HIGH PRECISION GEARED

SERVO MOTORS

CANOPEN

COMMUNICATION GUIDE

ROZUM ROBOTICS

RDrive

RDRIVE BY ROZUM ROBOTICS

INTRODUCTION

This guide is intended for specialists who implement control of RDrive servo motors based on CANOpen communication. The document describes the following:

- the basics of the CANOpen interface implemented for RDrive servo motors
- commands for controlling motion of RDrive servos
- parameters and settings of RDrive servos that can be read from or written to the RR object dictionary by means of CANOpen messages
- application cases detailing CANOpen communication for setting and executing single motions and motion trajectories, reading and writing servo settings, and reading actual servo parameters
- emergency messages in the CANOpen communication of the RDrive servo motors

WARNING SIGNS AND THEIR MEANINGS

Below are the warning symbols used throughout the manual and explanations of their meanings.

The sign denotes important information that is not directly related to safety, but that the user should be aware of.

The sign indicates important safety precautions the user should follow.

TABLE OF CONTENTS

1 CANOPEN IMPLEMENTATION FOR RDRIVE SERVOS

For CANOpen communication and control, RDrive servo motors rely on the CANOpen protocol stack and its lower-level implementation—the Controller Area Network (CAN). Within the context of the Open Systems Interconnection (OSI) model, CAN covers the physical and data link layers, whereas CANOpen implements the other above-lying ones.

1.1 The physical and data link layers

A CAN bus provides physical infrastructure for data communication. In RDrive servo motors, the physical CAN connection is provided by the CAN_{HIGH} and CAN_{LOW} wires run through the input flange (see Figure 1-1).

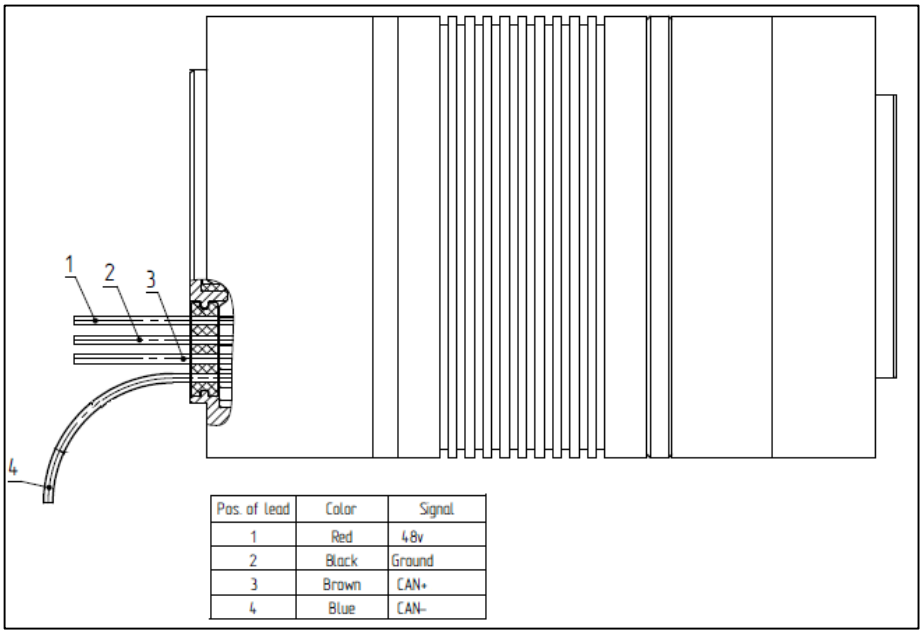

Figure 1-1: **RDrive cable gland with CAN wires**

1.1.1 CAN frame

CAN frames are basic means of communication in the CANOpen interface. These can either be **data frames** that transmit data from CAN nodes or **remote frames** that request transmission of data.

RDrive servos support only the standard data CAN frame format with an 11-bit identifier (see **Figure 1-2**).

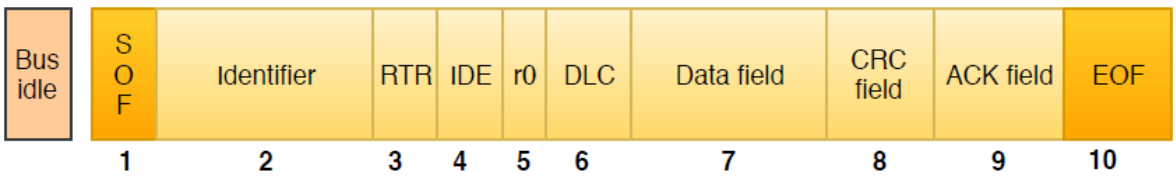

Figure 1-2: The standard CAN data frame

No.	Description
1	Start of the frame bit.
2	Identifies the priority of the message (11 bits long). The message with the lowest value of the identifier has the highest priority.
3	Remote transmission request bit to differentiate between a data frame and a remote frame. For a data frame, the value is set to 0, whereas, for a remote frame, it is set to 1.
4	Identifier extension (in the standard CAN frame, the bit is set to 0, which means no identifier extension is used).
5	Reserved bit.
6	Specifies the number of transmitted data bytes (4 bits long).
7	Contains actual data transmitted over the network (up to 64 bytes).
8	For detecting transmission errors (16 bits long).
9	For acknowledging correct reception of the data frame (2 bits long).
10	End of the frame bit.

1.1.2 CANOpen frame identifier

In CANOpen messages, the 11-bit CAN frame identifier is a COB-ID. Each COB-ID comprises:

- a **4-bit function code** identifying the type of the communication object (e.g., NMT, SYNC, EMCY, SDO) (see **Section 1.2**)
- a **7-bit Node ID** indicating the network node (servo) involved in the communication (see **Figure 1-3**).

With the 7-bit Node ID, the maximum number of nodes in a CANOpen network is limited to 127. The Node ID 0 is reserved for network management messages.

Function code				***Node address***						
bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Figure 1-3: **The COB-ID structure**

1.2 The CANOpen stack for RDrive servos

CANOpen covers the top five layers of the OSI model: network (addressing, message forwarding), transport (reliability, flow control), session (synchronization), presentation (standardized data encoding and representation), and application. To implement CANOpen for RDrive servos, the following protocol stack is used:

- SDO—Service data object
- NMT—Network state management
- Heartbeat

- EMCY—Emergency
- SYNC—Synchronization
- Timestamp

Table 1-1: **The CANOpen protocol stack**

CANOpen protocol	Function
NMT	The NMT protocol enables issuing commands to change device states (see **Section 1.2.1**): • Initialization • Stopped • Pre-operational • Operational An NMT message is a 2-byte CAN frame. The first byte is the command specifier, and the second one—the Node ID of the device assigned to execute the command. If the Node ID is set to 0, the NMT command is addressed to all devices in the network.
Heartbeat	The Heartbeat protocol is intended to monitor availability of the nodes on a CAN bus. At specified intervals, a Heartbeat producer transmits a message to one or more Heartbeat consumers. In case a consumer receives no Heartbeat message within a specified interval, it generates a Heartbeat event entailing further actions—e.g., resetting a node or indicating an error. For details, refer to **Section 1.2.2**. In addition, RDrive servos generate Heartbeat messages at the Bootloader stage during initialization (see **Section 3.1**). These Heartbeat messages contain the Node ID of the respective servo as preset by the manufacturer and are meant to signal that the servo has joined the CAN network as one of its nodes.
SYNC	The SYNC protocol allows for synchronizing nodes in a CAN network. A SYNC message is a single CAN frame with the default identifier—128. A SYNC producer broadcasts SYNC messages over the network at predefined intervals.
Timestamp	The Timestamp protocol specifies a time offset in milliseconds prior to starting a servo motion queue. A Timestamp message is a single 6-byte data frame.
EMCY	The EMCY protocol allows for indicating internal errors of network devices. An EMCY message is transmitted in connection with error events and can contain up to 8 data bytes.
SDO	The SDO protocol ensures direct access to the object dictionary. There are two types of SDO messages: for reading commands and parameters from a node and writing them to a node.

1.2.1 NMT messages and servo states

NMT frames are 2-bit messages containing an NMT command code and the ID of the network node to which the command is addressed.

COB-ID	Data byte 0	Data byte 1
0x000	Requested state (*NMT command code)*	Addressed node (*Node ID*)

Figure 1-4: **The NMT frame format**

All NMT command codes have the same COB-ID format—0x000. The NMT command code is one of the values as indicated in **Table 1-2** (Column 1), depending on the requested state (Column 2).

Table 1-2: **NMT command codes and their meanings**

NMT command code	Meaning
0x01	Go to the “operational” state
0x02	Go to the “stopped” state
0x80	Go to the “pre-operational” state
0x81	Go to the “reset node” state
0x82	Go to the “reset communication” state

The states of an RDrive servo as a CANOpen node can change as illustrated in the diagram in **Figure 1-5**.

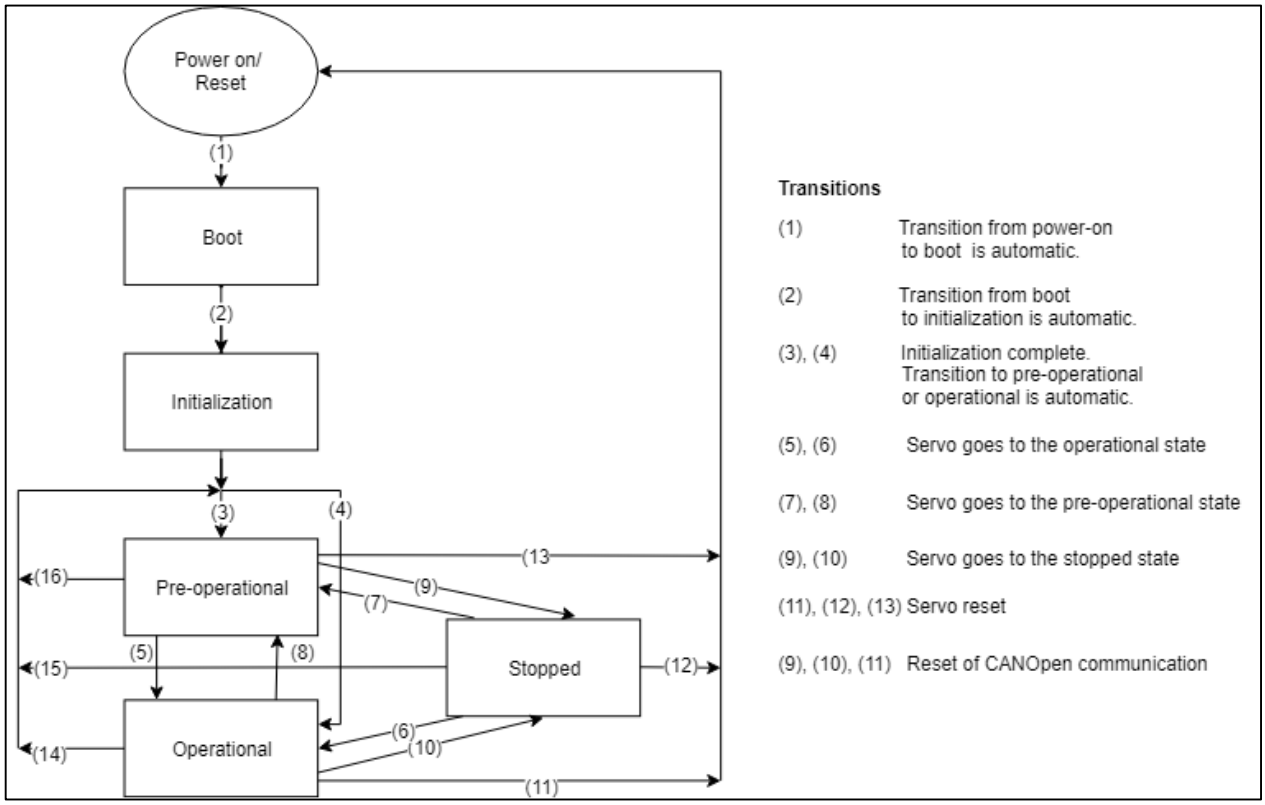

Figure 1-5: **State transition diagram for RDrive servos**

Table 1-3 describes the NMT states of RDrive servos and lists available types of communication for each of the states.

Table 1-3: NMT states and available types of communication

State	Details	SDO	PDO	NMT
Reset node	Complete reset of servos. No communication is possible. After a reset, the corresponding node is back to its power-on state.	-	-	-
Reset communication	Reset of CANOpen communication. No communication is possible.	-	-	-
Initialization	The servo initializes: transmits a bootup Heartbeat message, performs a self-test to check the firmware integrity, and uploads the firmware.	-	-	yes
Pre-operational	In the state, servos are available for communication, but cannot execute commands.	yes	yes	yes
Operational	In this state, it is possible to transmit all types of CANOpen messages. Servos are both available for communication and can execute commands.	yes	yes	yes
Stopped	No communication, except for Heartbeat and NMT messages.	no	no	yes

1.2.2 Heartbeat messages

Heartbeat frames are a special type of CANOpen messages that a network node (a servo) transmits at regular intervals to confirm its availability for communication.

COB-ID	Data byte 0
0x700 + node ID	NMT state code

Figure 1-6: The Heartbeat frame format

The communication status is a 1-byte value contained in the data byte part of the Heartbeat frame. The value corresponds to one of the NMT state codes listed in **Table 1-4** below.

Table 1-4: **NMT state codes and their descriptions**

NMT state code	State	Description
0x00	Bootup (initialization)	Sent once, when the firmware is initializing.
0x02	Bootloader	Bootloader state
0x03	Collision	Indicates address collision, which is equivalent to the stopped state. To change the collision state, a reboot is required.
0x04	Stopped	Sending or receiving EMCY/ SDO/ SYNC messages is not possible.
0x05	Operational	Normal operation.
0x7F	Pre-operational	A servo cannot operate properly, but can hold its current position or be released.

1.2.3 The Timestamp frame format

The Timestamp command specifies a delay before starting the motion queue of a servo motor. The command frame format is as shown below in **Figure 1-7**.

COB-ID	Data Byte 0	Data Byte 1	Data Byte 2	Data Byte 3	Data Byte 4	Data Byte 5
0x100	Timestamp in milliseconds (LSB)					

Figure 1-7: **The Timestamp frame format**

2 ROZUM ROBOTICS OBJECT DICTIONARY

An **object dictionary** is the centerpiece of CANOpen communication. It is a table with all communication-related and process data, including servo control commands, settings, and parameters.

Each entry in the dictionary is a communication object, identified with a 16-bit index. More complex objects are additionally identified with an 8-bit subindex. Access to all objects in the dictionary is through SDOs.

2.1 RDrive servo commands in the RR object dictionary

The **Table 2-1** is a command reference, listing commands for controlling RDrive servo that can be accessed from the Rozum Robotics object dictionary.

Table 2-1: **Control commands for RDrive servos in the Rozum Robotics object dictionary**

Index	Subindex	Attr.	Name	Bytes	Data type	Min.	Max.	Data format	Comment
0x2010	1	WO	Stop and release	1	U8	-	-		When you issue the command, the servo stops without retaining its last position. **Attention!** Affected by an external force (e.g., inertia), the servo may continue rotating for a while or start moving in the opposite direction. **Note:** For the function, you can set any value, accounting for the data type.
0x2010	2	WO	Stop and freeze	1	U8	-	-		When you issue the command, the servo stops, retaining its last position. **Note:** For the function, you can set any value, accounting for the data type.
0x2011	1	W	Run self-test	1	U8	-	-	0–Lo-range self-test ±2° of the flange 1–Hi-range self-test ±180° of the flange	The command is optional. Use the command to check servo motion after startup (see **Section 3.1**). **Note:** Once the command is issued, the servo shifts by ±5 degrees. Therefore, take measures to avoid injuries and damage.
0x2011	1	R	Get self-test status	1	U8	-	-		The command enables reading the results of the servo self-testing (including the status of the system integrity check at initialization). **Possible responses:** TEST_STATUS_OK = 0 (when the servo passed the self-tests successfully) TEST_STATUS_FAILED (when the servo failed the motion test) TEST_STATUS_SECURE_TEST_FAILED (when the servo failed the automatic system integrity check during initialization) TEST_STATUS_NOT_EXEC (the motion self-test cannot be executed) TEST_STATUS_BUSY (the self-testing is still in progress)
0x2012	1	WO	Set current	4	F32	-	-	Current, A	The command enables setting the current at which a specific servo should move.
0x2012	2	WO	Set brake current	4	F32	-	-	Current, A	The command enables setting the current at which a specific servo should decelerate.
0x2012	3	WO	Set velocity	4	F32	-	-	Velocity, °/sec.	Use the command to set the velocity with which a specific servo should move at the manufacturer-preset current. The manufacturer-preset current is the maximum current value in accordance with servo motor specifications. **Note:** To set a lower max current limit, use the “Set velocity with current limit” command.

0x2012	4	WO	Set position	4	F32	-	-	Position, °	Use the command to set the position that a specific servo should reach as a result of executing the command. **Note:** Executing the command, the servo moves to the specified position at the manufacturer-preset max velocity and current.
0x2012	5	WO	Set velocity with current limit	8	F32 F32			Velocity, °/sec Current limit, A	Use the command when you need a specific servo to rotate at desired velocity and at a current that is lower than the manufacturer-preset limit (in accordance with servo specifications). **Note:** When current is set to 0, servos ignore the limit during motion.
0x2012	6	WO	Set position with velocity and current limits	12	F32 F32 F32			Position, ° Velocity limit, °/sec Current limit, A	Use the command when you want a specific servo to move to a specific position at a current and velocity that are lower than manufacturer-preset values. The manufacturer-preset values are values in accordance with servo motor specifications. **Note:** When current is set to 0, servos ignore the limit during motion.
0x2012	7	WO	Set duty	4	F32	-1.0	1.0	Duty, [-1.0; 1.0]	The command enables setting the duty mode, which is the percentage of the input voltage to be supplied to a specific servo. The default (off) value is 0.0.
0x2013	subindex for the parameter from Table 2-4	RO	Custom parameter read	4	F32	-	-	Actual servo parameter (variable value)	Use the command when you need to read the actual value of a single servo parameter (one of those listed in **Table 2-4**).
0x2014	1	RO	Parameter array read		ARRAY[F32] (The array size depends on the “Set value array” command)	-	-		Use the command when you need to read the actual values of more than one servo parameters (as listed in **Table 2-4**). The command will return the array of variables as preset using the “Parameter array format read/write” command.
0x2015	1	RW	Parameter array format read/write		ARRAY[U8]	-	-		The command enables setting an array of multiple parameters that the user can subsequently read with the “Parameter array read” function.
0x2200	1	WO	Add PT point	8	F32 U32	-	-	Position, ° Time, ms	The command enables creating PT (position-time) motion points for a specific servo. PT points define the following: - what position the servo should reach - how long the movement from the previous position to the specified one should take The created PT points are arranged into a motion queue, which represents the motion trajectory of the servo.
0x2200	2	WO	Add PVT point	12	F32 F32 U32	-	-	Position, ° Velocity, °/sec Time, ms	The command enables creating PVT (position-velocity-time) motion points for a specific servo. PVT points define the following: - what position the servo should reach - with what velocity the servo should move to the specified position - how long the movement from the previous position to the specified one should take

									The created PVT points are arranged into a motion queue, which represents the motion trajectory of the servo.
0x2202	1	WO	Clear motion queue/N last points from the queue	4	U32	-	-		The command enables: a) to remove N last motion points from the motion queue of a specific servo Value != 0 clears the specified number of points b) to clear the entire motion queue Value = 0 clears the motion queue and stops the motor
0x2202	2	RO	Num. of the points in the queue	4	U32				The command enables reading the number of the PT (PVT) points in the motion queue of a specific servo.
0x2202	3	RO	Num. of the free cells in the queue (free space)	4	U32				The command enables reading the number of cells that are available for adding new PT (PVT) points to the motion queue of a specific servo.
0x2203	1	WO	Timings calculate PVAT: calculate	32	F32 F32 F32 U32 F32 F32 F32 U32			Start position, ° Start velocity, °/sec Start acceleration, °/sec^2 Start time, ms End position, ° End velocity, °/sec End acceleration, °/sec^2 End time, ms	The command enables calculating how long it will take a specific servo to move from the start position to the end one with specified motion parameters (e.g., velocity, acceleration).
0x2203	2	RO	Timings calculate PVAT: get results	4	U32			Time, ms	The command enables reading the result of executing the “Timings calculate PVAT: calculate” command (see above).
0x2203	5	RO	Timings calculate PVAT: get results	24	F32 F32 F32 F32 F32 F32			α0 α1 α2 α3 α4 α5	The command returns the polynom coefficients (including time in milliseconds) based on the calculations with the “Timings calculate PVAT: calculate” command.
0x2207	2	RO	Get maximum velocity	4	F32				The command enables reading the maximum velocity with which a specific servo can move at the current moment. The return is the lowest of the three limits—the user-defined one (see “Set velocity with current limit”), the maximum current as per servo specifications, and the maximum velocity calculated based on the supply voltage.
0x2208	1	WO	Set zero position	4	F32			Flange degrees	The command enables setting the actual position (in degrees) of a specific servo to any value defined by the user. For instance, when the actual servo position is 101 degrees and the new setting based on the command is 25 degrees, the servo is assumed to be positioned at 25 degrees. **Note:** The setting is volatile: it is no longer valid after a reset or a power outage.

0x2208	2	WO	Set zero position and save to EEPROM	4	F32			Flange degrees	The command enables setting the actual position (in degrees) of a specific servo to any value defined by the user and saving it to the EEPROM memory.

Notes:

- The *read* commands return the requested parameters with *ACK/NACK*, whereas the response to all *write* commands is *ACK/NACK*.

Glossary:

PVAT—position-velocity-acceleration-time
PVT—position-velocity-time
PT—position-time
R—read
W—write
RO—read only access
RW—read and write access
WO—write only access
ARRAY—array of variables

Table 2-2: Data formats

Type	Size	Format
U8	Unsigned byte	
U16	Unsigned word	Least significant bit (LSB)
U32	Unsigned double word	Least significant bit (LSB)
BOOL	Same as U8	
F24	Truncated single precision float-point variable	Single precision float-point variable, IEEE 754. The 4th least significant byte is truncated.
F32	Single precision float-point variable	Single precision float-point variable, IEEE 754

2.2 RDrive servo settings

Apart from the control commands listed in **Table 2-1**, the Rozum Robotics object dictionary for RDrive servo motors includes settings that represent important operating characteristics of the servos. The operations of reading and writing the settings are executed using SDO frames. The representation of the data part in the frame is as indicated in the "Data type" column (see also **Table 2-2**).

Table 2-3: **The RDrive settings in the RR object dictionary**

Index	Subindex	Attr.	Name	Bytes	Data type	Min	Max	Default	Data format	Comments
0x1009	0	RO	Manufacturer hardware version		String					Unique identifier of the controller + Hardware type + Hardware revision
0x100A	0	RO	Manufacturer software version (FW)		String					Major and minor firmware versions and build timestamp
0x1016	1	RW	Master Heartbeat timeout		U32				ms	Maximum master heartbeat waiting time
0x1017	0	RW	Heartbeat time		U16				ms	The interval between the Heartbeat messages of a device
0x1F89	0	RW	Boot time	2	U32	500	50,000	2,000	ms	Bootloader delay time
0x2000	0	RO	Error status bits	10	ARRAY					
0x2100	0	RW	CAN Node ID	1	U8	1	127	123		Device ID on the CAN bus
0x2101	0	RW	CAN bitrate	1	ENUM8	0	8	2		CAN bitrate of the device CAN_BAUD_1000 (Kbit) = 0, CAN_BAUD_500=2, CAN_BAUD_250=3, CAN_BAUD_125=4, CAN_BAUD_100=5, CAN_BAUD_50=6, CAN_BAUD_20=7, CAN_BAUD_10=8, CAN_BAUD_2000=9
0x2102	0	RO	Power-on counter	4	U32					To count total power-on cycles
0x2103	0	RO	System time counter	8	U64				ms	System time from the device startup
0x2300	1	RW	Power (consumption) limit	4	F32				W	User-defined maximum power consumption (when set to 0, the user limit is disabled)

0x2300	2	RW	Maximum acceleration	4	F32				degrees/sec^2	User-defined maximum acceleration of the servo's output flange (when set to 0, the user limit is disabled)
0x2300	3	RW	Maximum velocity	4	F32				degrees/sec	User-defined maximum velocity, deg/sec (when set to 0, the user limit is disabled)
0x2300	5	RW	Maximum position error	4	F32				degrees	User-defined maximum position error (when set to 0, the limit is disabled)
0x2300	6	RW	Min position	4	F32				degrees	Minimum multi-turn position
0x2300	7	RW	Max position	4	F32				degrees	Maximum multi-turn position
0x2300	8	RW	Zero position	4	F32				degrees	The absolute zero position of the servo (within the range of [0:360] degrees)
0x2300	9	RW	Voltage min	4	F32				volts	User-defined maximum working voltage (when set to 0, the limit is disabled)
0x2300	A	RW	Voltage max	4	F32				volts	User-defined minimum working voltage (when set to 0, the limit is disabled)
0x2300	B	RW	Current phase limit	4	F32				A	User-defined maximum phase current limit (when set to 0, the limit is disabled)
0x2300	C	RW	Current input limit	4	F32				A	User-defined maximum input current limit (when set to 0, the limit is disabled)
0x2301	0	RW	Position limit control is enabled	1	BOOL					Enables/disables the "Minimum multi-turn position" and "Maximum multi-turn position" limits

2.3 Reading RDrive servo parameters

The SDO protocol also enables reading **RDrive servo parameters**—variables representing actual operating values (e. g., current, position, temperature). The table below lists servo parameters available for reading together with their descriptions and subindices for SDO frames.

Table 2-4: **RDrive servo parameters available for reading**

Sub-index	Parameter	Name	Description
1	APP_PARAM_POSITION	Pos: gear	Multi-turn position of the servo's output flange
2	APP_PARAM_VELOCITY	Vel: gear	Velocity of the servo's output flange in degrees per second
3	APP_PARAM_POSITION_ROTOR	Pos: rotor	Single-turn position of the rotor
4	APP_PARAM_VELOCITY_ROTOR	Vel: rotor	Velocity of the rotor in RPM
5	APP_PARAM_POSITION_GEAR_360 //0x05	Pos: gear360	Single-turn position of the servo's output flange
7	APP_PARAM_CURRENT_INPUT	Current: input	Input current
9	APP_PARAM_VOLTAGE_INPUT	Voltage: input	Input voltage
11	APP_PARAM_CURRENT_PHASE	Current: phase	Phase current
12	APP_PARAM_TEMPERATURE_ACTUATOR	Temperature: actuator	Temperature of the servo's stator (°C)
13	APP_PARAM_TEMPERATURE_ELECTRONICS	Temperature: electronics	Temperature of the servo's PCB (°C)
28	APP_PARAM_CONTROLLER_VELOCITY_ERROR	Ctrl: vel: error	Velocity error of the controller (RPM)
29	APP_PARAM_CONTROLLER_VELOCITY_SETPOINT	Ctrl: vel: setpoint	Velocity setpoint of the controller (RPM)
30	APP_PARAM_CONTROLLER_VELOCITY_FEEDBACK	Ctrl: vel: feedback	Velocity feedback of the controller (i.e., velocity of the rotor in RPM)
31	APP_PARAM_CONTROLLER_VELOCITY_OUTPUT //0X1F	Ctrl: vel: output	Velocity output of the controller (control current in Amperes)
32	APP_PARAM_CONTROLLER_POSITION_ERROR	Ctrl: pos: error	Position error of the controller in degrees
33	APP_PARAM_CONTROLLER_POSITION_SETPOINT	Ctrl: pos: setpoint	Position setpoint of the controller (i.e., flange angle in degrees)
34	APP_PARAM_CONTROLLER_POSITION_FEEDBACK	Ctrl: pos: feedback	Position feedback of the controller (i.e., position of the flange in degrees)
35	APP_PARAM_CONTROLLER_POSITION_OUTPUT	Ctrl: pos: output	Position output of the controller (in rotor RPM)

36	APP_PARAM_CONTROL_MODE	Control mode	CONTROL_MODE_OFF = 0, ///< Motor is not controlled CONTROL_MODE_DUTY = 1, ///< Duty control CONTROL_MODE_VEL = 2, ///< Velocity control CONTROL_MODE_CURRENT = 3, ///< Current control CONTROL_MODE_BRAKE = 4, ///< Brake control CONTROL_MODE_POS = 5, ///< Position control CONTROL_MODE_MOTION = 6, ///< Motion control CONTROL_MODE_OPENLOOP = 7, ///< Open-loop current control CONTROL_MODE_INACTIVE = 8 ///< Motor is not controlled

3 APPLICATION CASES

In the section, there are a number of application cases demonstrating a sequence of CANOpen messages required to cause a servo motor to execute commands, as well as to write or read parameters and settings.

Color designations:

- **green**—a read command
- **purple**—a write command

3.1 Case 1. Servo motor initialization

Initializing, an RDrive servo motor goes through the following **three** stages:

Stage 1. The *Bootloader* stage.

Once power is supplied to the servo motor, it starts and transmits automatically the *BOOTUP Heartbeat* message:

CANOpen protocol	Frame format (Hexadecimal)
Heartbeat	712 # 2

Stage 2. Once the **boot time** expires, the servo starts uploading the firmware and transmits the *BOOTUP Heartbeat* message:

CANOpen protocol	Frame format (Hexadecimal)
Heartbeat	712 # 0

Then, the servo motor runs a system integrity test. If the ***test results are ok***, the device sends either the *OPER* or the *PRE-OPER Heartbeat* message:

CANOpen protocol	Frame format (Hexadecimal)	State
Heartbeat	712 # 5	OPER

or

CANOpen protocol	Frame format (Hexadecimal)	State
Heartbeat	712 # 7F	PRE-OPER

Step 3. (Optional) The user sends the **Run self-test** command to run a motion self-test.

Format:

CANOpen protocol	Command	SDO format (Hexadecimal)	Response
SDO	**Run self-test**	**(INDEX: 0x2011 SUB:1)** **0**	**SDO response**

Once the Run self-test command is issued, the servo motor shifts by ±5 degrees. Take measures to prevent damage to the motion system or the accessories attached to the servo.

Then, the user can request the self-test status with the **Get self-test status** command.

Format:

CANOpen protocol	Command	SDO format (Hexadecimal)	Response
SDO	Get self-test status	(INDEX: 0x2011 SUB:1)	SDO response + data: x [U8] (TEST_STATUS_OK = 0, TEST_STATUS_FAILED, TEST_STATUS_SECURE_TEST_FAILED, TEST_STATUS_NOT_EXEC, TEST_STATUS_BUSY)

If the user issues the command while the servo motor is still running a self-test, the command returns **TEST_STATUS_BUSY**. If the servo motor fails the self-test, the user gets an error status response (e.g., **TEST_STATUS_FAILED).**

3.2 Case 2. Rotating the servo motor with the angular velocity of 10,050 RPM

To set the angular velocity for servo motor rotation, use the **Set velocity** SDO command.

Format:

CANOpen protocol	Command	SDO format (Hexadecimal)	Response
SDO	Set velocity	(INDEX: 0x2012 SUB:3) cd cc 20 41	SDO response

In case the servo motor fails to rotate with the preset velocity, the command returns an SDO error code.

3.3 Case 3. Turning the motor shaft to the position of 90.456° in 5,004 ms (spline interpolation)

The motion trajectory of an RDrive servo motor is a spline. To turn a servo motor to the required position within the specified time, complete the following sequence of steps:

Step 1. Define a spline motion trajectory by setting a number of PT (position and time) points. Use the **Add PT point** command.

Format:

CANOpen protocol	Command	SDO format (Hexadecimal)	Response
SDO	**Add PT point**	**(INDEX: 0x2201 SUB: 1)** **79 e9 b4 42 8c 13 0 0**	**SDO response**

Step 2. Send a sync command to set a time offset before starting the servo motor motion: **0 ms** to start immediately or **200 ms** to start in 200 ms.

Sending a sync command when the servo motion queue is not empty and is being processed (moving) will generate an error.

Format:

CANOpen protocol	Command	Frame format (Hexadecimal)
TIMESTAMP	**TS_START**	**100 # c8 0 0 0 0 0** *(200 ms delay)*

TIMESTAMP commands return no response.

3.4 Case 4. Turning the motor shaft to the position of 90.456° in 5,000 ms

The case is similar to case #3, except for the time offset—it is set to 0. Therefore, the sequence of actions should be as below:

Step 1. Using the **Add PT point** command, define a spline motion trajectory by setting a number of PT points (position and time values).

Format:

CANOpen protocol	Command	SDO format (Hexadecimal)	Response
SDO	**Add PT point**	**(INDEX: 0x2201 SUB: 1)** **79 e9 b4 42 88 13 0 0**	**SDO response**

Step 2. Send a sync command to set a time offset to **0 ms**—to start immediately.

Sending a sync command when the servo motion queue is empty generates an error.

Format:

CANOpen protocol	Command	Frame format (Hexadecimal)
TIMESTAMP	TS_START	**100 # 0 0 0 0 0 0** *The timestamp is 0 ms -> immediate start.*

TIMESTAMP commands return no response.

3.5 Case 5. Executing a motion trajectory

In this example, we will execute **the following motion trajectory**: 34.567° in 123 ms, -321.001° in 8,900 ms with the end velocity of 2.0 RPM, -1° in 5,432 ms.

Step 1. Using the sequence of three SDO commands as demonstrated below, set a motion trajectory.

CANOpen protocol	Command	SDO format (Hexadecimal)	Response
SDO	Add PT point	**(INDEX: 0x2201 SUB: 1)** **9c 44 0a 42 7b 0 0 0**	SDO response
SDO	Add PVT point	**(INDEX: 0x2201 SUB: 2)** **21 80 a0 c3 c4 22 0 0**	SDO response
SDO	Add PT point	**(INDEX: 0x2201 SUB: 1)** **0 0 80 bf 38 15 0 0**	SDO response

Step 2. Send a sync command to set a time offset before starting the servo motor motion.

CANOpen protocol	Command	Frame format (Hexadecimal)
TIMESTAMP	TS_START	**100 # 7A 03 0 0 0 0** ***The time offset is 890 ms.***

TIMESTAMP commands return no response.

3.6 Case 6. Reading a single servo motor parameter

In this example, we demonstrate how to read the **APP_PARAM_CURRENT_INPUT** parameter (actual input current of the servo motor).

To read a single servo motor parameter (any parameter from Table 2-4), use the **Custom parameter read** command.

Format:

CANOpen service	Command	SDO format (Hexadecimal)	Response (Hexadecimal)
SDO	**Custom parameter read**	**(INDEX: 0x2013 SUB:7)**	**SDO response + data:** **x x x x [FLOAT32]** **0 0 20 c1** ***The returned parameter value is -10.0.***

The response to the SDO command contains the required reading.

3.7 Case 7. Reading multiple servo motor parameters

In this example, we demonstrate how to read **multiple parameters**:

- **APP_PARAM_POSITION** (actual servo position)
- **APP_PARAM_VOLTAGE_INPUT** (actual input voltage of the servo motor)
- **APP_PARAM_CURRENT_INPUT** (actual input current of the servo motor)

In all, it is possible to read up to 48 parameters at a time.

To read multiple servo parameters, complete the following sequence of steps:

Step 1. Set an array of parameters to read using the **Parameter array format read/write** command.

*There are 10 bytes reserved for parameter array (as each byte contains 8 bits, we get 10*8 = 80 bits for parameter markers).*

Format (setting the 2nd, 4th, and 23rd markers):

CANOpen Service	Command	SDO format (Hexadecimal)	Response (Hexadecimal)
SDO	**Parameter array format read/write**	**(INDEX: 0x2015) 14 0 80 0 0 0 0 0 0 0**	**SDO response**

Step 2. Read the preset parameter array with the **Parameter array read** command.

Format (the 2nd, 4th, and 23rd markers are set):

CANOpen service	Command	SDO format (Hexadecimal)	Response (Hexadecimal)	Data contents
SDO	**Parameter array read**	**(INDEX: 0x2014 SUB: 1)**	**SDO response**	**12 bytes [FLOAT32] x3** **x x x x z z z z y y y y** **79 e9 f6 42 0 0 c8 c2 1d 5a ac c0** **The 2nd parameter (APP_PARAM_POSITION) is 123,456.** **The 4th parameter (APP_PARAM_VOLTAGE_INPUT) is -100.00.** **The 23rd parameter (APP_PARAM_CURRENT_INPUT) is -5.386.**

In its response, the command returns the requested array of parameter values.

3.8 Case 8. Turning a servo motor off

Typically, a servo motor turns off without a command. However, to avoid damage to the system or injuries, it is advisable to stop the servo motor as described below.

To stop a servo, the user can, at his or her own discretion, send an NMT command (**NMT_CMD_STOP**) or any of the following SDO Commands: **Stop and release** or **Stop and freeze.**

Format:

CANOpen service	Command	Frame format (Hexadecimal)	Response (Hexadecimal)
NMT	NMT_CMD_STOP	**12 # 2 (for one servo motor only)** **0 # 2 (for all devices on the bus)**	*No response* *(However, you can learn whether a node is in the stopped state from its Heartbeat messages)*
SDO	**Stop and release**	**SDO Format: (INDEX: 0x2010 SUB: 1) 0**	**SDO response**
SDO	**Stop and freeze**	**SDO Format: (INDEX: 0x2010 SUB: 2) 0**	**SDO response**

In case the servo fails to stop, the command returns an SDO error code.

4 EMERGENCY MESSAGES

Whenever an error occurs on a CAN network node, an emergency (EMCY) message is generated. Such messages are transmitted to other network devices with high priority and contain an error code, an error register, and additional device-specific information. The frame format for the EMCY messages is as shown below:

COB-ID	Data byte 0	Data byte 1	Data byte 2	Data byte 3	Data byte 4	Data byte 5	Data byte 6	Data byte 7
0x080 + Node ID*	Error Code (Standard error codes according to CiA DS-301 and DS-401)		Error Register	Error status bits (Internal indication of the error condition)	Manufacturer error info (the parameter of the error)			
	LSB	MSB			Data format: F32			

Figure 6-1: The EMCY frame format

Table 4-1 lists errors of servo motors together with their error codes, error registers, and error status bits.

Table 4-1: **Servo motor errors**

Category	Error	Error Bit [CO_EM_errorStatusBits]	Error code [CO_EM_errorCodes]	Description
CAN	TX bus off	CO_EM_CAN_TX_BUS_OFF	CO_EMC_BUS_OFF_RECOVERED	CAN bus disconnected
	TX or RX bus passive	CO_EM_CAN_TX_BUS_PASSIVE	CO_EMC_CAN_PASSIVE	
	TX or RX bus warning	CO_EM_CAN_BUS_WARNING	CO_EMC_NO_ERROR	
	Master Heartbeat Timeout	CO_EM_HEARTBEAT_CONSUMER	CO_EMC_GENERIC	Master Heartbeat lost
Servo motion	Position limit error	CO_EM_MOTION_ERROR	CO_EMC401_POS_LIMIT	Servo has moved outside of the programmed position limits
	Following position error is too big	-	CO_EMC401_POS_FLW_ERROR	
		-	CO_EMC401_POS_FLW_STATIC_ERROR	
Servo hardware	UV	CO_EM_HW_VOLT_LO	CO_EMC401_IN_VOLT_LOW	Undervoltage
	OV	CO_EM_HW_VOLT_HI	CO_EMC401_IN_VOLT_HI	Overvoltage
	MOSFET Driver Error	CO_EM_HARDWARE_ERROR	CO_EMC401_PWRCTRL_ERROR	
	PCB Temperature Error	CO_EM_TEMPERATURE_INTRNL_ERROR	CO_EMC401_POWER_TEMP_OVER	Servo PCB controller error
	Stator Temperature Error	CO_EM_TEMPERATURE_INTRNL_ERROR	CO_EMC401_MOTOR_TEMP_OVER	Motor temperature error
	OC	CO_EM_HW_CUR_LIMIT	CO_EMC401_PWRCTRL_ERROR	Overcurrent
Servo self-test	Fail configuration parameters	CO_EM_FLT_CONFIG_CONSTRAINT	-	
	Fail CRC of the configuration	CO_EM_FLT_CONFIG_CRC	-	
	Stator temperature sensor error	CO_EM_FLT_NTC	CO_EMC_DEVICE_SPECIFIC	
	Current sensor A error	CO_EM_FLT_CS0	CO_EMC_DEVICE_SPECIFIC	
	Current sensor B error	CO_EM_FLT_CS1	CO_EMC_DEVICE_SPECIFIC	
	MOSFET driver error	CO_EM_FLT_DRIVER	CO_EMC_DEVICE_SPECIFIC	
	Voltage sensor error	CO_EM_FLT_VS0	CO_EMC_DEVICE_SPECIFIC	
	Motor encoder disconnected	CO_EM_FLT_ENC_M_OFF	CO_EMC_DEVICE_SPECIFIC	
	Gear encoder disconnected	CO_EM_FLT_ENC_G_OFF	CO_EMC_DEVICE_SPECIFIC	
	Motor Encoder fault	CO_EM_FLT_ENC_M_STUP_CRC	CO_EMC_DEVICE_SPECIFIC	
		CO_EM_FLT_ENC_M_LEVEL		
		CO_EM_FLT_ENC_M_SIG		
	Gear Encoder fault	CO_EM_FLT_ENC_G_STUP_CRC	CO_EMC_DEVICE_SPECIFIC	
		CO_EM_FLT_ENC_G_LEVEL		
		CO_EM_FLT_ENC_G_SIG		