After having changed the parameter in 2211-00 Preset it is not necessary to set new values for travel limits as the Preset function calculates them automatically and initializes again the positive and negative limits according to the values set in 220C-00 Max delta pos and 220D-00 Max delta neg objects. For a detailed explanation see on page 45.

220E-00 Jog speed

[Unsigned16, rw]
This object contains the maximum speed the device is allowed to reach when using the **Jog +** and **Jog -** functions (see the 2200 Control Word object). The parameter is expressed in revolutions per minute (rpm). See also the "Jog: speed control" section on page 43.
Default = 2000 (min. = 1, max. = 3000)

220F-00 Work speed

[Unsigned16, rw]
This object contains the maximum speed the device is allowed to reach in automatic work mode (movements are controlled using the **Start** and **Stop** commands -see the 2200 Control Word object- and are performed in order to reach the position set in 2201-00 Target Position). The parameter is expressed in revolutions per minute (rpm). See also the "Positioning: position and speed control" section on page 44.
Default = 2000 (min. = 1, max. = 3000)

2210-00 Count direction

[Unsigned16, rw]
It sets whether the position value output by the device increases (count up information) when the shaft rotates clockwise (0) or counter-clockwise (1). Clockwise and counter-clockwise rotations are viewed from the shaft side.
0 = count up information with clockwise rotation (default)
1 = count up information with counter-clockwise rotation

WARNING
Changing this value causes also the position calculated by the controller to be necessarily affected. Therefore it is compulsory to set a new value in the **2211-00 Preset** object and then check the values set next to the **220C-00 Max delta pos** and **220D-00 Max delta neg** objects.

2211-00 Preset

[Signed32, rw]
Use this object to set the Preset value. The Preset function is meant to assign a desired value to a physical position of the axis. The chosen physical position will get the value set next to this item and all the previous and the following positions will get a value according to it. The preset value will be set for the position of the axis in the moment when the value is entered. The preset value is activated when the bit 11 **Setting the preset** in the **2200 Control Word** object is switched from logic level low ("0") to logic level high ("1").
Default = 0 (min. = -268 435 456, max. = 268 435 456)

NOTE
We suggest activating the preset when the actuator is in stop. See the **Setting the preset** command on page 93.

WARNING
A new value has to be set in the **2211-00 Preset** object every time the **2204-00 Distance per revolution** value is changed. After having entered a new value in **2211-00 Preset** it is not necessary to set new values for travel limits as the Preset function calculates them automatically and initializes again the positive and negative limits according to the values set in the **220C-00 Max delta pos** and **220D-00 Max delta neg** items. For a detailed explanation see on page 45.

2212-00 Step jog

[Unsigned16, rw]

If the incremental jog function is enabled (bit 4 **Incremental jog** in the **2200 Control Word** = 1), the activation of the bits **Jog +** and **Jog -** causes a single step toward the positive or negative direction having the length, expressed in pulses, set next to this item to be executed at rising edge; then the actuator stops and waits for another command.

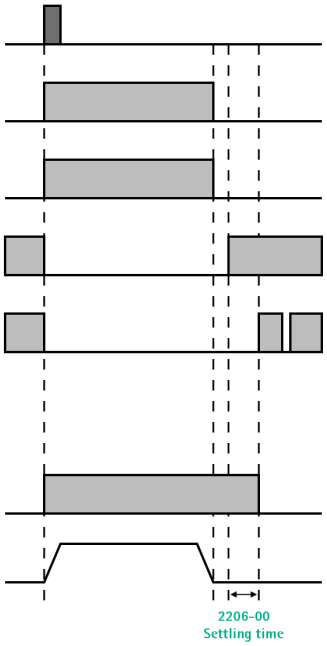
Default = 1000 (min. = 1, max. = 10000).

NOTE

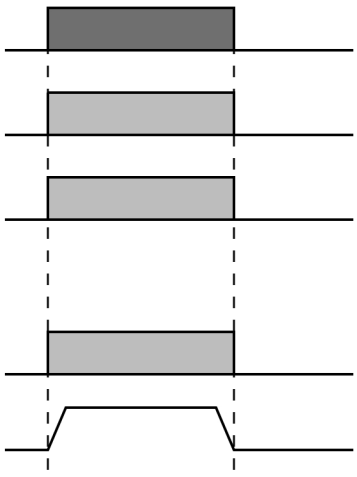
Always save the new values after setting in order to store them in the non-volatile memory permanently. Use the **Save parameters** function available in the **2200 Control Word** object, see on page 90.

Should the power supply be turned off all data that has not been saved previously will be lost!

EXAMPLE 1

Positioning	RD6
Start (bit 6 - 2200 Control Word) **Executing a command** (bit 7 - 2202 Status Word) **Axis running** (bit 6 - 2202 Status Word) **Target position reached** (bit 8 - 2202 Status Word) **Axis in position** (bit 0 - 2202 Status Word) **Axis enabled** (bit 2 - 2202 Status Word: PWM is active) Theoretical velocity	

EXAMPLE 2

Jog +	RD6
Jog + (bit 0 - 2200 Control Word) **Executing a command** (bit 7 - 2202 Status Word) **Axis running** (bit 6 - 2202 Status Word) **Axis enabled** (bit 2 - 2202 Status Word: PWM is active) Theoretical velocity	

7.9 SDO Abort codes

SDO transfer could be unsuccessful; causes of error are listed and described in the SDO Abort Codes (see ETG1000.6 "EtherCAT Specification – Part 6. Application Layer protocol Specification", par. 5.6.2.7.2, table 40).

7.10 Emergency Error Codes

Emergency Service is used by server for transmitting diagnostic messages to the client using MailBox; Error Codes are listed and described in the ETG1000.6 "EtherCAT Specification – Part 6. Application Layer protocol Specification", par. 5.6.4.2, table 50.

Error Code		Error Register	Diagnostic Data				
Byte (0)	Byte (1)	Byte (2)	Byte (3)	Byte (4)	Byte (5)	Byte (6)	Byte (7)

Error Code — State Transition Errors of state machine:
(for detailed description see ETG1000.6 par. 5.6.4.3)
A000hex: transition error from PRE-OPERATIONAL to SAFE-OPERATIONAL
A001hex: transition error from SAFE-OPERATIONAL to OPERATIONAL

Error Register — EtherCAT state machine current state (ESM)

Diagnostic Data — information about possible error causes (see ETG1000.6 par. 5.6.4.3.2-5).

7.11 AL Status Error Codes

If the state transition requested by the Master through the "AL Control Register" is unsuccessful, Slave sets to 1 the "Error Indicator Bit" in "AL Status Register" and writes the cause of the error in "AL Status Code Register".
Values and descriptions of "AL Status Code" are available in ETG1000.6 "EtherCAT Specification – Part 6. Application Layer protocol Specification", par.5.3.2, Table 11.

8 Modbus® interface

Lika DRIVECOD positioning units are Slave devices and implement the Modbus application protocol (level 7 of the OSI model) and the "Modbus over Serial Line" protocol (levels 1 & 2 of the OSI model).
For any further information or omitted specifications please refer to the "Modbus Application Protocol Specification V1.1b" and "Modbus over Serial Line. Specification and Implementation Guide V1.02" available at www.modbus.org.

8.1 Configuring the device using Lika's setting up software

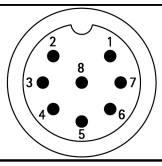
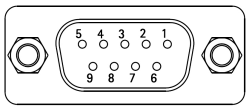
RD6 DRIVECOD positioning units can be equipped with several communication interfaces such as EtherNet/IP, EtherCAT, POWERLINK, MODBUS RTU, Profibus-DP, CANopen DS 301 etc. All versions except the MODBUS RTU one are equipped with an RS-232 service serial port in compliance with the MODBUS protocol. It can be used to configure the actuator. For this purpose all versions are supplied with a software expressly developed and released by Lika Electronic in order to allow an easy set up of the device. The program allows the operator to set the working parameters of the device; control manually some movements and functions; and monitor whether the device is running properly. The program is supplied for free and can be installed in any PC fitted with a Windows operating system (Windows XP or later). The executable file to launch the program is **ROTADRIVE INTERFACE.EXE** and is available in the enclosed documentation or at the address **www.lika.biz > ROTARY ACTUATORS > ROTARY ACTUATORS/POSITIONING UNITS (DRIVECOD)**. The program is designed to be installed simply by copying the executable file to the desired location and there is **no installation** process. To launch it just double-click the file icon. To close the program press the **CLOSE** button in the title bar.

NOTE
Before starting the program, connect the device to the personal computer through an RS-232 serial port. The serial interface of the DRIVECOD unit is an RS-232 type connector. Should the personal computer not be equipped with an RS-232 serial port, you must install a USB / RS-232 converter, easily available in the market. For any information on the connection scheme and the cable pinout refer to the instruction sheet provided with the converter.

On the DRIVECOD side the cable must be connected to the M12 8-pin male connector service serial port. See the "Electrical connections" section on page 31.

A connection assembly fitted with M12 8-pin / USB connectors is available on request; please contact Lika Electronic Technical Assistance & After Sale Service and quote the following items: **IF92 converter** + **EC-RD6-M12M8 connection cable**.

NOTE
If you use the IF92 converter + connection cable, you are required to install the drivers of the USB Serial Converter and the USB Serial Port first. The drivers are available in the Software folder of the actuator and downloadable from Lika's web site.

			
Function	**RS-232 M12 8-pin male connector**	**9-pin D-SUB female connector**	**Function**
TD	6	2	RD
RD	7	3	TD
0Vdc	8	5	0Vdc

Always make sure that the RD of the DRIVECOD unit is cross-wired to the TD of the PC and the TD of the PC is cross-wired to the RD of the DRIVECOD unit.

Please note that the configuration parameters of the MODBUS service serial port have fixed values, so the user cannot change them.
They are:

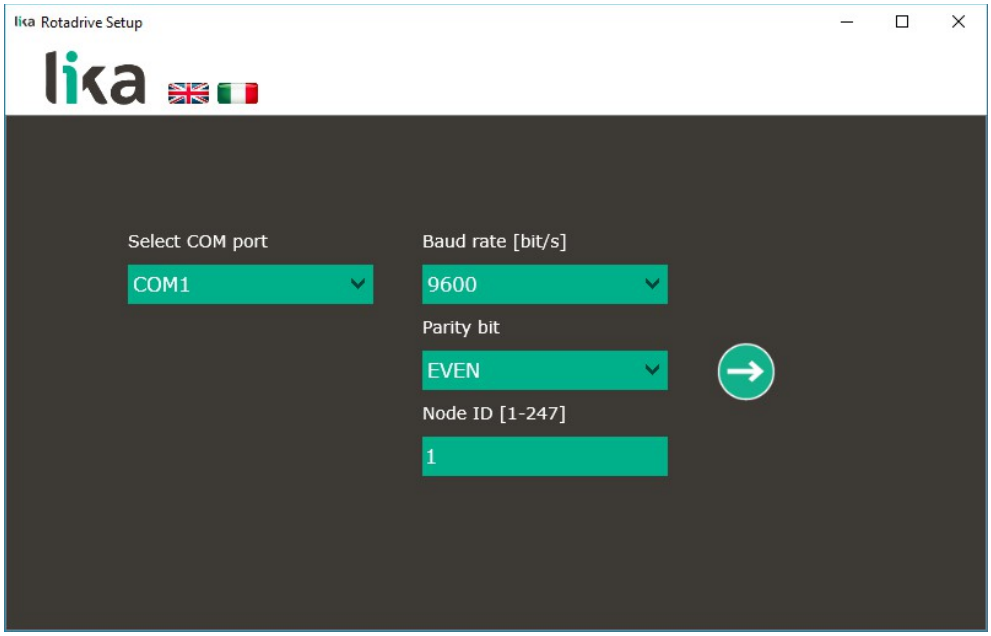
RS-232 Modbus Serial port settings	**Default value**
Baud rate	9600
Byte size	8
Parity	Even
Stop bits	1

The MODBUS address is 1. It cannot be changed. See the "8.2 "Serial configuration" page" section hereafter.

NOTE
Please note that only the L5 LED (controller power supply) and the L8 LED (motor enabling) operation is available when the MODBUS interface is active.

8.2 "Serial configuration" page

When you start the program, the **Serial configuration** page appears on the screen.



First of all this page allows the operator to choose the language used to display texts and items in the user interface. Click on the **Italian flag** icon to choose the Italian language; click on the **UK flag** icon to choose the English language. The default language is according to the language of the operating system.

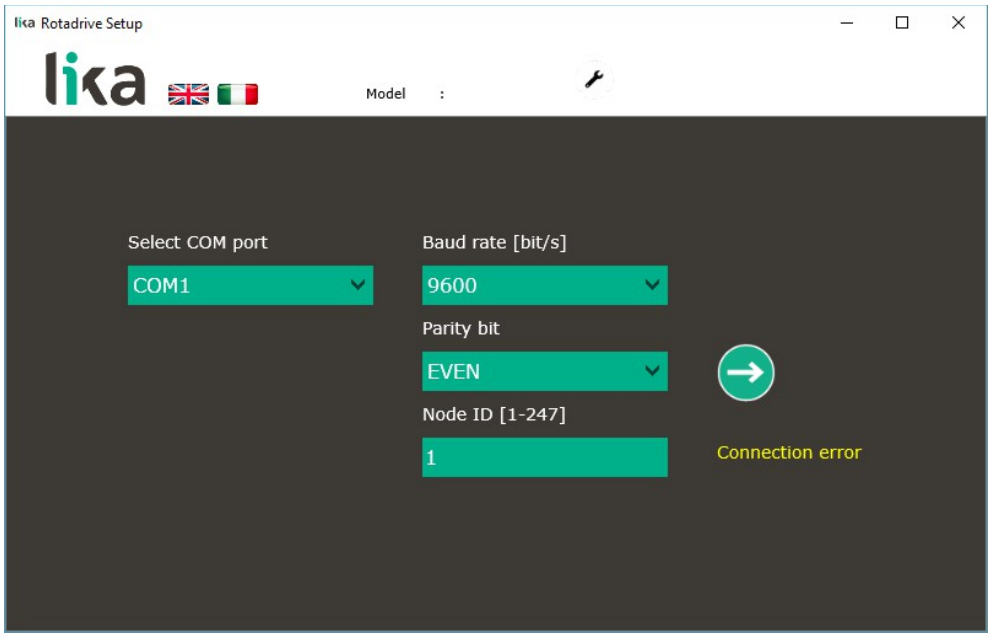
Furthermore, by clicking on Lika's logo button on the top left you enter Lika's web site **www.lika.biz**.

The **Serial configuration** page allows you to choose the serial port of the personal computer the RD6 unit is connected to (**Select COM port** drop-down box) and then set the configuration parameters. Serial port settings in the personal computer must compulsorily match those in the connected Lika device.

For serial port settings see the previous section.

Lastly set the node address of the device the personal computer is connected to through the **Node ID [1-247]** drop-down box (for all RD6-Modbus positioning units = **1**). See on page 35.

Now you are ready to establish the connection to the Slave: press the **CONFIRMATION** button on the right to start searching for the connected device.



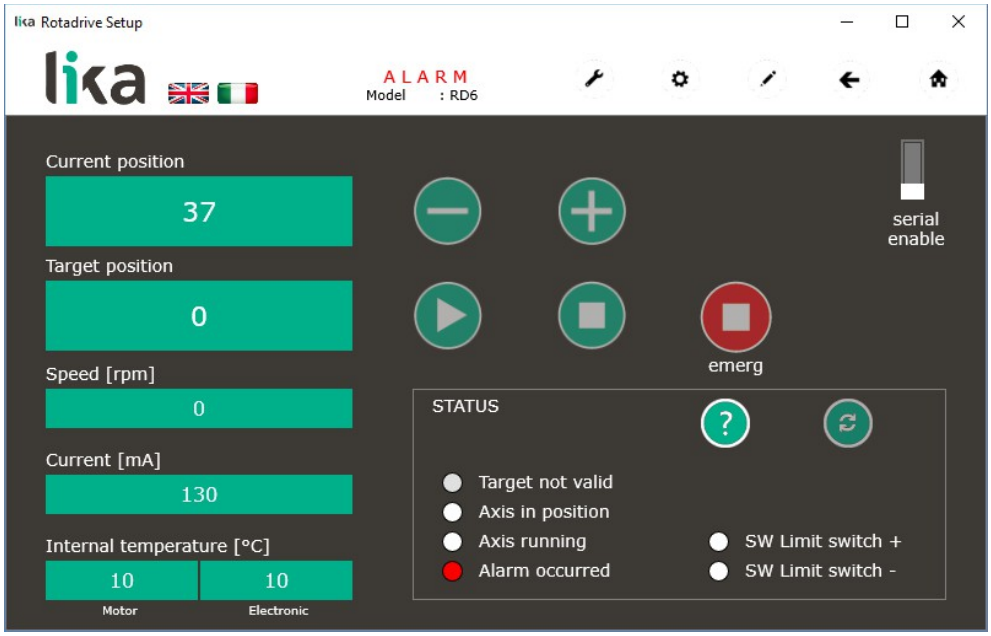
If the connection cannot be established (for instance, because of a wrong COM port setting or a wrong Node ID), the messages **Connection error**, **Device not responding**, **Error opening COM**, **Select COM port** or **Node ID error** may appear under the confirmation button. Please check the settings and try the connection again.

NOTE

Please note that the **FIRMWARE** button appears in the white bar at the top of the page (toolbar). It allows to enter the **Programming firmware** page. For complete information refer to the "8.7 "Programming firmware" page" section on page 116.

8.3 "Main" page

As soon as you press the **CONFIRMATION** button in the **Serial configuration** page, if the connection is established properly you enter the **Main** page.



In the white bar at the top of the page (toolbar) the DRIVECOD model you are connected to appears next to the **Model** label.
The ALARM warning message blinks as the unit is in emergency condition.
In the same bar the following buttons become available.

FIRMWARE

The **FIRMWARE** button allows to enter the **Programming firmware** page. This page allows the operator to upgrade the firmware of the DRIVECOD unit by downloading upgrading data to the flash memory. For complete information refer to the "8.7 "Programming firmware" page" section on page 116.

SETUP

The **SETUP** button allows to enter the **Parameters** page. In this page the list of the parameters available to set the RD6 positioning units (machine data) is displayed. For complete information refer to the "8.8 "Parameters" page" section on page 119.

SCHEDULE

The **SCHEDULE** button allows to enter the **Program** page. The functions available in the **Program** page allow the operator to create and save work programs in order to test the operation of the RD6 unit. For complete information refer to the "8.9 "Program" page" section on page 122.

ARROW BACK

The **ARROW BACK** button allows to go back to the page you looked through previously.

HOME

HOME button. When the **Main** page is displayed, press the **HOME** button to enter the **Serial configuration** page. When any other page is displayed, press the **HOME** button to enter the **Main** page.

In this page some further information on the position and states of the DRIVECOD unit appears.
The following items are available on the left.

Current position
See the Current position [0x02-0x03] registers on page 158.

Target position
See the Target position [0x2B-0x2C] registers on page 152. Set the position you need the unit to reach and then press the **ENTER** key in the keyboard to confirm it. As soon as you press the **START** button the device starts moving in order to reach the commanded position set next to this **Target position** item, then it stops and activates the **Axis in position** and **Target position reached** status bits (see the "8.5 "Status" box" section on page 114). For detailed information on creating a complete cycle of movements and test the operation of the RD6 actuator (speed, acceleration, deceleration, etc.) refer to the "8.9 "Program" page" section on page 122.

Speed [rpm]
See the Current velocity [0x04] register on page 158.

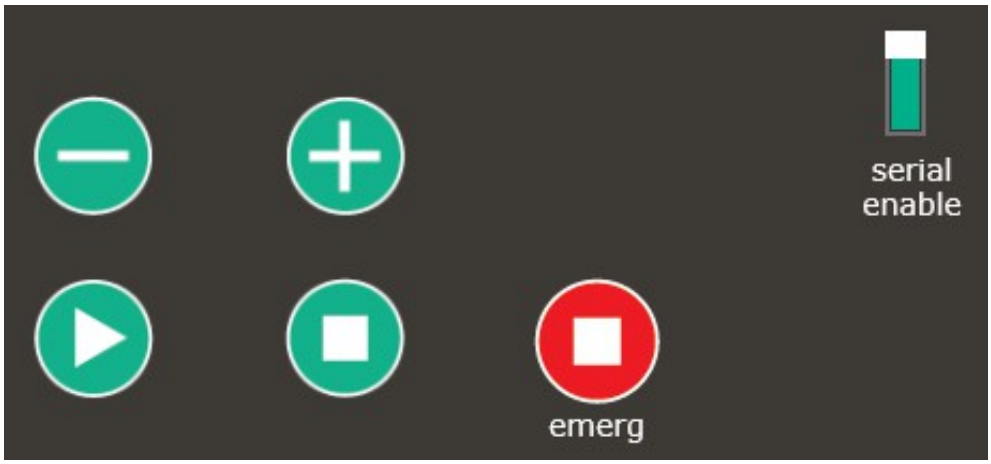
Current [mA]
See the Current value [0x0B] register on page 160.

Internal temperature [°C] Motor
See the Temperature value [0x07] register on page 158.

Internal temperature [°C] Electronic
See the Temperature value [0x07] register on page 158.

8.4 MODBUS commands

At the top right of the **Main** page some commands are available.



They allow to control manually and monitor the connected device. When you first enter the **Main** page, all commands are disabled as the unit is still under EtherCAT network control. To start programming, controlling manually and monitoring the device through the RS-232 service serial interface and MODBUS protocol, it is necessary to enable the available commands by gaining control of the unit in the MODBUS network via PC. To do this, set the **SERIAL ENABLE** slider 

(see Extra commands register [0x29] on page 147).
All commands become immediately available for use.

JOG -

See the **Jog -** item on page 148. If the **Incremental JOG** item is enabled = ON, the **Jog step length** value that is set currently appears between the **JOG –** button and the **JOG +** button. This button is available only if the MODBUS commands are enabled (see the **SERIAL ENABLE** slider 

). See the "8.8 "Parameters" page" section on page 119.

JOG +

See the **Jog +** item on page 148. If the **Incremental JOG** item is enabled = ON, the **Jog step length** value that is set currently appears between the **JOG –**

button and the **JOG +** button. This button is available only if the MODBUS commands are enabled (see the **SERIAL ENABLE** slider). See the "8.8 "Parameters" page" section on page 119.

START

Pressing the **START** button causes the unit to start running in order to reach the position set next to the **Target position** item. As soon as the commanded position is reached, the device stops and activates the **Axis in position** and **Target position reached** status bits (see the "8.5 "Status" box" section on page 114). For a normal halt of the device press the **STOP** button; for an immediate emergency halt press the **EMERGENCY** button. This button is available only if the MODBUS commands are enabled (see the **SERIAL ENABLE** slider). See the **Start** item on page 149.

STOP

Press the **STOP** button to force a normal halt of the device, respecting the deceleration values. This button is available only if the MODBUS commands are enabled (see the **SERIAL ENABLE** slider). See the **Stop** item on page 149.

EMERGENCY

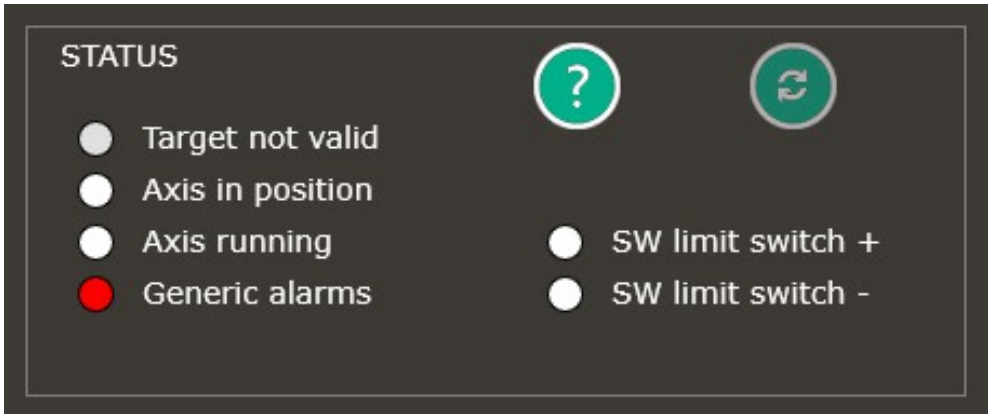
When the unit is running, press the **EMERGENCY** button to force an immediate halt in emergency condition. Press the **RESET** button to restore the normal work condition of the device (see the "8.5 "Status" box" section on page 114). This button is available only if the MODBUS commands are enabled (see the **SERIAL ENABLE** slider). See the **Emergency** item on page 150.

Serial enable

As previously stated, when you first enter the **Main** page, all commands are disabled as the unit is still under EtherCAT network control. To start programming, controlling manually and monitoring the device through the RS-232 service serial interface and MODBUS protocol, it is necessary to enable the available commands by gaining control of the unit in the MODBUS network via PC. To do this, set the **SERIAL ENABLE** slider (see **Extra commands register [0x29]** on page 147).

8.5 "Status" box

The **Status** box is available at the bottom right of the page.



Functions in the box provide short information about the status of the rotary actuator and allow to enter further pages containing more detailed information. The active alarms and serious states are lit red. The active ordinary states are lit green.

Please note that at power-on for safety reasons the RD6 unit is necessarily in an emergency condition: therefore when you first connect and enter the **Main** page the **Alarm occurred** warning is lit red. To know more about the specific active alarm enter the **Alarm and status** page by pressing the **STATUS & INFO** button, refer to the "8.6 "Alarm and status" page" section on page 115. To restore the **Idle** state of the device, press the **RESET** button in the box. Alarm warnings and emergencies will be reset and the normal work condition of the device will be restored.

For complete information on the states and alarms that appear in this box please refer to the **Status word [0x01]** register on page 156; and to the **Alarms register [0x00]** on page 154.

STATUS & INFO

Press this button to enter the **Alarm and status** page where specific information on the states and alarms that are active in the rotary actuator can be found. Refer to the "8.6 "Alarm and status" page" section on page 115.

RESET

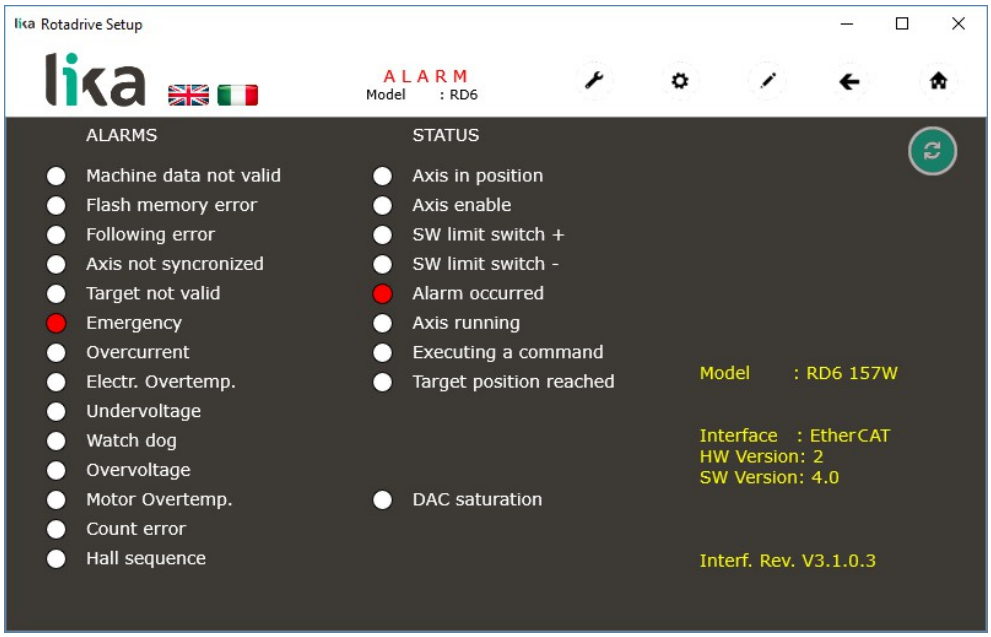
If an alarm is active, it is signalled through the generic warnings in the **Status** box. To know more about the specific active alarm enter the **Alarm and status** page by pressing the **STATUS & INFO** button, refer to the "8.6 "Alarm and status" page" section on page 115. Press this button to reset the alarm and restore the normal work condition of the device. This button is available only if

the MODBUS commands are enabled (see the **SERIAL ENABLE** slider 

). See the **Alarm reset** item on page 149.

8.6 "Alarm and status" page

When you press the **STATUS & INFO** button  in the **Status** box you enter the **Alarm and status** page.



As previously stated the **Status** box provides short information about the states and the alarms that are active in the rotary actuator. This **Alarm and status** page provides more detailed information on the active statuses and alarms. The active alarms and serious states are lit red. The active ordinary states are lit green.
For complete information on the states and alarms that appear in this page please refer to the Status word [0x01] register on page 156; and to the Alarms register [0x00] on page 154.

Furthermore the page provides detailed information on the connected actuator and the software tool. Data is listed in yellow at the bottom right of the page.
In particular you can found:

- **Model**: the model of the connected actuator;

- **Interface**: the protocol of the connected actuator;
- **HW version**: the hardware version of the connected actuator;
- **SW version**: the software version of the connected actuator;
- **Interf. Rev.**: the release of the software tool.

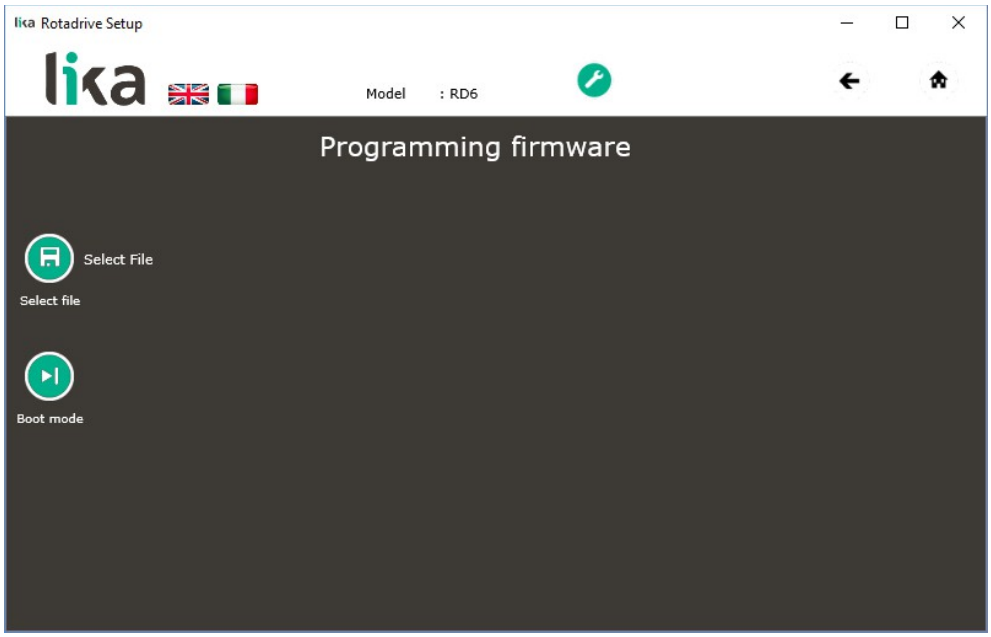
RESET

RESET button . See the previous page.

Press the **HOME** button to go back to the **Main** page.

8.7 "Programming firmware" page

By pressing the **FIRMWARE** button in the toolbar the operator enters the **Programming firmware** page.



The functions available in this page allow the operator to upgrade the firmware of the DRIVECOD unit by downloading upgrading data to the flash memory.
The firmware is a software program which controls the functions and operation of a device; the firmware program, sometimes referred to as "user program", is stored in the flash memory integrated inside the DRIVECOD unit. DRIVECOD

units are designed so that the firmware can be easily updated by the user himself. This allows Lika Electronic to make new improved firmware programs available during the lifetime of the product.
Typical reasons for the release of new firmware programs are the necessity to make corrections, improve and even add new functionalities to the device.
The firmware upgrading program consists of a single file having .BIN extension. It is released by Lika Electronic Technical Assistance & After Sale Service.

WARNING
The firmware upgrading process for any DRIVECOD unit has to be accomplished by skilled and competent personnel. If the upgrade is not performed according to the instructions provided or a wrong or incompatible firmware program is installed, then the unit may not be updated correctly, in some cases preventing the DRIVECOD unit from working.
If the latest firmware version is already installed in the DRIVECOD unit, you do not need to proceed with any new firmware installation. Current firmware version can be verified from the **SW Version** item in the **Alarm and status** page after connecting properly to the unit (see the previous page).
If you are not confident that you can perform the update successfully please contact Lika Electronic Technical Assistance & After Sale Service.

To upgrade the firmware program please proceed as follows:

1. make sure that the following configuration parameters are set (they are unmodifiable in the serial port of the DRIVECOD unit): baud rate = 9600 bits/s; parity = even; if they are set otherwise in your PC, please set them; see the "4.2.6 Modbus RS-232 service port" section on page 35;
2. make sure that the DRIVECOD unit you need to update is the only node connected to the personal computer;
3. connect to the unit, go online and then enter the **Programming firmware** page;
4. when you switch on the power supply, if user program is not present in the flash memory, you are not able to connect to the unit through the **Serial configuration** page; when this happens you need to enter directly the **Programming firmware** page; after the attempt to connect has failed the **FIRMWARE** button 

becomes available in the toolbar of the **Serial configuration** page; make sure that the correct serial port of the personal computer connected to the DRIVECOD unit is selected in the **Serial configuration** page;
5. press the **SELECT FILE** button 

; once you press the button the **Open** dialogue box appears on the screen: the operator must open the folder where the firmware upgrading .BIN file released by Lika Electronic is stored;

WARNING
Please note that for each DRIVECOD model having its own bus interface an appropriate firmware file is available. Make sure that you have the appropriate update for your DRIVECOD model. The .BIN file released by Lika Electronic has a file name that must be interpreted as follows.
For instance: RD6_EP_157_H1S1.BIN, where:

- RD6 = DRIVECOD unit model;
- EP = bus interface of the DRIVECOD unit (CB = CANopen; EC = EtherCAT; EP = EtherNet/IP; MB = MODBUS RTU; MT = MODBUS TCP; PB = Profibus; PL = POWERLINK; PT = Profinet);
- 157 = motor power;
- H1 = hardware version;
- S1 = firmware version.

6. select the .BIN file and confirm the choice by pressing the **OPEN** button, the dialog box closes;
7. the complete path of the file you have just confirmed appears next to the **SELECT FILE** button  ;
8. now press the **BOOT MODE** button  to enter the Boot Mode state;
9. the **UPLOAD** button  appears below in the page; press the button to start the firmware upgrading process;
10. a green download progress bar with percentage is displayed next to the **UPLOAD** button  ;

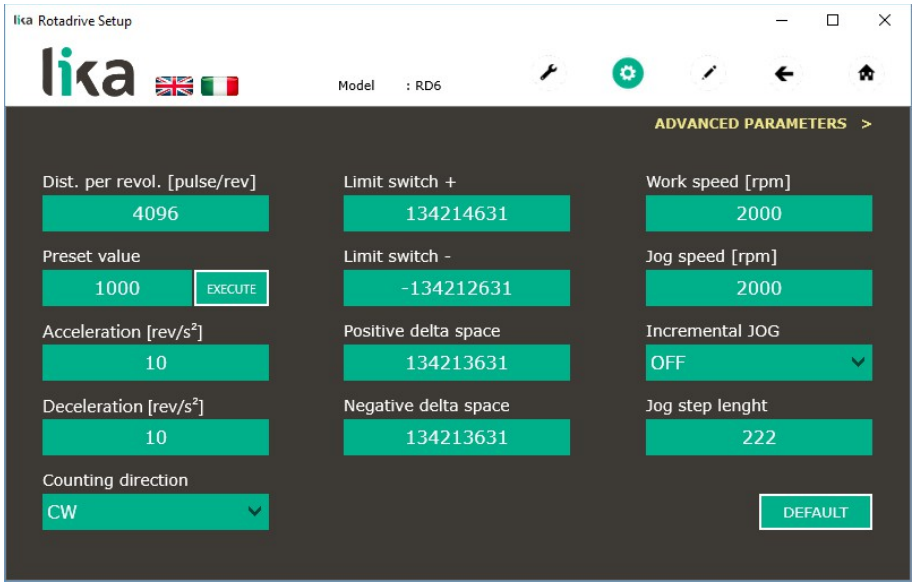
WARNING
Do not exit the **Programming firmware** page during installation, the process will be aborted!

11. as soon as the operation is carried out successfully, the **OK** message is displayed for a while; then the actuator reboots and the **Serial configuration** page appears on the screen;
12. reconnect to the DRIVECOD unit and restore the normal work condition.

Press the **HOME** button to go back to the **Main** page.

8.8 "Parameters" page

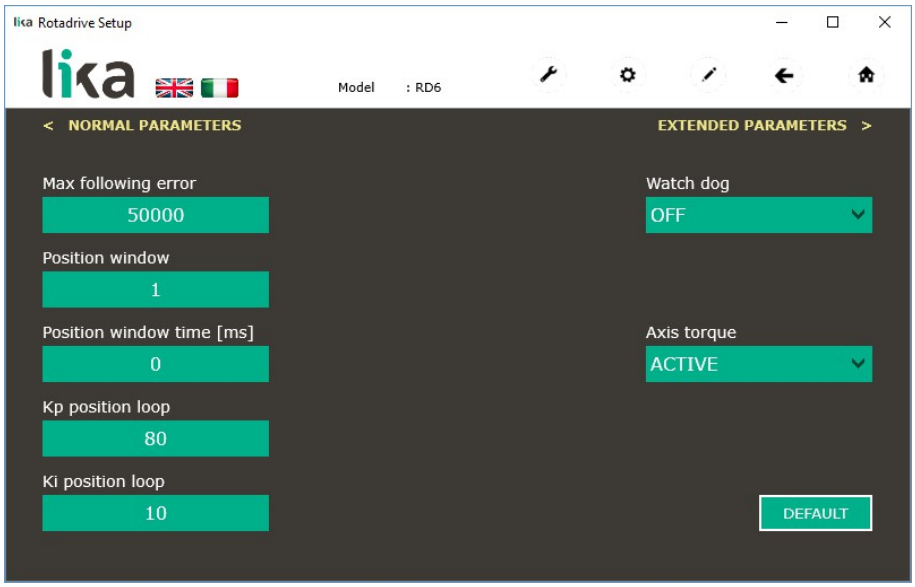
Press the **SETUP** button in the toolbar to configure the device and set the parameters. The page below will appear.



In this page the list of the "normal parameters" available to set the RD6 positioning units is displayed. Parameters in the page need to be set more usually when you configure and test a new program of the actuator.

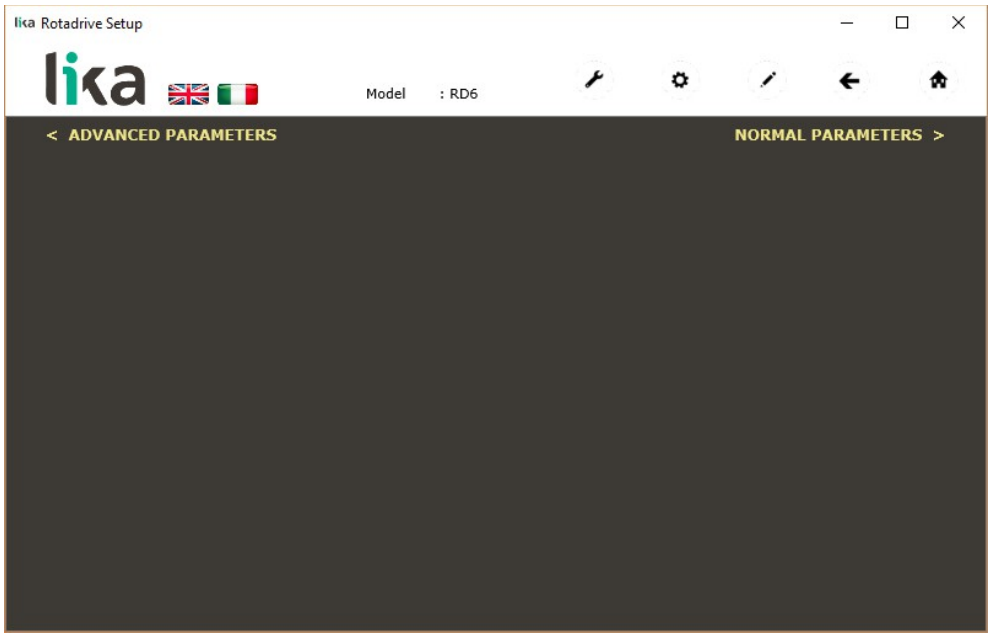
ADVANCED PARAMETERS > ADVANCED PARAMETERS >

A further page is accessible by pressing the **ADVANCED PARAMETERS >** button ADVANCED PARAMETERS >: the "advanced parameters" that need to be set more seldom are listed in the page.



EXTENDED PARAMETERS EXTENDED PARAMETERS >

Press the **EXTENDED PARAMETERS** > button EXTENDED PARAMETERS > to enter the page where the outputs can be enabled. RD6 rotary actuator does not provide output signals, so a blank page will appear.



NORMAL PARAMETERS NORMAL PARAMETERS

Press the **NORMAL PARAMETERS** button NORMAL PARAMETERS to go back to the first **Parameters** page.

For detailed information on the function and the setting of the parameters refer to the "8.14.1 Holding Register parameters" section on page 140.

The values that are currently set in the unit are shown under each field.
To enter a new value type it in the field and then press the **ENTER** key in the keyboard. If you set a value that is not allowed (out of range), at confirmation prompt the minimum or maximum value will be set instead. In some cases you are required to select the desired value by means of the drop-down box.
As soon as the new value is confirmed, it is downloaded to the unit and saved automatically.

DEFAULT DEFAULT

When you need to load the default parameters (they are set at the factory by Lika Electronic engineers to allow the operator to run the device for standard

operation in a safe mode) press the **DEFAULT** button 

. For any further information on loading the default parameters refer to the **Load default parameters** variable on page 150. The complete list of the machine data parameters and the relevant default values as set by Lika Electronic are available on page 170.

In the **Parameters** pages the following functions are also available.

EXECUTE 

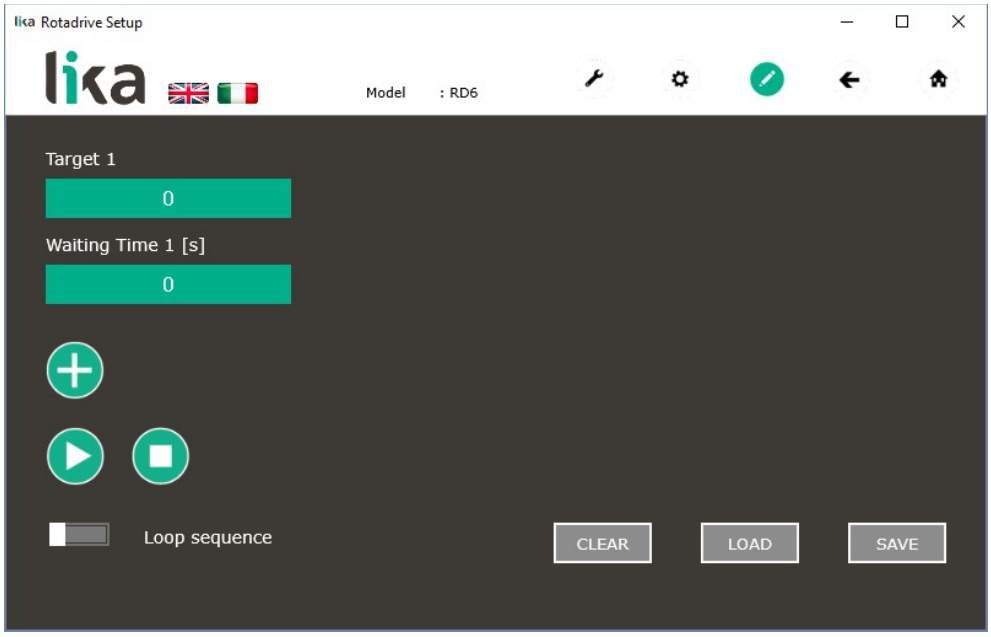
The **EXECUTE** button 

is placed next to the **Preset** item in the first **Parameters** page. When you enter a new value next to the **Preset** item, the Preset is saved but not activated. Press the **EXECUTE** button 

to activate the Preset value for the current position of the actuator's shaft. For any further information on activating the Preset value refer to the bit 11 **Setting the preset** in the **Control Word [0x2A]** register on page 151. Refer also to the **Preset [0x12-0x13]** registers on page 146.

8.9 "Program" page

Press the **SCHEDULE** button Schedule in the toolbar to enter the **Program** page. The page below will appear.

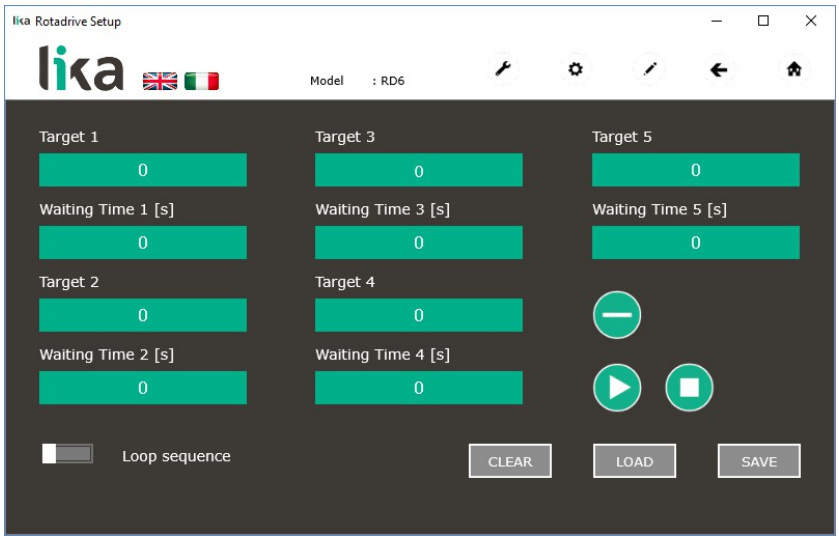


The functions available in the **Program** page allow the operator to create and save work programs for the RD6 unit. In this way it is possible to test the operation of the actuator (speed, acceleration, deceleration, etc.) by setting up to five targets and commanding their execution.

Target 1 ... 5

The first position the device is commanded to reach (target position) must be set next to the **Target 1** item. Press the **ENTER** key in the keyboard to confirm.

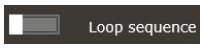
It is possible to enter up to five subsequent positions. Press the **PLUS** button to add more target positions for the actuator to reach.

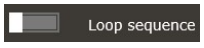


You can press the **MINUS** button to delete the last target.

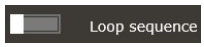
Waiting Time 1 [s]

Next to the **Waiting Time 1 [s]** item you must set the interval between the first step (commanded movement) and the second. Press the **ENTER** key in the keyboard to confirm.

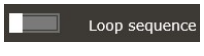
Loop sequence 

The **Loop sequence** slider 

enables / disables the "loop" function, i.e. after pressing the **START** button the device goes on running and executing the set steps without interruption, from **Target 1** to **Target 5** (if enabled) and again from **Target 1** to **Target 5**, until you press the **STOP** button .

If the **Loop sequence** slider 

is activated, when you press the **START** button , the device starts moving in order to reach the first commanded position **Target 1**; a green light appears next to the field and a green progress bar is shown at the top of the page; as soon as the commanded position set next to the **Target 1** item is reached, the device stops. After the set interval **Waiting Time 1 [s]** has expired, a green light appears next to the **Target 2** item and the RD6 unit restarts running in order to reach the second commanded position **Target 2**; again a green progress bar is shown at the top of the page; and so on, from the first to the fifth commanded position (if enabled) and then again from the first to the fifth commanded position without interruption, until you press the **STOP** button .

If the **Loop sequence** slider 

is not activated, when you press the **START** button , the device starts running in order to reach the first commanded position **Target 1**; as soon as the first commanded position **Target 1** is reached, the device stops and waits for the set interval **Waiting Time 1 [s]**

to expire: the sequence of movements is carried out; you must then press the **START** button  again to command the unit to reach the second position **Target 2**; and so on.

The following buttons are available in the page:

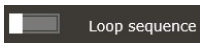
PLUS 

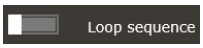
Press the **PLUS** button  to add more target positions for the actuator to reach.

MINUS 

Press the **MINUS** button  to delete the last target.

START 

Press the **START** button  to command the unit to reach the set target position. If the **Loop sequence** slider 

is activated, you are required to press the **START** button  just once: the actuator will reach in sequence all the commanded positions that are enabled. If the **Loop sequence** slider 

is not activated, you must press the **START** button  after each step to go on.

STOP 

Press the **STOP** button  to stop the sequence of movements.

CLEAR 

Press the **CLEAR** button 

to zero-set the counter designed to count the steps during the execution of the running program: when the operator presses the **START** button  the device will start running from step 1, i.e. in order to reach the first commanded position **Target 1**, whatever the position reached before the counter was zero-set.

LOAD 

Press the **LOAD** button 

to load a work program that has been saved previously. Once you press the button, the **Open** dialog box appears on the screen: the operator must open the folder where the previously saved .dat file is stored, then select it and finally confirm the choice by pressing the **OPEN** button, the dialog box closes and the work values are loaded automatically.

SAVE 

Press the **SAVE** button 

to save the parameters of the work program you have just created. Once you press the button the **Save as** dialog box appears on the screen: the operator must type the name of the .dat file and specify the path where the file has to be stored. When you press the **SAVE** button to confirm, the dialog box closes. Set values are saved automatically.

8.10 Modbus Master / Slaves protocol principle

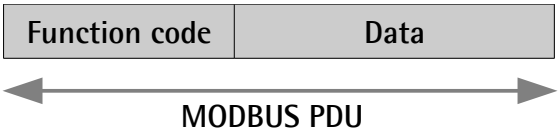
The Modbus Serial Line protocol is a Master – Slaves protocol. One only Master (at the same time) is connected to the bus and one or several (up to 247) Slave nodes are also connected to the same serial bus. A Modbus communication is always initiated by the Master. The Slave nodes will never transmit data without receiving a request from the Master node. The Slave nodes will never communicate with each other. The Master node initiates only one Modbus transaction at the same time.

The Master node issues a Modbus request to the Slave nodes in two modes:

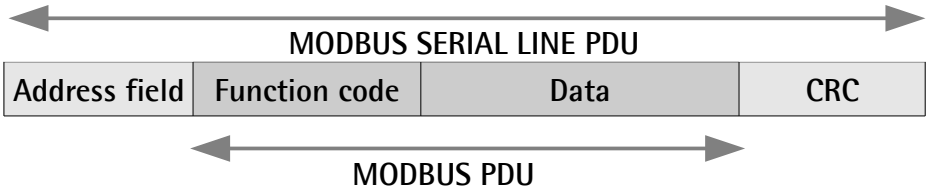
- **UNICAST mode**: in that mode the Master addresses an individual Slave. After receiving and processing the request, the Slave returns a message (a "reply") to the Master. In that mode, a Modbus transaction consists of two messages: a request from the Master and a reply from the Slave. Each Slave must have a unique address (from 1 to 247) so that it can be addressed independently from the other nodes. Lika devices only implement commands in "unicast" mode.
- **BROADCAST mode**: in that mode the Master can send a request to all Slaves at the same time. No response is returned to "broadcast" requests sent by the Master. The "broadcast" requests are necessarily writing commands. The address 0 is reserved to identify a "broadcast" exchange. Lika devices do not implement commands in "broadcast" mode.

8.11 Modbus frame description

The Modbus application protocol defines a simple Protocol Data Unit (PDU) independent of the underlying communication layers:



The mapping of Modbus protocol on a specific bus or network introduces some additional fields on the Protocol Data Unit. The client that initiates a Modbus transaction builds the Modbus PDU, and then adds fields in order to build the appropriate communication PDU.



- **ADDRESS FIELD**: on Modbus Serial Line the address field only contains the Slave address. As previously stated (see the "8.10 Modbus Master / Slaves protocol principle" section on page 125), the valid Slave node addresses are in the range of 0 – 247 decimal. The individual Slave devices are assigned addresses in the range of 1 – 247. A Master addresses a Slave by placing the Slave address in the **ADDRESS FIELD** of the message. When the Slave returns its response, it places its own address in the response **ADDRESS FIELD** to let the Master know which Slave is responding.
- **FUNCTION CODE**: the function code indicates to the Server what kind of action to perform. The function code can be followed by a **DATA** field that contains request and response parameters. For any further information on the implemented function codes refer to the "8.13 Function codes" section on page 130.
- **DATA**: the **DATA** field of messages contains the bytes for additional information and transmission specifications that the server uses to take the action defined by the **FUNCTION CODE**. This can include items such as discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field. The structure of the **DATA** field depends on each **FUNCTION CODE** (refer to the "8.13 Function codes" section on page 130).
- **CRC (Cyclic Redundancy Check)**: error checking field is the result of a "Redundancy Check" calculation that is performed on the message

contents. This is intended to check whether transmission has been performed properly. The CRC field is two bytes, containing 16-bit binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The device that receives recalculates a CRC during receipt of the message and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results.

The Modbus protocol defines three PDUs. They are:

- **Modbus Request PDU**;
- **Modbus Response PDU**;
- **Modbus Exception Response PDU**.

The **Modbus Request PDU** is defined as {function_code, request_data}, where:
function_code = Modbus function code [1 byte];
request_data = this field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function, etc. [n bytes].

The **Modbus Response PDU** is defined as {function_code, response_data}, where:
function_code = Modbus function code [1 byte];
response_data = this field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function, etc. [n bytes].

The **Modbus Exception Response PDU** is defined as {exception-function_code, exception_code}, where:
exception-function_code = Modbus function code + 0x80 [1 byte];
exception_code = Modbus Exception code, refer to the table "Modbus Exception Codes" in the section 7 of the document "Modbus Application Protocol Specification V1.1b".

8.12 Transmission modes

Two different serial transmission modes are defined in the Modbus serial protocol: the **RTU** (**Remote Terminal Unit**) **mode** and the **ASCII mode**. The transmission mode defines the bit contents of message fields transmitted serially on the line. It determines how information is packed into the message fields and decoded. The transmission mode and the serial port parameters must be the same for all devices on a Modbus Serial Line. All devices must implement the RTU mode, while the ASCII mode is an option. Lika devices only implement RTU transmission mode, as described in the following section.

8.12.1 RTU transmission mode

When devices communicate on a Modbus serial line using the RTU (Remote Terminal Unit) mode, each 8-bit byte in a message contains two 4-bit hexadecimal characters. Each message must be transmitted in a continuous stream of characters. Synchronization between the messages exchanged by the transmitting device and the receiving device is achieved by placing an interval of at least 3.5 character times between successive messages, this is called "silent interval". In this way a Modbus message is placed by the transmitting device into a frame that has a known beginning and ending point. This allows devices that receive a new frame to begin at the start of the message and to know when the message is completed. So when the receiving device does not receive a message for an interval of 4 character times, it considers the previous message as completed and the next byte will be the first of a new message, i.e. an address.
When baud rate = 9600 bit/s the "silent interval" is 4 ms.
When baud rate = 19200 bit/s the "silent interval" is 2 ms.

The format (11 bits) for each byte in RTU mode is as follows:

Coding system: 8-bit binary
Bits per Byte: 1 start bit;
8 data bits, least significant bit (lsb) sent first;
1 bit for parity completion (= Even);
1 stop bit.

Modbus protocol uses a "big-Endian" representation for addresses and data items. This means that when a numerical quantity greater than a single byte is transmitted, the most significant byte (MSB) is sent first.
Each character or byte is sent in this order (left to right):

lsb (Least Significant Bit) ... msb (Most Significant Bit)

Start	1	2	3	4	5	6	7	8	Parity*	Stop

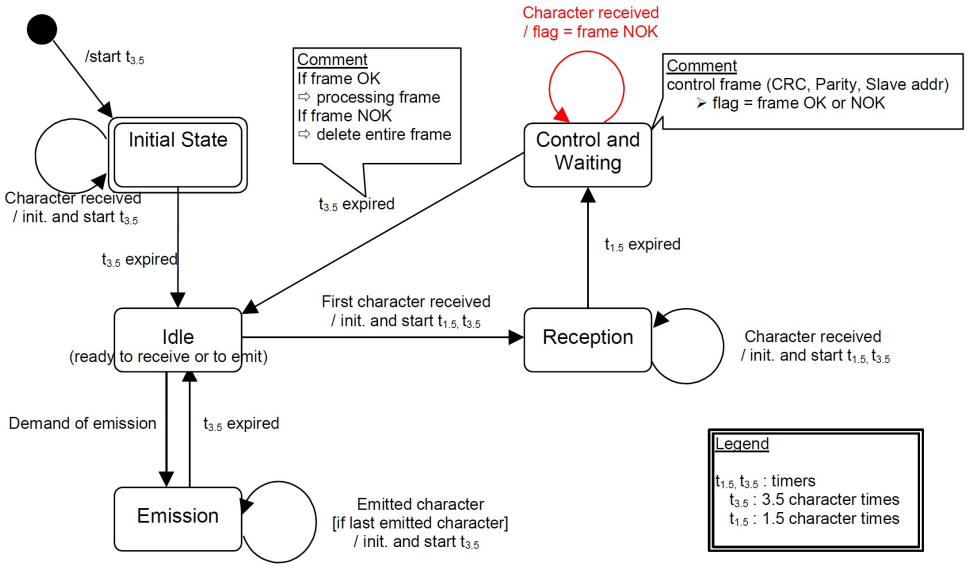
* When "No parity" is activated, the parity bit is replaced by a stop bit.

The default parity mode must be even parity.

The maximum size of the Modbus RTU frame is 256 bytes, its structure is as follows:

Slave Address	Function code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low | CRC Hi

The following drawing provides a description of the RTU transmission mode state diagram. Both "Master" and "Slave" points of view are expressed in the same drawing.



- Transition from **Initial State** to **Idle** state needs an interval of at least 3.5 character times (time-out expiration = $t_{3.5}$).
- **Idle** state is the normal state when neither emission nor reception is active. In RTU mode, the communication link is declared in **Idle** state when there is no transmission activity after a time interval equal to at least 3.5 characters ($t_{3.5}$).
- A request can only be sent in **Idle** state. After sending a request, the Master leaves the **Idle** state and cannot send a second request at the same time.
- When the link is in **Idle** state, each transmitted character detected on the link is identified as the start of the frame. The link goes to **Active** state. Then the end of the frame is identified when no more character is transmitted on the link after the time interval of at least $t_{3.5}$.
- After detection of the end of frame, the CRC calculation and checking is completed. Afterwards the address field is analysed to determine if the frame is addressed to the device. If not, the frame is discarded. In order to reduce the reception processing time the address field can be analysed as soon as it is received without waiting the end of frame. In this case the CRC will be calculated and checked only if the frame is actually addressed to the Slave.

8.13 Function codes

As previously stated, the function code indicates to the Server what kind of action to perform. The function code field of a Modbus data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (the range 128 ... 255 is reserved and used for Exception Responses). When a message is sent from a Client to a Server device the function code field tells the Server what kind of action to perform. Function code "0" is not valid.
There are three categories of Modbus function codes, they are: **public function codes**, **user-defined function codes** and **reserved function codes**.
Public function codes are in the range 1 ... 64, 73 ... 99 and 111 ... 127; they are well defined function codes, validated by the MODBUS-IDA.org community and publicly documented; furthermore they are guaranteed to be unique. Ranges of function codes from 65 to 72 and from 100 to 110 are **user-defined function codes**: user can select and implement a function code that is not supported by the specification, it is clear that there is no guarantee that the use of the selected function code will be unique. **Reserved function codes** are not available for public use.

8.13.1 Implemented function codes

Lika RD6 Modbus positioning units only implement public function codes, they are described hereafter.

03 Read Holding Registers

FC = 03 (Hex = 0x03) ro
This function code is used to READ the contents of a contiguous block of holding registers in a remote device; in other words, it allows to read the values set in a group of work parameters placed in order. The Request PDU specifies the starting register address and the number of registers. In the PDU registers are addressed starting at zero. Therefore registers numbered 1-16 are addressed as 0-15.
The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits (msb) and the second contains the low order bits (lsb).
For the complete list of holding registers accessible using **03 Read Holding Registers** function code please refer to the "8.14.1 Holding Register parameters" section on page 140.

Request PDU

Function code	1 byte	**0x03**
Starting address	2 bytes	0x0000 to 0xFFFF
Quantity of registers	2 bytes	1 to 125 (0x7D)

Response PDU

Function code	1 byte	**0x03**
Byte count	1 byte	2 x **N***
Register value	**N*** x 2 bytes	

*N = Quantity of registers

Exception Response PDU

Error code	1 byte	**0x83 (=0x03 + 0x80)**
Exception code	1 byte	01 or 02 or 03 or 04

Here is an example of a request to read the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9).

Request	
Field name	(Hex)
Function	**03**
Starting address Hi	**00**
Starting address Lo	**07**
No. of registers Hi	**00**
No. of registers Lo	**02**

Response	
Field name	(Hex)
Function	**03**
Byte count	**04**
Register 8 value Hi	**00**
Register 8 value Lo	**0A**
Register 9 value Hi	**00**
Register 9 value Lo	**0A**

As you can see in the table, **Acceleration [0x07]** parameter (register 8) contains the value 00 0A hex, i.e. 10 in decimal notation; **Deceleration [0x08]** parameter (register 9) contains the value 00 0A hex, i.e. 10 in decimal notation.

The full frame needed for the request to read the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9) to the Slave having the node address 1 is as follows:

Request PDU (in hexadecimal format)
[01][03][00][07][00][02][75][CA]
where:
[01] = Slave address
[03] = **03 Read Holding Registers** function code
[00][07] = starting address (**Acceleration [0x07]** parameter, register 8)
[00][02] = number of requested registers
[75][CA] = CRC

The full frame needed to send back the values of the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9) from the Slave having the node address 1 is as follows:

Response PDU (in hexadecimal format)
[01][03][04][00][0A][00][0A][5A][36]
where:
[01] = Slave address
[03] = **03 Read Holding Registers** function code
[04] = number of bytes (2 bytes for each register)
[00][0A] = value of register 8 **Acceleration [0x07]**, 00 0A hex = 10 dec
[00][0A] = value of register 9 **Deceleration [0x08]**, 00 0A hex = 10 dec
[5A][36] = CRC

04 Read Input Register

FC = 04 (Hex = 0x04)

This function code is used to READ from 1 to 125 contiguous input registers in a remote device; in other words, it allows to read some results values and state / alarm messages in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU registers are addressed starting at zero. Therefore input registers numbered 1-16 are addressed as 0-15.
The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits (msb) and the second contains the low order bits (lsb).
For the complete list of input registers accessible using **04 Read Input Register** function code please refer to the "8.14.2 Input Register parameters" section on page 154.

Request PDU

Function code	1 byte	**0x04**
Starting address	2 bytes	0x0000 to 0xFFFF
Quantity of Input Registers	2 bytes	0x0000 to 0x007D

Response PDU

Function code	1 byte	**0x04**
Byte count	1 byte	2 x **N***
Input register value	**N*** x 2 bytes	

*N = Quantity of registers

Exception Response PDU

Error code	1 byte	**0x84 (=0x04 + 0x80)**
Exception code	1 byte	01 or 02 or 03 or 04

Here is an example of a request to read the **Current position [0x02-0x03]** parameter (input registers 3 and 4).

Request	
Field name	(Hex)
Function	**04**
Starting address Hi	**00**
Starting address Lo	**02**
Quantity of Input Reg. Hi	**00**
Quantity of Input Reg. Lo	**02**

Response	
Field name	(Hex)
Function	**04**
Byte count	**04**
Register 3 value Hi	**00**
Register 3 value Lo	**00**
Register 4 value Hi	**2F**
Register 4 value Lo	**F0**

As you can see in the table, the **Current position [0x02-0x03]** parameter (input registers 3 and 4) contains the value 00 00 2F F0 hex, i.e. 12272 in decimal notation.

The full frame needed for the request to read the **Current position [0x02-0x03]** parameter (input registers 3 and 4) to the Slave having the node address 1 is as follows:

Request PDU (in hexadecimal format)
[01][04][00][02][00][02][D0][0B]
where:
[01] = Slave address
[04] = **04 Read Input Register** function code
[00][02] = starting address (**Current position [0x02-0x03]** parameter, register 3)
[00][02] = number of requested registers
[D0][0B] = CRC

The full frame needed to send back the value of the **Current position [0x02-0x03]** parameter (registers 3 and 4) from the Slave having the node address 1 is as follows:

Response PDU (in hexadecimal format)
[01][04][04][00][00][2F][F0][E7][F0]
where:
[01] = Slave address
[04] = **04 Read Input Register** function code
[04] = number of bytes (2 bytes for each register)
[00][00] = value of register 3 **Current position [0x02-0x03]**, 00 00 hex = 0 dec
[2F][F0] = value of register 4 **Current position [0x02-0x03]**, 2F F0 hex = 12272 dec
[E7][F0] = CRC

06 Write Single Register

FC = 06 (Hex = 0x06)

This function code is used to WRITE a single holding register in a remote device. The Request PDU specifies the address of the register to be written. Registers are addressed starting at zero. Therefore register numbered 1 is addressed as 0.
The normal response is an echo of the request, returned after the register contents have been written.
For the complete list of registers accessible using **06 Write Single Register** function code please refer to the "8.14.1 Holding Register parameters" section on page 140.

Request PDU

Function code	1 byte	**0x06**
Register address	2 bytes	0x0000 to 0xFFFF
Register value	2 bytes	0x0000 to 0xFFFF

Response PDU

Function code	1 byte	**0x06**
Register address	2 bytes	0x0000 to 0xFFFF
Register value	2 bytes	0x0000 to 0xFFFF

Exception Response PDU

Error code	1 byte	**0x86 (=0x06 + 0x80)**
Exception code	1 byte	01 or 02 or 03 or 04

Here is an example of a request to write the value 00 96 hex (= 150 dec) in the **Acceleration [0x07]** parameter (register 8).

Request	
Field name	(Hex)
Function	**06**
Register address Hi	**00**
Register address Lo	**07**
Register value Hi	**00**
Register value Lo	**96**

Response	
Field name	(Hex)
Function	**06**
Register address Hi	**00**
Register address Lo	**07**
Register value Hi	**00**
Register value Lo	**96**

As you can see in the table, the value 00 96 hex, i.e. 150 in decimal notation, is set in the **Acceleration [0x07]** parameter (register 8).

The full frame needed for the request to write the value 00 96 hex (= 150 dec) in the **Acceleration [0x07]** parameter (register 8) to the Slave having the node address 1 is as follows:

Request PDU (in hexadecimal format)
[01][06][00][07][00][96][B8][65]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][07] = address of the register (**Acceleration [0x07]** parameter, register 8)
[00][96] = value to be set in the register
[B8][65] = CRC

The full frame needed to send back a response following the request to write the value 00 96 hex (= 150 dec) in the **Acceleration [0x07]** parameter (register 8) from the Slave having the node address 1 is as follows:

Response PDU (in hexadecimal format)
[01][06][00][07][00][96][B8][65]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][07] = address of the register (**Acceleration [0x07]** parameter, register 8)
[00][96] = value set in the register
[B8][65] = CRC

16 Write Multiple Registers

FC = 16 (Hex = 0x10)

This function code is used to WRITE a block of contiguous registers (1 to 123 registers) in a remote device.
The values to be written are specified in the request data field. Data is packed as two bytes per register.
The normal response returns the function code, starting address and quantity of written registers.
For the complete list of registers accessible using **16 Write Multiple Registers** function code please refer to the "8.14.1 Holding Register parameters" section on page 140.

Request PDU

Function code	1 byte	**0x10**
Starting address	2 bytes	0x0000 to 0xFFFF
Quantity of registers	2 bytes	0x0001 to 0x007B
Byte count	1 byte	2 x **N***
Registers value	**N*** x 2 bytes	value

*N = Quantity of registers

Response PDU

Function code	1 byte	**0x10**
Starting address	2 bytes	0x0000 to 0xFFFF
Quantity of registers	2 bytes	1 to 123 (0x7B)

Exception Response PDU

Error code	1 byte	**0x90 (= 0x10 + 0x80)**
Exception code	1 byte	01 or 02 or 03 or 04

Here is an example of a request to write the values 150 and 100 dec in the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9) respectively.

Request	
Field name	(Hex)
Function	**10**
Starting address Hi	**00**
Starting address Lo	**07**
Quantity of registers Hi	**00**
Quantity of registers Lo	**02**
Byte count	**04**
Register 8 value Hi	**00**

Response	
Field name	(Hex)
Function	**10**
Starting address Hi	**00**
Starting address Lo	**07**
Quantity of registers Hi	**00**
Quantity of registers Lo	**02**

Register 8 value Lo	**96**
Register 9 value Hi	**00**
Register 9 value Lo	**64**

As you can see in the table, the value 00 96 hex, i.e. 150 in decimal notation, is set in the **Acceleration [0x07]** parameter (register 8); the value 00 64 hex, i.e. 100 in decimal notation, is set in the **Deceleration [0x08]** parameter (register 9).

The full frame needed for the request to write the values 00 96 hex (= 150 dec) and 00 64 hex (= 100 dec) in the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9) to the Slave having the node address 1 is as follows:

Request PDU (in hexadecimal format)
[01][10][00][07][00][02][04][00][96][00][64][53][8E]
where:
[01] = Slave address
[10] = **16 Write Multiple Registers** function code
[00][07] = starting address (**Acceleration [0x07]** parameter, register 8)
[00][02] = number of requested registers
[04] = number of bytes (2 bytes for each register)
[00][96] = value to be set in the register 8 **Acceleration [0x07]**, 00 96 hex = 150 dec
[00][64] = value to be set in the register 9 **Deceleration [0x08]**, 00 64 hex = 100 dec
[53][8E] = CRC

The full frame needed to send back a response following the request to write the values 00 96 hex (= 150 dec) and 00 64 hex (= 100 dec) in the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9) from the Slave having the node address 1 is as follows:

Response PDU (in hexadecimal format)
[01][10][00][07][00][02][F0][09]
where:
[01] = Slave address
[10] = **16 Write Multiple Registers** function code
[00][07] = starting address (**Acceleration [0x07]** parameter, register 8)
[00][02] = number of written registers
[F0][09] = CRC

NOTE
For further examples refer to the "8.16 Programming examples" section on page 164.

WARNING
For safety reasons, when the DRIVECOD unit is on, a continuous data exchange between the Master and the Slave has to be planned in order to be sure that the communication is always active; this is intended to prevent danger situations from arising in case of failures in the communication network.
For this purpose the Watch dog function is implemented and can be activated as optional. The Watch dog function is a safety timer that uses a time-out to detect loop or deadlock conditions. For instance, should the serial communication be cut off while a command is still active and running -a jog command for example- the Watch dog safety system immediately takes action and commands a safety stop of the device; furthermore an alarm is triggered. To enable the Watch dog function, set to "=1" the **Watch dog enable** bit in the **Control Word [0x2A]** variable. If "=0" is set the Watch dog is not enabled; if "=1" is set the Watch dog is enabled. When the Watch dog function is enabled, if the device does not receive a message from the Server within 1 second, the system forces an alarm condition (the **Watch dog** alarm message is invoked to appear as soon as the Modbus network communication is restored).

8.14 Programming parameters

Hereafter the parameters available for RD6 Modbus devices are listed and described as follows:

Parameter name [Register address]
[Register number, data types, attribute]

- The register address is expressed in hexadecimal notation.
- The register number is expressed in decimal notation.
- Attribute:
  ro = read only access
  rw = read and write access

The MODBUS registers are 16-bit long; while the actuator parameters can be 1-register long, i.e. 16-bit long, or 2-register long, i.e. 32-bit long.

Unsigned16 parameter structure:

byte	MSB			LSB		
bit	15	...	8	7	...	0
	msb		lsb	msb		lsb

Unsigned32 parameter structure:

word	MSW			LSW		
bit	31	...	16	15	...	0
	msb		lsb	msb		lsb

8.14.1 Holding Register parameters

Holding registers (**Machine data** parameters) are accessible for both writing and reading; to read the value set in a parameter use the **03 Read Holding Registers** function code (reading of multiple registers); to write a value in a parameter use the **06 Write Single Register** function code (writing of a single register) or the **16 Write Multiple Registers** (writing of multiple registers); for any further information on the implemented function codes refer to the "8.13.1 Implemented function codes" section on page 130.

NOTE
Always save the new values after setting in order to store them in the non-volatile memory permanently. Use the **Save parameters** function available in the Control Word [0x2A] register, see on page 148.
Should the power supply be turned off all data that has not been saved previously will be lost!

WARNING
For safety reasons the following holding register parameters Extra commands register [0x29], Control Word [0x2A] and Target position [0x2B-0x2C] are not stored in the memory. So they are required to be set after each power-on.

Distance per revolution [0x00]

[Register 1, Unsigned16, rw]
This parameter sets the number of pulses per each complete revolution of the shaft. It is useful to relate the revolution of the shaft and a linear measurement. For example: the unit is joined to a warm screw having 5 mm (0.197") pitch; by setting Distance per revolution [0x00] = 500, at each shaft revolution the system performs a 5 mm (0.197") pitch with one-hundredth of a millimetre resolution.
Default = 4096 (min. = 1, max. = 4096)

WARNING
After having changed this parameter you must then set new values also next to the Preset [0x12-0x13] parameter. For a detailed explanation see on page 45 and relevant parameters.
Please note that the parameters listed hereafter are closely related to the Distance per revolution [0x00] parameter; hence when you change the value in Distance per revolution [0x00] also the value in each of them necessarily changes. They are: Position window [0x01], Max following error [0x03-0x04], Positive delta [0x09-0x0A], Negative delta [0x0B-0x0C], Target position [0x2B-0x2C], Current position [0x02-0x03] and Position following error [0x05-0x06]. See also on page 145.

NOTE
If Distance per revolution [0x00] is not a power of 2 (2, 4, ..., 2048, 4096), at position control a positioning error could occur having a value equal to one pulse.

Position window [0x01]

[Register 2, Unsigned16, rw]
This parameter defines the tolerance window limits for the **Target position [0x2B-0x2C]** value. As soon as the axis is within the tolerance window limits, the bit 8 **Target position reached** in the **Status word [0x01]** goes high ("=1"). When the axis is within the tolerance window limits for the time set in the **Position window time [0x02]** parameter, the bit 0 **Axis in position** in the **Status word [0x01]** goes high ("=1"). The parameter is expressed in pulses. See also the "Positioning: position and speed control" section on page 44.
Default = 1 (min. = 0, max. = 65535)

Position window time [0x02]

[Register 3, Unsigned16, rw]
It represents the time for which the axis has to be within the tolerance window limits set in the **Position window [0x01]** parameter before the state is signalled through the **Axis in position** status bit of the **Status word [0x01]**. The parameter is expressed in milliseconds. See also the "Positioning: position and speed control" section on page 44.
Default = 0 (min. = 0, max. = 10000)

Max following error [0x03-0x04]

[Registers 4-5, Unsigned32, rw]
This parameter defines the maximum allowable difference between the real position and the theoretical position of the device. If the device detects a value higher than the one set in this parameter, the **Following error** alarm is triggered and the unit stops. The parameter is expressed in pulses.
Default = 50000 (min. = 0, max. = 1000000)

Kp position loop [0x05]

[Register 6, Unsigned16, rw]
This parameter contains the proportional gain used by the PI controller for the position loop. The value has been optimized by Lika Electronic according to the technical characteristics of the device.
Default = 80 (min. = 0, max. = 1000)

Ki position loop [0x06]

[Register 7, Unsigned16, rw]
This parameter contains the integral gain used by the PI controller for the position loop. The value has been optimized by Lika Electronic according to the technical characteristics of the device.
Default = 10 (min. = 0, max. = 1000)

Acceleration [0x07]

[Register 8, Unsigned16, rw]
This parameter defines the acceleration value that has to be used by the device when reaching both the **Jog speed [0x0D]** and the **Work speed [0x0E]**. The parameter is expressed in revolutions per second^2 [rev/s^2]. See also the "6.2 Movements: jog and positioning" section on page 43.
Default = 10 (min. = 1, max. = 500)

Deceleration [0x08]

[Register 9, Unsigned16, rw]
This parameter defines the deceleration value that has to be used by the device when stopping. The parameter is expressed in revolutions per second^2 [rev/s^2]. See also the "6.2 Movements: jog and positioning" section on page 43.
Default = 10 (min. = 1, max. = 500)

Positive delta [0x09-0x0A]

[Registers 10-11, Unsigned32, rw]
This value is used to calculate the maximum forward (positive) limit the device is allowed to reach starting from the preset value. Should it happen that the maximum forward limit is reached, a signalling is activated through the **SW limit switch +** status bit of the **Status word [0x01]** (the bit is forced high). The parameter is expressed in pulses.
SW limit switch + = **Preset [0x12-0x13]** + **Positive delta [0x09-0x0A]**.
For further information please refer to the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.
Default = 134 213 631 (min. = 0, max. = 134 213 631)

WARNING
Please mind the maximum acceptable value for this item depends on the set scaling.

EXAMPLE
When **Distance per revolution [0x00]** = 4,096 and **Preset [0x12-0x13]** = 0, the maximum acceptable value for **Positive delta [0x09-0x0A]** is:
(4,096 steps per revolution * 32,768 revolutions) - 1 step – 4,096 steps (i.e. 1 revolution for safety reasons) = 134 213 631 (see the default value)
When **Distance per revolution [0x00]** = 1,024 and **Preset [0x12-0x13]** = 0, the maximum acceptable value for **Positive delta [0x09-0x0A]** is:
(1,024 steps per revolution * 32,768 revolutions) - 1 step – 1,024 steps (i.e. 1 revolution for safety reasons) = 33 553 407

When **Distance per revolution [0x00]** = 256 and **Preset [0x12-0x13]** = 0, the maximum acceptable value for **Positive delta [0x09-0x0A]** is:
(256 steps per revolution * 32,768 revolutions) - 1 step – 256 steps (i.e. 1 revolution for safety reasons) = 8 388 351
See further examples in the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.

WARNING
Every time **Distance per revolution [0x00]** and **Preset [0x12-0x13]** parameters are changed, **Positive delta [0x09-0x0A]** and **Negative delta [0x0B-0x0C]** values have to be checked carefully. Each time you change the value in **Distance per revolution [0x00]** you must then update the value in **Preset [0x12-0x13]** in order to define the zero of the shaft as the system reference has now changed.
After having changed the parameter in **Preset [0x12-0x13]** it is not necessary to set new values for travel limits as the Preset function calculates them automatically and initializes again the positive and negative limits according to the values set in **Positive delta [0x09-0x0A]** and **Negative delta [0x0B-0x0C]** items. For a detailed explanation see on page 45.

Negative delta [0x0B-0x0C]

[Registers 12-13, Unsigned32, rw]
This value is used to calculate the maximum backward (negative) limit the device is allowed to reach starting from the preset value. Should it happen that the maximum backward limit is reached, a signalling is activated through the **SW limit switch -** status bit of the **Status word [0x01]** (the bit is forced high).
The parameter is expressed in pulses.
SW limit switch - = **Preset [0x12-0x13]** - **Negative delta [0x0B-0x0C]**.
For further information please refer to the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.
Default = 134 213 631 (min. = 0, max. = 134 213 631)

WARNING
Please mind the maximum acceptable value for this item depends on the set scaling.

EXAMPLE
When **Distance per revolution [0x00]** = 4,096 and **Preset [0x12-0x13]** = 0, the maximum acceptable value for **Negative delta [0x0B-0x0C]** is:

(4,096 steps per revolution * 32,768 revolutions) – 1 step – 4,096 steps (i.e. 1 revolution for safety reasons) = 134 213 631 (see the default value)
When **Distance per revolution [0x00]** = 1,024 and **Preset [0x12-0x13]** = 0, the maximum acceptable value for **Negative delta [0x0B-0x0C]** is:
(1,024 steps per revolution * 32,768 revolutions) – 1 step – 1,024 steps (i.e. 1 revolution for safety reasons) = 33 553 407
When **Distance per revolution [0x00]** = 256 and **Preset [0x12-0x13]** = 0, the maximum acceptable value for **Negative delta [0x0B-0x0C]** is:
(256 steps per revolution * 32,768 revolutions) – 1 step – 256 steps (i.e. 1 revolution for safety reasons) = 8 388 351
See further examples in the "6.3 Distance per revolution, Preset, Max delta pos / Positive delta and Max delta neg / Negative delta" section on page 45.

WARNING
Every time **Distance per revolution [0x00]** and **Preset [0x12-0x13]** parameters are changed, **Positive delta [0x09-0x0A]** and **Negative delta [0x0B-0x0C]** values have to be checked carefully. Each time you change the value in **Distance per revolution [0x00]** you must then update the value in **Preset [0x12-0x13]** in order to define the zero of the shaft as the system reference has now changed.
After having changed the parameter in **Preset [0x12-0x13]** it is not necessary to set new values for travel limits as the Preset function calculates them automatically and initializes again the positive and negative limits according to the values set in **Positive delta [0x09-0x0A]** and **Negative delta [0x0B-0x0C]** items. For a detailed explanation see on page 45.

Jog speed [0x0D]

[Register 14, Unsigned16, rw]
This parameter contains the maximum speed the device is allowed to reach when using the **Jog +** and **Jog -** functions (see the **Control Word [0x2A]** parameter). The parameter is expressed in revolutions per minute (rpm). See also the "Jog: speed control" section on page 43.
Default = 2000 (min. = 1, max. = 3000)

Work speed [0x0E]

[Register 15, Unsigned16, rw]
This parameter contains the maximum speed the device is allowed to reach in automatic work mode (movements are controlled using the **Start** and **Stop** commands -see the **Control Word [0x2A]** parameter- and are performed in order to reach the position set in **Target position [0x2B-0x2C]**). The parameter

is expressed in revolutions per minute (rpm). See also the "Positioning: position and speed control" section on page 44.
Default = 2000 (min. = 1, max. = 3000)

Code sequence [0x0F]

[Register 16, Unsigned16, rw]
It sets whether the position value output by the device increases (count up information) when the shaft rotates clockwise (0) or counter-clockwise (1). Clockwise and counter-clockwise rotations are viewed from shaft side.
0 = count up information with clockwise rotation (default)
1 = count up information with counter-clockwise rotation

WARNING
Changing this value causes also the position calculated by the controller to be necessarily affected. Therefore it is compulsory to set a new value in the **Preset [0x12-0x13]** parameter and then check the values set next to the **Positive delta [0x09-0x0A]** and **Negative delta [0x0B-0x0C]** items.

Offset [0x10-0x11]

[Registers 17-18, Integer32, ro]
This variable defines the difference between the position value transmitted by the device and the real position: real position – preset. The value is expressed in pulses.
Default = 0

Preset [0x12-0x13]

[Registers 19-20, Integer32, rw]
Use this parameter to set the Preset value. The Preset function is meant to assign a desired value to a physical position of the axis. The chosen physical position will get the value set next to this item and all the previous and the following positions will get a value according to it. The preset value will be set for the position of the axis in the moment when the value is entered. The preset value is activated when the bit 11 **Setting the preset** in the **Control Word [0x2A]** register is switched from logic level low ("0") to logic level high ("1").
Default = 0 (min. = -268 435 456, max. = 268 435 456)

NOTE
We suggest activating the preset when the actuator is in stop. See the **Setting the preset** command on page 151.

WARNING
A new value has to be set in the **Preset [0x12-0x13]** registers every time the **Distance per revolution [0x00]** value is changed. After having entered a new value in **Preset [0x12-0x13]** it is not necessary to set new values for travel limits as the Preset function calculates them automatically and initializes again the positive and negative limits according to the values set in the **Positive delta [0x09-0x0A]** and **Negative delta [0x0B-0x0C]** items. For a detailed explanation see on page 45.

Jog step length [0x14]

[Register 21, Unsigned16, rw]
If the incremental jog function is enabled (bit 4 **Incremental jog** in the **Control Word [0x2A]** = 1), the activation of the bits **Jog +** and **Jog -** causes a single step toward the positive or negative direction having the length, expressed in pulses, set next to this item to be executed at rising edge; then the actuator stops and waits for another command.
Default = 1000 (min. = 1, max. = 10000).

Extra commands register [0x29]

[Register 42, Unsigned16, rw]
Byte structure of the **Extra commands register [0x29]**:

byte	MSB			LSB		
bit	15	...	8	7	...	0
	msb		lsb	msb		lsb

Byte 0
bit 0: Not used.

Control by PC
bit 1: This function is reserved only for use and service of Lika Electronic engineers (only used with Modbus service port).

bits 2 ... 7 Not used.
Byte 1 Not used.

WARNING
For safety reasons the **Extra commands register [0x29]** holding register parameter is not stored in the memory. So it is required to be set after each power-on.

Control Word [0x2A]

[Register 43, Unsigned16, rw]

This variable contains the commands to be sent in real time to the Slave in order to manage it.

Byte structure of the **Control Word [0x2A]** register:

byte	MSB			LSB		
bit	15	...	8	7	...	0
	msb		lsb	msb		lsb

Byte 0

Jog +

bit 0

If the bit 4 **Incremental jog** = 0, as long as **Jog +** = 1, the Slave moves toward the positive direction; otherwise if the bit 4 **Incremental jog** = 1, the activation of this bit causes a single step toward the positive direction having the length, expressed in pulses, set next to the **Jog step length [0x14]** item to be executed at rising edge; then the actuator stops and waits for another command. Velocity, acceleration and deceleration are performed according to the values set next to the **Jog speed [0x0D]**, **Acceleration [0x07]** and **Deceleration [0x08]** parameters respectively. For a detailed description of the jog control see on page 43.

Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.

Jog -

bit 1

If the bit 4 **Incremental jog** = 0, as long as **Jog -** = 1, the Slave moves toward the negative direction; otherwise if the bit 4 **Incremental jog** = 1, the activation of this bit causes a single step toward the negative direction having the length, expressed in pulses, set next to the **Jog step length [0x14]** item to be executed at rising edge; then the actuator stops and waits for another command. Velocity, acceleration and deceleration are performed according to the value set next to the **Jog speed [0x0D]**, **Acceleration [0x07]** and **Deceleration [0x08]** parameters respectively. For a detailed description of the jog control see on page 43.

Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.

Stop

bit 2

If set to "=1" the Slave is allowed to execute the movements as commanded. If, while the unit is running, this bit switches to "=0" then the Slave must stop executing the deceleration procedure set in **Deceleration [0x08]**. For an immediate halt in the movement, use the bit 7 **Emergency**.

Alarm reset

bit 3

This command is used to reset an alarm condition of the Slave but only if the fault condition has ceased. In a normal work condition this bit is set to "0". Setting this bit to "1" causes the normal work status of the device to be restored. The normal work status is resumed by switching this bit from "0" to "1".

Please note that should the alarm be caused by wrong parameter values (see **Machine data not valid** and **Wrong parameters list [0x08-0x09]**), the normal work status can be restored only after having set proper values. The **Flash memory error** alarm cannot be reset.

Incremental jog

bit 4

If set to "=0", the activation of the bits **Jog +** and **Jog -** causes the Slave to move as long as **Jog +** / **Jog -** = 1. Setting this bit to 1 the incremental jog function is enabled, that is: the activation of the bits **Jog +** and **Jog -** causes a single step toward the positive or negative direction having the length, expressed in pulses, set next to the **Jog step length [0x14]** item to be executed at rising edge; then the actuator stops and waits for another command.

bit 5

Not used.

Start

bit 6

When it is set to "=1" the device moves in order to reach the set target position (see **Target position [0x2B-0x2C]** on page 152). For a complete description of the position control see on page 44.

Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.

Emergency

bit 7

This bit has to be normally high ("=1") otherwise it will cause the device to stop immediately. For a normal stop (not immediate) respecting the set deceleration see above the bit 2 **Stop**. At power-on it is forced low (="0") for safety reasons. Switch it high ("=1") to resume normal operation.

Byte 1

Watch dog enable

bit 8

Setting the **Watch dog enable** bit to "=1" causes the Watch dog function to be enabled; setting the **Watch dog enable** bit to "=0" causes the Watch dog function to be disabled. When the Watch dog function is enabled, if the device does not receive a message from the Server within 1 second, the system forces an alarm condition (the **Watch dog** alarm is invoked to appear as soon as the Modbus network communication is restored). The Watch dog function is a safety timer that uses a time-out to detect loop or deadlock conditions. For instance, should the serial communication be cut off while a command is still active and running -a jog command for example- the Watch dog safety system immediately takes action and commands a safety stop of the device; furthermore an alarm is triggered.

Save parameters

bit 9

Data is saved on non-volatile memory at each rising edge of the bit; in other words, save is performed each time this bit is switched from logic level low ("0") to logic level high ("1"). Always save the new values after setting in order to store them in the non-volatile memory permanently. Should the power supply be turned off all data that has not been saved previously will be lost!

Load default parameters

bit 10

The default parameters (they are set at the factory by Lika Electronic engineers to allow the operator to run the device for standard operation in a safe mode) are restored at each rising edge of the bit; in other words, the default parameters loading operation is performed each time this

bit is switched from logic level low ("0") to logic level high ("1"). The complete list of machine data and relevant default parameters preset by Lika Electronic engineers is available on page 170.

Always save the new values after setting in order to store them in the non-volatile memory permanently. Should the power supply be turned off all data that has not been saved previously will be lost!

WARNING

The unit has been adjusted by performing a full-load mechanical running test; thence default values which has been set refer to a device running in such condition. Furthermore they are intended to ensure a standard and safe operation which not necessarily results in a smooth running and an optimum performance. Thus to suit the specific application requirements it may be advisable and even necessary to enter new parameters instead of the factory default settings; in particular it may be necessary to change velocity, acceleration, deceleration and gain values.

Setting the preset

bit 11 — It sets the current position to the value set next to the **Preset [0x12-0x13]** registers. The operation is performed at each rising edge of the bit, i.e. each time this bit is switched from logic level low ("0") to logic level high ("1"). We suggest activating the preset when the actuator is in stop. For more information refer to page 146.

Release axis torque

bit 12 — When the axis has reached the commanded position, it maintains the torque.
If set to "=0", when the axis is in position, the PWM is kept active.
If set to "=1", when the axis is in position, the PWM is deactivated (the torque is released).

bits 13 ... 15 — Not used.

WARNING

For safety reasons the **Control Word [0x2A]** holding register parameter is not stored in the memory. So it is required to be set after each power-on.

Target position [0x2B-0x2C]

[Registers 44-45, Integer32, rw]
It sets the position to be reached, otherwise referred to as commanded position. When the **Start** command is sent while the **Stop** and **Emergency** bits are "=1" and the alarm condition is off, the device moves in order to reach the target position set next to this item.
As soon as the axis is within the tolerance window limits set next to the Position window [0x01] register, the bit 8 **Target position reached** in the Status word [0x01] goes high ("=1"). When the position is within the tolerance window limits set next to the Position window [0x01] register, after the delay set next to the Position window time [0x02] item, the bit 0 **Axis in position** in the Status word [0x01] goes high ("=1").
For more information refer also to the "Positioning: position and speed control" section on page 44.
Default = 0 (min. = 0, max. = within maximum positive limit / maximum negative limit)

NOTE
Position override function
It is possible to change the target position value even on the fly, while the device is still reaching a previously commanded target position and without sending a new **Start** command. To do this, just set a new target value in the Target position [0x2B-0x2C] registers. See also on page 44.

NOTE
Jog +, **Jog -** and **Start** functions cannot be enabled simultaneously. For instance: if a **Jog +** command is sent to the Slave while it is moving to the target position, the jog command will be ignored; if **Jog +** and **Jog -** commands are sent simultaneously, the device will not move or, if already moving, it will stop its movement.
When the Watch dog function is enabled (**Watch dog enable** in Control Word [0x2A] is set to "=1"), should the device be disconnected from the Modbus network while it is moving (for instance because of a broken cable or a faulty wiring), the device stops moving immediately and activates the Watch dog alarm bit (the alarm is invoked to appear as soon as the Modbus network communication is restored).

WARNING
For safety reasons the Target position [0x2B-0x2C] holding register parameter is not stored in the memory. So it is required to be set after each power-on.

NOTE
Save the set values using the **Save parameters** function.
Should the power be turned off all data not saved will be lost!

8.14.2 Input Register parameters

The **Input Register** parameters are accessible for reading only; to read the value set in an input register parameter use the **04 Read Input Register** function code (reading of multiple input registers); for any further information on the implemented function codes refer to the "8.13.1 Implemented function codes" section on page 130.

Alarms register [0x00]

[Register 1, Unsigned16, ro]
This variable is meant to show the alarms currently active in the device.
Structure of the alarms byte:

byte	MSB			LSB		
bit	15	...	8	7	...	0
	msb		lsb	msb		lsb

The available alarm error codes are listed hereafter:

Byte 0

Machine data not valid

bit 0 — One or more parameters are not valid, set proper values to restore the normal work condition. See the list of the wrong parameters in the **Wrong parameters list [0x08-0x09]** item.

Flash memory error

bit 1 — Internal error, it cannot be restored.

Counting error

bit 2 — For safety reasons, both the absolute encoder position and the incremental encoder position are read and saved to two separate registers. If any difference between the values in the registers is found the error is signalled.

Following error

bit 3 — The difference between the real position and the theoretical position is greater than the value set in the **Max following error [0x03-0x04]** parameter; we suggest reducing the work speed.

Axis not synchronized

bit 4 — Internal error, it cannot be restored.

Target not valid

bit 5 — The set target position is over the maximum travel limits.

Emergency

bit 6 — Bit 7 **Emergency** in **Control Word [0x2A]** has been forced to low value (0); or alarms are active in the unit.

Overcurrent

bit 7 — Motor overcurrent.

Byte 1

Electronics Overtemperature

bit 8 — The temperature of the MOSFETs detected by an internal probe is exceeding the maximum ratings (see **Temperature value [0x07]** on page 158). Please wait some minutes for the actuator to cool down. Ensure that the operating temperature is within the range.

Motor Overtemperature

bit 9 — The temperature of the motor detected by an internal probe is exceeding the maximum ratings (see **Temperature value [0x07]** on page 158). Please wait some minutes for the actuator to cool down. Ensure that the operating temperature is within the range.

Undervoltage

bit 10 — The power supply voltage is under the minimum ratings allowed. Please ensure that the power supply voltage is within the range.

Watch dog

bit 11 — When the Watch dog function is enabled (bit 8 **Watch dog enable** in **Control Word [0x2A]** is set to "=1"), if the device does not receive a message from the Server within 1 second, the system forces an alarm condition (the **Watch dog** alarm bit is activated). The alarm is invoked to appear as soon as the Modbus network communication is restored. The Watch dog function is a safety timer that uses a time-out to detect loop or deadlock conditions. For instance, should the serial communication be cut off while a command is still active and running -a jog command for example- the Watch dog safety system immediately takes action and commands a safety stop of the device; furthermore an alarm is triggered.

bits 12 and 13 — Not used.

Hall sequence

bit 14 — An error has been detected in the Hall sensors commutation sequence.

Overvoltage

bit 15 — The power supply voltage is over the maximum ratings allowed. Please ensure that the power supply voltage is within the range.
If the alarm is triggered during the braking operation, please consider the counter-electromotive force (back EMF). To prevent such situation from arising, decrease the deceleration ramp or evaluate attentively the characteristics of the 24V power supply pack (capacitor module).

To reset a faulty condition use the **Alarm reset** command, Control Word [0x2A] bit 3. In a normal work condition the **Alarm reset** bit is set to "0". Setting the bit to "1" causes the normal work status of the device to be restored. The normal work status is resumed by switching this bit from "0" to "1". This command resets the alarm but only if the fault condition has ceased.

Please note that should the alarm be caused by wrong parameter values (see Machine data not valid and Wrong parameters list [0x08-0x09]), the normal work status can be restored only after having set proper values. The Flash memory error alarm cannot be reset.

Status word [0x01]

[Register 2, Unsigned16, ro]
This register contains information about the current state of the device.
Byte structure of the Status word [0x01] register:

byte	MSB			LSB		
bit	15	...	8	7	...	0
	msb		lsb	msb		lsb

Byte 0

Axis in position

bit 0 — The value is "=1" when the device reaches and keeps the set position (Target position [0x2B-0x2C]) for the time set next to the Position window time [0x02] register. It is kept active until the position error is lower than Position window [0x01]. For

	further information please refer to the "Positioning: position and speed control" section on page 44.
bit 1	Not used.
Axis enabled	
bit 2	It shows the enabling status of the motor. This bit is "=1" when the motor is enabled, that is: PWM is active and the axis is under closed-loop control (while reaching a target position or using a jog, for instance). It is "=0" when the motor is disabled, that is when the controller is off after a positioning or jog movement or because of an alarm condition.
SW limit switch +	
bit 3	The value is "=1" should it happen that the device reaches the maximum positive limit (positive limit switch). For more information see the **Positive delta [0x09-0x0A]** parameter.
SW limit switch -	
bit 4	The value is "=1" should it happen that the device reaches the maximum negative limit (negative limit switch). For more information see the **Negative delta [0x0B-0x0C]** parameter.
Alarm	
bit 5	The value is "=1" when an alarm occurs, see details in the **Alarms register [0x00]** variable.
Axis running	
bit 6	The value is "=0" when the device is not moving. The value is "=1" while the device is moving.
Executing a command	
bit 7	The value is "=0" when the controller is not executing any command. The value is "=1" while the controller is executing a command.
Byte 1	
Target position reached	
bit 8	The value is "=1" when the device reaches the target position set next to the **Target position [0x2B-0x2C]** item (it is within the limits set next to the **Position window [0x01]**). The bit is kept active until a new **Target position [0x2B-0x2C]** value or the **Alarm reset** command are sent. For more information

refer also to the "Positioning: position and speed control" section on page 44.

bits 9 ... 11 — Not used.

PWM saturation

bit 12 — The current supplied for controlling the motor phases has reached the saturation point and cannot be increased further. The motor operation is affected by excessive dynamics or something is jamming the movement.

bits 13 ... 15 — Not used.

Current position [0x02-0x03]

[Registers 3-4, Integer32, ro]
Current position of the device expressed in pulses.

Current velocity [0x04]

[Register 5, Integer16, ro]
Speed of the device expressed in revolutions per minute [rpm], updated at every second.

Position following error [0x05-0x06]

[Registers 6-7, Integer32, ro]
This variable contains the difference between the target position and the current position step by step. If this value is greater than the one set in the **Max following error [0x03-0x04]** parameter, then the **Following error** alarm is triggered and the unit stops. The value is expressed in pulses.

Temperature value [0x07]

[Register 8, Integer16, ro]
This variable shows both the temperature of the motor and the temperature of the electronics as detected by internal probes. The value is expressed in °C (Celsius degrees). The minimum detectable temperature is -20°C.
The meaning of the 16 bits in the register is as follows:

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MSB								LSB							
Major number								Minor number							
Temperature of the motor								Temperature of the electronics							

Value 18 1A hex in hexadecimal notation corresponds to the binary representation 0001 1000 0001 1010 and has to be interpreted as: temperature of the motor = 24°C; temperature of the electronics = 26°C.

Wrong parameters list [0x08-0x09]

[Registers 9-10, Unsigned32, ro]

The operator has set invalid data and the **Machine data not valid** alarm has been triggered. This variable is meant to show the list of the wrong parameters, respecting the structure shown in the following table.

Please note that the normal work status can be restored only after having set proper values.

Bit	Parameter
0	Not used
1	Distance per revolution [0x00]
2	Acceleration [0x07]
3	Deceleration [0x08]
4	Positive delta [0x09-0x0A]
5	Negative delta [0x0B-0x0C]
6	Jog speed [0x0D]
7	Work speed [0x0E]
8	Code sequence [0x0F]
9	Preset [0x12-0x13]
10	Jog step length [0x14]
11	Kp position loop [0x05]
12	Ki position loop [0x06]
13	Position window time [0x02]
14	Max following error [0x03-0x04]
15	Not used

Motor voltage [0x0A]

[Register 11, Unsigned16, ro]

It shows the motor voltage expressed in millivolts (mV).

Current value [0x0B]

[Register 12, Unsigned16, ro]
This variable shows the value of the current absorbed by the motor (rated current). The value is expressed in mA (milliamperes).

Hall [0x0C]

[Register 13, Unsigned16, ro]
This function is reserved only for use and service of Lika Electronic engineers.

Duty cycle [0x0D]

[Register 14, Unsigned16, ro]
This function is reserved only for use and service of Lika Electronic engineers.

DIP switch baud rate [0x0E]

[Register 15, Unsigned16, ro]
This is meant to show the data transmission rate (baud rate) of the serial port the RD6 unit is equipped with; the data transmission rate has to be set through the provided DIP switch. In this model the baud rate DIP switch has fixed value and is not accessible to the user.

DIP switch node ID [0x0F]

[Register 16, Unsigned16, ro]
This is meant to show the node address set in the RD6 unit; the node address has to be set through the provided DIP switch. In this model the node ID DIP switch has fixed value and is not accessible to the user.

SW Version [0x10]

[Register 17, Unsigned16, ro]
This is meant to show the software version of the DRIVECOD unit.
The meaning of the 16 bits in the register is as follows:

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MSB								LSB							
Major number								Minor number							

Value 01 02 hex in hexadecimal notation corresponds to the binary representation 00000001 00000010 and has to be interpreted as: version 1.2.

HW Version [0x11]

[Register 18, Unsigned16, ro]
This is meant to show the hardware version and model of the DRIVECOD unit.
The meaning of the 16 bits in the register is as follows:

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
not used		Motor size		Interface				not used				Hardware version			

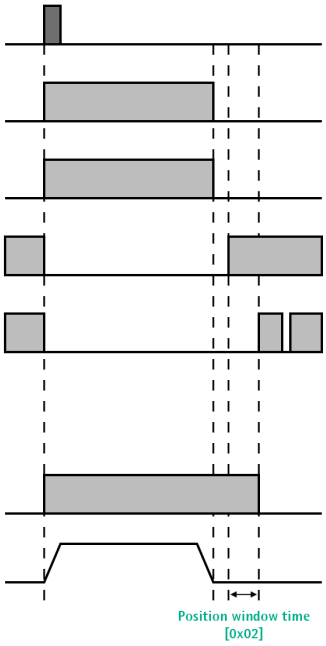
where:

00 ... 03	= hardware version
08 .. 11	= RD6 model equipped with the following interface: 0x00 = MODBUS RTU; 0x01 = Profibus; 0x02 = CANopen; 0x03 = POWERLINK; 0x04 = EtherCAT; 0x05 = MODBUS TCP; 0x06 = EtherNet/IP; 0x07 = Profinet
12	01h = 157 W motor size
13	01h = 250 W motor size

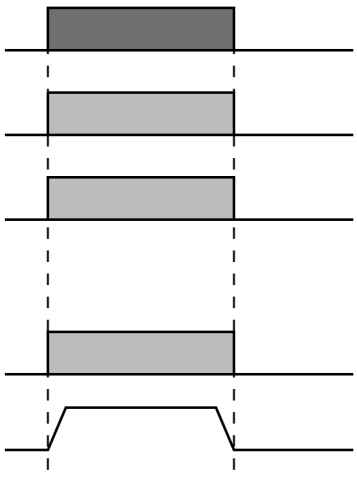
Value 10 01 hex in hexadecimal notation corresponds to the binary representation 0001 0000 0000 0001 and has to be interpreted as follows: hardware version 1 (LSbyte = 0x01); RD6 157 W model with MODBUS RTU interface (MSbyte = 0x10).

NOTE
Save the set values using the **Save parameters** function.
Should the power be turned off all data not saved will be lost!

EXAMPLE 1

Positioning	RD6
Start (bit 6 - Control Word [0x2A])	
Executing a command (bit 7 - Status word [0x01])	
Axis running (bit 6 - Status word [0x01])	
Target position reached (bit 8 - Status word [0x01])	
Axis in position (bit 0 - Status word [0x01])	
Axis enabled (bit 2 - Status word [0x01]: PWM is active)	
Theoretical velocity	

EXAMPLE 2

Jog +	RD6
Jog + (bit 0 - Control Word [0x2A])	
Executing a command (bit 7 - Status word [0x01])	
Axis running (bit 6 - Status word [0x01])	
Axis enabled (bit 2 - Status word [0x01]: PWM is active)	
Theoretical velocity	

8.15 Exception codes

When a Client device sends a request to a Server device it expects a normal response. One of four possible events can occur from the Master's query:

- If the Server device receives the request without a communication error and can handle the query normally, it returns a normal response.
- If the Server does not receive the request due to a communication error, no response is returned. The client program will eventually process a timeout condition for the request.
- If the Server receives the request, but detects a communication error (parity, CRC, ...), no response is returned. The client program will eventually process a timeout condition for the request.
- If the Server receives the request without a communication error, but cannot handle it (for example, if the request is to read a non-existent output or register), the Server will return an exception response informing the Client about the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

FUNCTION CODE FIELD: in a normal response, the Server echoes the function code of the original request in the function code field of the response. All function codes have a most significant bit (msb) of 0 (their values are all below 80 hexadecimal). In an exception response, the Server sets the msb of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response. With the function code's msb set, the client's application program can recognize the exception response and can examine the data field for the exception code.
DATA FIELD: in a normal response, the Server may return data or statistics in the data field (any information that was requested in the request). In an exception code, the Server returns an exception code in the data field. This defines the Server condition that caused the exception.

For any information on the available exception codes and their meaning refer to the "MODBUS Exception Responses" section on page 48 of the "MODBUS Application Protocol Specification V1.1b" document.

8.16 Programming examples

Hereafter are some examples of both reading and writing parameters. All values are expressed in hexadecimal notation.

8.16.1 Using the 03 Read Holding Registers function code

EXAMPLE 1
Request to read the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9) to the Slave having the node address 1.

Request PDU
[01][03][00][07][00][02][75][CA]
where:
[01] = Slave address
[03] = **03 Read Holding Registers** function code
[00][07] = starting address (**Acceleration [0x07]** parameter, register 8)
[00][02] = number of requested registers
[75][CA] = CRC

Response PDU
[01][03][04][00][0A][00][0A][5A][36]
where:
[01] = Slave address
[03] = **03 Read Holding Registers** function code
[04] = number of bytes (2 bytes for each register)
[00][0A] = value of register 8 **Acceleration [0x07]**, 00 0A hex = 10 dec
[00][0A] = value of register 9 **Deceleration [0x08]**, 00 0A hex = 10 dec
[5A][36] = CRC

Acceleration [0x07] parameter (register 8) contains the value 00 0A hex, i.e. 10 in decimal notation; **Deceleration [0x08]** parameter (register 9) contains the value 00 0A hex, i.e. 10 in decimal notation.

8.16.2 Using the 04 Read Input Register function code

EXAMPLE 1
Request to read the **Current position [0x02-0x03]** parameter (registers 3 and 4) to the Slave having the node address 1.

Request PDU
[01][04][00][02][00][02][D0][0B]
where:
[01] = Slave address
[04] = **04 Read Input Register** function code
[00][02] = starting address (**Current position [0x02-0x03]** parameter, register 3)
[00][02] = number of requested registers
[D0][0B] = CRC

Response PDU
[01][04][04][00][00][2F][F0][E7][F0]
where:
[01] = Slave address
[04] = **04 Read Input Register** function code
[04] = number of bytes (2 bytes for each register)
[00][00] = value of register 3 **Current position [0x02-0x03]**, 00 00 hex = 0 dec
[2F][F0] = value of register 4 **Current position [0x02-0x03]**, 2F F0 hex = 12272 dec
[E7][F0] = CRC

Current position [0x02-0x03] parameter (registers 3 and 4) contains the value 00 00 2F F0 hex, i.e. 12272 in decimal notation.

EXAMPLE 2
Request to read the **Alarms register [0x00]** variable (register 1) to the Slave having the node address 1.

Request PDU
[01][04][00][00][00][01][31][CA]
where:
[01] = Slave address
[04] = **04 Read Input Register** function code
[00][00] = starting address (**Alarms register [0x00]** variable, register 1)
[00][01] = number of requested registers

[31][CA] = CRC

Response PDU
[01][04][02][00][81][79][50]
where:
[01] = Slave address
[04] = **04 Read Input Register** function code
[02] = number of bytes (2 bytes for each register)
[00][81] = value of register 1 **Alarms register [0x00]**, 00 81 hex = 0000 0000 1000 0001 bin
[79][50] = CRC

This means that in the **Alarms register [0x00]** variable (register 1) the bits 0 and 7 are active (logic level high = 1), i.e. (see on page 154): **Machine data not valid** and **Emergency**.

8.16.3 Using the 06 Write Single Register function code

EXAMPLE 1
Request to write the value 00 96 hex (= 150 dec) in the **Acceleration [0x07]** parameter (register 8) of the Slave having the node address 1.

Request PDU
[01][06][00][07][00][96][B8][65]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][07] = address of the register (**Acceleration [0x07]** parameter, register 8)
[00][96] = value to be set in the register
[B8][65] = CRC

Response PDU
[01][06][00][07][00][96][B8][65]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][07] = address of the register (**Acceleration [0x07]** parameter, register 8)
[00][96] = value set in the register
[B8][65] = CRC

The value 00 96 hex, i.e. 150 in decimal notation, is set in the **Acceleration [0x07]** parameter (register 8).

EXAMPLE 2
Request to write the value 00 84 hex in the **Control Word [0x2A]** variable (register 43) of the Slave having the node address 1.

Request PDU
[01][06][00][2A][00][84][A8][61]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][2A] = address of the register (**Control Word [0x2A]** variable, register 43)
[00][84] = value to be set in the register
[A8][61] = CRC

Response PDU
[01][06][00][2A][00][84][A8][61]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][2A] = address of the register (**Control Word [0x2A]** variable, register 43)
[00][84] = value set in the register
[A8][61] = CRC

The value 00 84 hex = 0000 0000 1000 0100 in binary notation is set in the **Control Word [0x2A]** variable (register 43). In other words, the **Stop** and **Emergency** bits are forced to the logical level high (bit 2 = 1; bit 7 = 1): the unit is ready to execute the motion command as requested.

EXAMPLE 3
Request to write the value 0A 80 hex in the **Control Word [0x2A]** variable (register 43) of the Slave having the node address 1.

Request PDU
[01][06][00][2A][0A][80][AF][02]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][2A] = address of the register (**Control Word [0x2A]** variable, register 43)
[0A][80] = value to be set in the register
[AF][02] = CRC

Response PDU
[01][06][00][2A][0A][80][AF][02]
where:
[01] = Slave address
[06] = **06 Write Single Register** function code
[00][2A] = address of the register (**Control Word [0x2A]** variable, register 43)
[0A][80] = value set in the register
[AF][02] = CRC

The value 0A 80 hex = 0000 0010 1000 0000 in binary notation is set in the **Control Word [0x2A]** variable (register 43). In other words, the device is forced in stop (bit 2 **Stop** = 0) but not in emergency condition (bit 7 **Emergency** = 1); furthermore data save is requested (bit 9 **Save parameters** = 1).

8.16.4 Using the 16 Write Multiple Registers function code

EXAMPLE 1
Request to write the values 150 and 100 in the parameters **Acceleration [0x07]** (register 8) and **Deceleration [0x08]** (register 9) of the Slave having the node address 1.

Request PDU
[01][10][00][07][00][02][04][00][96][00][64][53][8E]
where:
[01] = Slave address
[10] = **16 Write Multiple Registers** function code
[00][07] = starting address (**Acceleration [0x07]** parameter, register 8)
[00][02] = number of requested registers
[04] = number of bytes (2 bytes for each register)
[00][96] = value to be set in the register 8 **Acceleration [0x07]**, 00 96 hex = 150 dec
[00][64] = value to be set in the register 9 **Deceleration [0x08]**, 00 64 hex = 100 dec
[53][8E] = CRC

Response PDU
[01][10][00][07][00][02][F0][09]
where:
[01] = Slave address
[10] = **16 Write Multiple Registers** function code
[00][07] = starting address (**Acceleration [0x07]** parameter, register 8)
[00][02] = number of written registers
[F0][09] = CRC

The value 00 96 hex, i.e. 150 in decimal notation, is set in the **Acceleration [0x07]** parameter (register 8); the value 00 64 hex, i.e. 100 in decimal notation, is set in the **Deceleration [0x08]** parameter (register 9).

9 Default parameters list

EtherCAT®

Parameters list	Default value		
2200 Control Word	0		
2201-00 Target Position	0		
2204-00 Distance per revolution PPR	4,096		
2205-00 Position tolerance P	1		
2206-00 Settling time ms	0		
2207-00 Max following error P	50,000		
2208-00 Proportional gain	80		
2209-00 Integral gain	10		
220A-00 Acceleration rev/s^2	10		
220B-00 Deceleration rev/s^2	10		
220C-00 Max delta pos P	134 213 631		
220D-00 Max delta neg P	134 213 631		
220E-00 Jog speed rpm	2,000		
220F-00 Work speed rpm	2,000		
2210-00 Count direction	0		
2211-00 Preset P	0		
2212-00 Step jog P	1,000		

Modbus

Parameters list	Default value		
Distance per revolution [0x00] PPR	4,096		
Position window [0x01] P	1		
Position window time [0x02] ms	0		
Max following error [0x03-0x04] P	50,000		
Kp position loop [0x05]	80		
Ki position loop [0x06]	10		
Acceleration [0x07] rev/s^2	10		
Deceleration [0x08] rev/s^2	10		
Positive delta [0x09-0x0A] P	134 213 631		
Negative delta [0x0B-0x0C] P	134 213 631		
Jog speed [0x0D] rpm	2,000		
Work speed [0x0E] rpm	2,000		
Code sequence [0x0F]	0		
Preset [0x12-0x13] P	0		
Jog step length [0x14] P	1,000		
Control Word [0x2A]	0		
Target position [0x2B-0x2C]	0		

This page intentionally left blank

Document release	Release date	Description	HW	SW	XML file	Interface
1.0	25.07.2016	First issue	H2	S2 S3	V1	V1.0 V1.1
1.1	17.11.2016	General update, Modbus examples	H2	S3	V2	V1.2
1.2	17.01.2019	General update, new Modbus interface	H2	S4	V2	3.1.0.3

Dispose separately

Lika Electronic
Via S. Lorenzo, 25 • 36010 Carrè (VI) • Italy

Tel. +39 0445 806600
Fax +39 0445 806699

info@lika.biz • www.lika.biz