Servosila Device Reference

Device Type: Servo Controller (0xA020192)

Revision F (May 2023)

Table of Contents

Configuration Parameters

Configuration - Datasheet

The "Datasheet" section contains parameters that characterize the key components of an electric drive: a motor, encoder(s), Hall sensors and a gearbox. The information is either found in datasheets supplied by manufacturers of those components, or measured using the controller's auto-configuration capabilities.

The "Datasheet" section is an input into computation of various parameters of control laws that determine performance of the electric drive. Note that not all of the "Datasheet" parameters are mandatory for every configuration of electric drives. Watch out for the units in which those parameters are defined since conversion of the units might be necessary for proper configuration. Instead of filling out the "Datasheet" section directly, one might want to use a "Spreadsheet" tool that comes with the accompanying software. The tool properly provisions the "Datasheet" parameters into the controller while also computing parameters of various control laws.

Start configuring an electric drive by filling-out the "Datasheet" section, manually or using the "Spreadsheet" tool, or by launching the controller's auto-configuration procedure that fills out much of the section automatically.

#	Parameter	Units	Description	CANopen
1	Motor Type (Brushless or Brushed)	-	• 0: Brushless • 1: Brushed	UINT16, 0x2000, 0x01, rw
2	Maximum Limit on Continuous Current (Line-to-Line)	A	This is a maximum current the motor can handle indefinitely without over-heating. Do not confuse this with a short-term peak current which could be much higher. The "Maximum Limit on Continuous Current" is one of the most critical performance and safety parameters in the "Datasheet" section. On one hand, the parameter defines the maximum continuous torque the electric drive can produce. The higher this limit is set, the more electric current is allowed to be driven through the motor by the controller, the more torque the motor produces, the better the dynamics of the electric drive is. On the other hand, driving more current through the motor means generating more heat in the motor's winding. The heat is what burns electric motors. This means that making a mistake and setting this parameter too high might have fatal consequences for the motor. Setting this parameter too low would mean that the motor is not used to its full capacity in terms of torque. In short, it is important to set this parameter right. However, this parameter is not what can be experimentally determined or measured by the controller itself during an auto-configuration procedure. Such a procedure would have to burn a few motors to figure out what maximum phase-to-phase electric current a particular model of the motor can handle continuously with a stalled shaft. Obviously, this is not a practical approach. If one has to guess the limit, start from a very conservative low value, and gradually increase it until the motor starts heating up too much. Some of the motors will burn before showing any signs of heating up. This means that this parameter has to be taken from a datasheet supplied by the motor's manufacturer.	FLOAT32, 0x2000, 0x02, rw

			Note that if a particular application does not require all the torque the motor can produce, it would be wise to set the limit lower than a nominal value suggested by the manufacturer. This would establish a safety margin at the expense of torque. If the design goal is to push the motor to its limit in terms of torque and dynamics, it is still wise to start the commissioning procedure at a lower limit, and gradually raise the limit while observing how the motor handles the heat, particularly in stalled situations such as braking or servo modes. Note that a motor is operating in its worse possible situation as far as heat dissipation is concerned whenever the motor is producing maximum torque (=maximum current) while stalled (=no motion), for example, in a braking mode or in a low-speed direct drive mode. Note a difference between a "maximum current" and a "nominal current/maximum continuous current" characteristics of motors. Since manufacturers of the motors use varying terminology, it is easy to get confused and make a mistake. The term "maximum current" is what manufacturers often use to define an electric current the motor can handle for relatively short periods of time, typically a few seconds. This is not what needs to be configured here. What we are looking for is a limit on "continuous" current, often defined as the "nominal" or "rated" current in datasheets. This is a phase-to-phase electric current that a stalled motor can handle indefinitely without any damage due to heat. To summarize the point, be extra careful with datasheet entries called "maximum current". Those entries might define a current that is way above a "continuous current/nominal current" limit that we are looking for. HINT: If a brushless motor's datasheet defines both "nominal power" and "nominal voltage" parameters, it is possible to derive implied "nominal current" using a formula given below. However, please do not confuse "maximum power" with "nominal power" as those might differ by a factor of five or more. Maximum Continuous Current (Line-to-Line) = Nominal Power (W) / Nominal Voltage (V) * 1.414213562	
3	Poles Number (Rotor Poles)	-	The Poles Number parameter should be taken from the motor's datasheet, or it can be determined experimentally. Note that the number of rotor poles is always an EVEN number since magnet poles always come in pairs. Note that some motor manufacturers specify "Pole Pairs" instead of "Poles Number" in their datasheets. However, it is easy to convert "pole pairs" to "poles". Just multiply the number of pairs by 2 to get the number of poles. For example, if a motor has 8 "Pole Pairs", then it means that Poles Number is 8*2=16 poles. There is a simple procedure that experimentally determines the Poles Number. There is a video that explains how to commission an "Unknown Motor".	UINT16, 0x2000, 0x03, rw
4	Phase Resistance (Line-to-Line)	Ohm	The phase-to-phase electrical resistance parameter should be taken from the motor's datasheet, or it can be measured using the controller itself. Even if the parameter is provided in the motor's datasheet, it is still recommended to	FLOAT32, 0x2000, 0x04,

			measure it and re-confirm the value. Electrical resistance generally increases as the temperature of the motor increases. As such, do not be surprised if the measured value differs from a value taken from the datasheet. The resistance is automatically measured by the controller during auto-configuration procedure.	rw
5	Phase Inductance (Line-to-Line)	H	The phase-to-phase inductance parameter should be taken from the motor's datasheet, or it can be measured using the controller itself. Even if the parameter is provided in the motor's datasheet, it is still recommended to measure it and re-confirm the value for a given motor. The inductance is automatically measured by the controller during auto-configuration procedure.	FLOAT32, 0x2000, 0x05, rw
6	Back-Emf Constant (Ke)	V/ (rad/ s)	The Back-Emf Constant (Ke) parameter defines how much voltage a motor generates whenever an external force spins its rotor thus turning the motor into an electric generator. Although not immediately apparent from the definition, the Back-Emf Constant (Ke) also tells how much torque the motor produces given an electric current that flows through the motor. There is a well-known relationship between an electric current flowing through a motor and a torque produced by the motor. The Ke constant plays a prominent role in a formula that details the relationship. The Back-Emf Constant (Ke) should generally be MEASURED using the controller itself. It is usually not a good idea to take the value of Ke straight from a motor's datasheet as there are so many ways the value can be measured by the manufacturer, and there are so many units the value can be expressed in, that it is very easy to make a mistake when configuring the controller. It is better to allow the controller to measure the constant, than risking to misconfigure it. Many manufacturers of the motors provide alternative constants called Velocity Constant (Kv) or Torque Constant (Kt) instead of the Back-Emf Constant (Ke) that the controller expects. However, there is a simple way to convert one constant into another (Ke, Kv, Kt). The configuration software comes with a tool that performs the conversions. It is wise to first measure Ke using the controller, and then compare it to a Ke value taken from a datasheet or derived from either Kv or Kt constant, assuming that at least one of those constants is provided by the manufacturer. The controller expects that the Back-Emf Constant (Ke) is configured in units called "V (peak, line-to-neutral) per electrical rad/s". The controller measures Back-Emf Constant (Ke) in those units, so one do not have to worry about units conversion. If one still wants to venture into converting between various units	FLOAT32, 0x2000, 0x06, rw

			used to define the Back-Emf (Ke) constant, the software comes with a tool that helps with that. To summarize, as the Back-Emf Constant (Ke) is something that can be easily misunderstood or misconfigured, it is better to rely on the controller's own capabilities to properly measure and auto-configure the parameter. The Back-Emf Constant (Ke) is automatically measured by the controller during auto-configuration procedure.	
7	Payload: Viscous Damping Constant	Nm/ Hz	Viscous damping refers to forces of friction that are proportional to the motor's speed of rotation. An example of viscous damping would be water giving mechanical resistance to a pump's motor. Note that viscous damping refers to forces originated in both the motor and its payload, rather than just in the motor. This means that the parameter needs to be determined experimentally, rather than taken from the motor's datasheet. If Viscous Damping is not a concern, set the parameter's value to 0 or to a very small value, and just proceed with commissioning the motor. There is an experimental procedure that allows measuring the Viscous Damping Constant using the controller's built-in capabilities. The controller expects that the Viscous Damping Constant is provided in "Newton-Meters per Electrical Revolution per Second (Hz)" units. The Viscous Damping Constant is used by the controller in the following ways: • Feed-Forward Optimization: Viscous Damping Compensation for Electronic Speed Control and Servo Control. • As an input into a procedure that measures "Moment of Inertia of Rotor and Payload". A Feed-Forward Optimization called "Viscous Damping Compensation" improves the dynamics and efficiency of an electric drive in case the drive experiences much viscous damping. The optimization can be enabled once the Viscous Damping Constant has been measured using an experimental routine. By default, the optimization is turned off, and thus the constant is not used for anything other than as an input into a procedure that measures "Moment of Inertia of Rotor and Payload". To properly measure "Moment of Inertia of Rotor and Payload", another parameter in this section, using the controller's built-in capabilities, one first needs to properly measure "Viscous Damping Constant". Typically, one would initially leave the Viscous Damping parameter as 0 or a very small value, complete the first pass of the configuration, and later come back to the parameter to measure it, refine the configuration, and enable an	FLOAT32, 0x2000, 0x07, rw

			optimization feature that utilizes the parameter.	
8	Moment of Inertia of Rotor and Payload	kg*m2	The "Moment of Inertia of Rotor and Payload" parameter is used by the controller to determine parameters of control laws related to Electronic Speed Control (ESC) mode. Specifically, "ESC Kp" parameter of speed control loop is proportional to the moment of inertia. If the moment of inertia is not measured correctly, the electric drive might experience vibrations or noise whenever operated in a speed-controlled mode or in a servo mode. It is generally not required to be very precise in determining the moment of inertia. It is enough to be "about right" since the control laws have significant stability margins. The controller can measure the Moment of Inertia directly. Note that before invoking the procedure, one might have to measure "Viscous Damping Constant", another "Datasheet" section's parameter. The moment of inertia is automatically measured by the controller during an auto-configuration procedure. In many cases, it is sufficient to just launch the auto-configuration procedure that figures everything out. However, the procedure assumes that Viscous Damping is not present. If that turns out to be not the case, the auto-configuration procedure overestimates the moment of inertia. This overestimation might lead to vibrations or noise in the drive, whenever the drive is operated under Electronic Speed Control (ESC). In that case, please refer to a tutorial that explains how to measure both Moment of Inertia and Viscous Damping Constant together, and then rectify control laws. The Moment of Inertia is needed to estimate an "ESC Kp" parameter of a control law for Electronic Speed Control (ESC). An initial value for the Moment of Inertia can be taken from the motor's datasheet. A challenge is that the datasheet provides the moment of inertia of the motor's own rotor while what is needed is a combined moment of inertia of both the rotor and its payload. Thus it is recommended to use the controller to directly measure the combined moment of inertia instead of relying on the datasheet value. Typically, one would start commissioning a motor with a value taken from a datasheet, and would come back later to refine the moment of inertia by measuring it using the controller's built-in capabilities.	FLOAT32, 0x2000, 0x08, rw
9	Hall Sensors	0 or 1	• 0: Sensorless • 1: Sensored The parameter "Hall Sensors" tells the controller whether or not the motor has built-in Hall sensors. Many brushless motors come with Hall sensors. The sensors enable the controller to maintain a stable torque at low or zero speeds. NOTE: Both this parameter and a corresponding section "Peripheral: Hall Sensors" are automatically configured by the controller's auto-configuration procedure. Typically, one would not edit these manually.	BOOL, 0x2000, 0x09, rw

			An advantage of having Hall sensors is that the motor works in a much more stable way at low or zero speeds as compared to "sensorless" motors. With Hall sensors, the motor can produce torque at zero speed or whenever the motor is stalled. This is not the case with "sensorless" motors. If a motor does not have Hall sensors or an encoder (or the sensors are not yet wired to the controller), the controller operates the motor in a so-called "sensorless" mode. An issue with "sensorless" mode is that it only works whenever the motor reaches a certain speed. Getting a motor to that speed requires the controller to employ a special technique called "Kickstart" (configured and enabled separately). The "Kickstart" technique might cause ripples of torque at very low speeds, just like a gasoline engine. The ripples at low speeds are not a problem in a wide range of applications such as pumps, propellers, or electric scooters that do not need to operate at very low speeds. Other applications warrant a use of "sensored" motors or even better, motors with encoders. To summarize the point, "sensorless" motors are cheaper and easy to use, but might face challenges whenever operated at very low speeds. Motors with Hall sensors ("sensored") have better performance at low speeds, produce torque at zero speeds, but might cost a bit more, and require extra wiring as well as extra configuration on the controller side. Note that even if a brushless motor has built-in Hall sensors, it is absolutely legal to initially run the motor as a "sensorless" one. The Hall sensors can be wired to the controller later as a way to improve performance at low or zero speeds of an already operational drive. In fact, it is recommended to do it that way since it allows to gradually introduce complexity into the drive's wiring and configuration. The controller can also utilize an external encoder (a "Motor Encoder") instead of Hall sensors when operating a "sensorless" motor. Such an encoder is even better than Hall sensors, when it comes to performance at low or zero speeds. An added benefit is that the encoder can be used for Servo/Direct Drive functions. The issue is that an encoder is typically much more expensive than built-in Hall sensors, and might require complicated mechanical installation, unless the encoder is built into the motor itself. Set this parameter to "Sensorless" and proceed to configuring the "Motor Encoder" parameter if you are commissioning a drive with an encoder, but without Hall sensors. The controller's auto-configuration procedure is capable of determining if the motor has Hall sensors that are properly wired to the controller. If the sensors are connected to the controller, but the auto-configuration procedure does not identify the motor as a "Sensored" one, please check the cabling and connectors between the controller and the Hall sensors of the motor.	
10	Motor Encoder	-	• 0: No encoder • 1: Quadrature Encoder • 2: BISS-C/SSI Encoder	UINT16, 0x2000, 0x0A,

			• 3: SPI Encoder • 4: PWM Encoder The controller is designed to interface to up to two absolute encoders integrated into the same electric drive. For configuration purposes, the encoders are identified as a "Motor Encoder" and a "Servo Encoder". The encoders play different roles as explained below: • A "Motor Encoder" is an encoder rigidly connected to the motor's rotor or built into the motor itself. The "Motor Encoder" is used to sense the rotor's angular position in relation to the motor's stator. This measurements are used by the controller for commutation of the stator's windings to create torque and to measure speed. The place of mechanical installation must be chosen in such a way that no gearbox or a belt could introduce a backlash in the linkage between the rotor and the encoder. When rigidly attached to the rotor, such an encoder can be used by the controller to accurately measure the angular position of the rotor. The angular position of the rotor is what the controller needs to manage magnetic fields inside the motor in such a way that the motor generates torque. An added benefit is that such an encoder can be used for Servo/Direct Drive or Brake functions. The encoder could be of a SINGLE-TURN or a MULTI-TURN type. • A "Servo Encoder" on the other hand is a load-side encoder dedicated to performing servo control functions, due to control issues related to backlash of a gearbox/speed reducer. An encoder playing this role might be linked to the rotor via backlash-introducing mechanisms, such as a gearbox or a belt. The backlash implies that some other means (Hall sensors, a motor encoder, or sensorless control) is to be used by the controller to sense the angular position of the rotor. Thus the distinction between the roles of a "Servo Encoder" and a "Motor Encoder". Note that a "Servo Encoder" can be of a ROTARY SINGLE TURN, ROTARY MULTI-TURN or a LINEAR type. To summarize, a "Motor Encoder" is a functional substitute for Hall sensors or sensorless control, with further benefits related to that it can be used for Direct Drive Control, Servo Control, or Brake functions. Such an encoder is much better than Hall sensors when it comes to controlling a brushless motor at low or zero speeds, due to higher resolution of shaft position sensing, but is more expensive in terms of hardware. Typical Direct Drive motor features such an encoder employed for both spinning the motor and for servo positioning, thus killing two birds with one stone. Note that if an electric drive is equipped with Motor Encoder, it does not make sense to use Hall Sensors since the encoder typically has a much better resolution than Hall Sensors. Besides a zero backlash requirement, the motor encoder's interface to the controller must be of a low latency. This is especially true for encoders with	rw

			digital interfaces such as SPI or SSI. High latency or high backlash of an encoder makes it difficult for the controller to use the encoder for the purpose of spinning a brushless motor. In such a case, the controller shall be configured to use Hall sensors (if present) or sensorless control instead. Note that some types of encoders share pins and thus cannot be used together at the same time. After enabling an encoder here, proceed to configuring an associated encoder peripheral in a corresponding section. It usually makes sense to first run a brushless motor in a sensorless mode, and only later connect an encoder to the controller. This approach helps gradually introduce complexity into the controller's configuration. Note that even if a motor is equipped with an encoder, it is possible to run it as a plain sensorless brushless motor. CAUTION: If you enable a "Motor Encoder" here, make sure the configuration parameter "encoder bias vs. electrical position" of the corresponding encoder peripheral is properly set. The parameter defines an angular offset of the rotor vs. motor encoder readings. The parameter is important for proper positioning of magnetic field inside the motor. Failing to do so may cause the motor to unexpectedly accelerate out of control once given a motion command.	
11	Servo Encoder	-	• 0: No encoder • 1: Quadrature Encoder • 2: BISS-C/SSI Encoder • 3: SPI Encoder • 4: PWM Encoder A "Servo Encoder" is a load-side encoder dedicated to servo control function. This encoder is allowed to be linked to the motor via backlash-introducing mechanisms such as a gearbox or a belt. The implies that some other means (Hall sensors, a "Motor Encoder", or Sensorless Control) shall be used by the controller to sense the angular position of the rotor. Note that a "Servo Encoder" can be of a ROTARY SINGLE-TURN, ROTARY MULTI-TURN or a LINEAR type. Note that some types of encoders share pins and thus cannot be used together at the same time. After enabling an encoder here, proceed to configuring an associated encoder peripheral in a corresponding section. It usually makes sense to first run a brushless motor in a sensorless mode, and only later connect an encoder to the controller. This approach helps gradually introduce complexity into the controller's configuration. Note that even if a motor is equipped with an encoder, it is possible to run it as a plain sensorless brushless motor.	UINT16, 0x2000, 0x0B, rw

			If "Feature: Use Servo Encoder for Speed Measurements" is enabled, then the controller uses the "Servo Encoder" to also measure electrical speed instead of relying on Hall Sensors or Sensorless Control for this purpose. In most cases, it is beneficial to use Servo Encoder instead of Hall Sensors for speed measurements, since Hall Sensors are not suitable for measuring very low speeds. However, very high gearbox reduction ratios could dilute the advantages of Servo Encoder when it comes to precision of measuring rotor's speed as compared to measurements using Hall Sensors. Thus, this feature can be turned on or off via the configuration parameter.	
12	Gearbox Reduction Ratio	-	The "Gearbox Reduction Ratio" parameter needs to be set whenever a servo actuator is equipped with a gearbox, a belt, or any other type of a speed reducer. Otherwise set this parameter to 1.0. For ROTARY servos, set the parameter to match a reduction ratio of the servo mechanism's gearbox or belt. For example, if the reduction ratio is 100:1, then set this parameter to 100. If there is a multi-stage speed reducing mechanism (such as a belt driving a harmonic gear set), make sure a combined reduction ratio is used. For LINEAR servo mechanisms, determine how many rotor revolutions it takes the motor to move the linear encoder from its initial position (such as a leftmost one) to the furthest end position (such as a rightmost one). It might turn out to be a fractional number. Use this number as the reduction ratio here. If an electric drive does not have a "Servo Encoder", a gearbox or a belt, just set this parameter to 1.0.	FLOAT32, 0x2000, 0x0C, rw

Configuration - Control Laws

The "Control Laws" section defines settings for various control laws implemented in the firmware of the controller. The "Control Laws" section defines how the drive responds to commands given by a parent control system. Note that the parameters within this section are automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller. The parameters are automatically derived from parameters configured in the "Datasheet" section. In most cases, one should not enter the "Control Laws" parameters manually, but instead use one of the provided automated tools to generate the an initial set of parameters. After an initial set of control laws has been generated by a tool, minor adjustments can be made manually for tuning purposes.

#	Parameter	Units	Description	CANopen
1	Speed Filter: T	sec	The "Speed Filter: T" parameter, a time constant, plays a role in determining what perturbations in speed readings the electric drive needs to respond to. Note that the speed perturbation might be caused by sudden changes in payload characteristics ("bumps on the road"), or may turn out to be a noise in speed measurements due discrete nature of an encoder or an ADC. The speed filter smooths out speed readings, thus hiding some of the noise coming to the control laws. The goal is to choose a value for the "Speed	FLOAT32, 0x2002, 0x02, rw

			Filter: T" parameter in such a way that the noise is removed, while relevant information about changes in speed readings caused by payload forces, is allowed to pass through the filter. Shorter "Speed Filter: T" time generally increases overall dynamics of the electric drive. However, the effect is indirect, but due to the fact that many other parameters in "Control Laws" section are dependent on chosen value of "Speed Filter: T". Always use "Spreadsheet" tool to modify the parameter, since the tool keeps all the parameters in "Control Laws" section in sync. The "Spreadsheet" tool and an auto-configuration routine of the controller analytically compute a conservative value for this parameter by analyzing the information provided in the "Datasheet" section. Nevertheless, a manual adjustment might be required for performance optimization. Tuning intuition: • Higher-resolution encoders warrant for lower "Speed Filter: T" times. The higher the resolution of an encoder, the more counts per second the encoder feeds into the controller's speed-computing algorithm, the less ripple of speed estimates the encoder causes, the less filtering is required, the shorter "Speed Filter: T" time can be set. Remember to always use "Spreadsheet" tool to modify the parameter, since many other parameters in "Control Laws" section depend on chosen "Speed Filter: T" and need to be kept in sync. • DECREASING the time constant TOO MUCH causes the controller to filter LESS noise in speed readings. This often manifests itself as an audible "white" noise coming from the drive, as the drive starts overreacting to unfiltered noise in speed readings. • INCREASING the time constant TOO MUCH causes the drive to filter out not just the noise, but also useful changes in speed readings caused by external payload forces. This causes the drive to experience difficulties in maintaining a commanded speed under an influence of external forces. This might manifest itself in speed oscillations or in longer times that take the drive to arrive to a commanded speed. NOTE: An initial conservative value for this parameter is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller. The initial value may turn out to be just right, or may require manual tuning during a drive commissioning process.	
2	Field Oriented Control: FOC Kp	V/A	The term "Field Oriented Control (FOC)" refers to an efficient method of controlling brushless motors. Search the Internet for background information on the method. The controller uses the FOC method, when getting brushless motors to produce a commanded torque. The FOC Kp parameter is a proportional gain of two similar PI controllers that manage Iq and Id electrical currents within a brushless motor under	FLOAT32, 0x2002, 0x12, rw

			Field-Oriented Control (FOC). The higher the gain, the stronger the controller responds to perturbations in the electrical currents by managing Ud and Uq output voltages. NOTE: The value for this parameter is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller. Typically, it is not required to manually change or set this parameter.	
3	Field Oriented Control: FOC T	sec	The FOC T parameter is an integral time constant of two similar PI controllers that manage Iq and Id electrical currents within a brushless motor under Field-Oriented Control (FOC). The higher the time constant, the slower the PI controllers responds to perturbations in the electrical currents. NOTE: The value for this parameter is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller. Typically, it is not required to manually change or set this parameter.	FLOAT32, 0x2002, 0x13, rw
4	Electronic Speed Control: ESC Kp	A/Hz	Electronic Speed Control (ESC) is a function of the controller that maintains a constant speed of the motor by automatically increasing or decreasing torque in response to changes in speed. Note that the speed readings are filtered by Speed Filter prior to being processed by Electronic Speed Control (ESC) function. The ESC Kp parameter is a proportional gain of a PI controller that commands torque to maintain a constant speed. The parameter defines how much electrical current needs to be driven through the motor to adjust torque in order to compensate for a change in speed. The higher a combined "Moment of Inertia of Rotor and Payload" is, the higher ESC Kp parameter should be. Tools: • The "Spreadsheet" tool computes a correct value for this parameter, whenever given a correct value of "Moment of Inertia of Rotor and Payload". • The controller automatically measures the "Moment of Inertia" during an auto-configuration procedure, and uses the measurement to compute a correct value for ESC Kp parameter. Note that measuring the moment of inertia requires that "Viscous Damping Constant" is measured first. Since ESC Kp parameter depends on a combined "Moment of Inertia of Rotor and Payload", it is important that the moment is properly measured, estimated, or even guessed. Note that direct measurements of the moment of inertia by the controller can be quite imprecise, especially when a complex	FLOAT32, 0x2002, 0x22, rw

			payload is connected to the electrical drive. As errors in the estimates directly translate into errors in ESC Kp parameter, some manual adjustments to the ESC Kp parameter might be required. Tuning intuition: • The higher a combined "Moment of Inertia of Rotor and Payload" is, the higher ESC Kp parameter should be. ESC Kp is proportional to the combined moment of inertia. • INCREASING the parameter too much might causes the motor to produce an audible "white" noise when running under Electronic Speed Control (ESC). • DECREASING the parameter too much causes the drive to arrive to a commanded speed too slowly, or experience difficulties in maintaining a constant speed. If manual adjustments are to be performed, it is recommended to use the "Spreadsheet" tool for this purpose. Note that when using the tool, it is better to manually adjust the "Moment of Inertia" and then re-compute the ESC Kp parameter, rather than adjusting the ESC Kp parameter itself. Note that the speed here is defined in electrical revolutions per second (Hz). To convert Hz to motor shaft's revolutions per second, just divide it by the number of pole pairs. For example, assuming the speed is 20 Hz (electrical), and Poles Number is 8, then the corresponding speed in motor shaft's revolutions per second is 20 / (8/2)= 5.0 Hz (revolutions per second), which is 5 * 60 = 300 RPM.	
5	Electronic Speed Control: ESC T	sec	The "ESC T" parameter is an integral time constant of a PI controller that commands torque to maintain a constant speed. The higher the time constant, the slower the ESC controller reacts to small perturbations in speed readings. This parameter should generally be computed using the "Spreadsheet" software tool. The controller itself can also determine an appropriate value for this parameter during an auto-configuration procedure. Typically, it is not required to manually change or set this parameter.	FLOAT32, 0x2002, 0x23, rw
6	Servo: Kp	Hz/ rad	The "Servo: Kp" parameter defines a proportional relationship between a speed of servo motion and a distance to a target position the servo is commanded to move to. The closer the servo approaches the target position, the slower it moves. This proportional relationship is governed by the "Servo: Kp" parameter. The speed becomes 0.0 whenever the target position is reached, and the motor stops.	FLOAT32, 0x2002, 0x32, rw

			Example: lets assume that "Servo: Kp" is configured as 20 Hz/rad. If the current distance to the target is 2.0 rad, the servo starts traveling with a speed of 20 * 2.0 = 40 Hz. As the servo's shaft travels along, the distance to the target position eventually gets reduced to 0.5 rad, so the speed at that point drops to 20 * 0.5 = 10 Hz. Whenever the servo reaches its destination, the distance to the target becomes 0.0 rad, so the speed drops to 0.0 Hz, and the motor stops. The distance to the destination (measured in encoder counts), is first normalized by dividing it by the encoder's resolution and then by multiplying it by (2*PI) rad. This is done for both ROTARY and LINEAR encoders. For example, if a servo encoder has a resolution of 65536 counts, and the distance to the destination is 1000 counts, then the normalized distance would be 1000 / 65536 * (2*PI) = 0.095873799 rad. This normalized distance is multiplied by the pre-configured "Servo: Kp" factor to determine the speed with which the servo should travel at that point (as explained above). The speed is then clamped by comparing it to "Servo: Speed Limit", another configuration parameter. For example, if the "Servo: Speed Limit" is set to be 20Hz, then the speed with which the servo travels never gets higher than 20Hz. In fact, most of the time, the servo travels at this speed limit until the speed starts dropping upon approaching a target. All the speeds here are defined in electrical revolutions per second (Hz). To convert the electrical revolutions per second to servo shaft's revolutions per second, divide it by the number of pole pairs and then divide it by the gearbox's reduction ratio. For example, assuming the speed is 20 Hz (electrical), Poles Number is 8, and the gearbox reduction ratio is 100, then the corresponding speed in servo shaft's revolutions per second is 20 / (8/2) / 100 = 0.05 revolutions per second, which is 0.05 * 60 = 3.0 RPM.	
7	Servo: Speed Limit (Electrical Frequency)	Hz	The "Servo: Speed Limit" parameter sets a maximum speed with which the servo travels to its target position. The servo moves at this speed limit most of the time, only reducing the speed before arriving to its target destination. The period when servo is moving with this maximum speed is called a "constant speed" segment of the servo motion. The limit is used to clamp the speed computed using "Servo: Kp" parameter. Note that the speed is defined in electrical Hz (electrical revolutions per second). This parameter can be changed dynamically (e.g. via CANopen interface) before executing a particular coordinated move.	FLOAT32, 0x2002, 0x34, rw
8	Servo: Dead Zone Radius	counts	The "Servo: Dead Zone Radius" defines how close to a target position the servo needs to attempt to arrive. This parameter is sometimes useful for preventing oscillations around target positions in servo mode. Otherwise, keep this parameter as 0 (default).	UINT32, 0x2002, 0x36, rw

9	Micro-Speeding: Speed Limit (Electrical Frequency)	Hz	The "Micro-Speeding: Speed Limit" parameter defines a speed at which the controller switches between Field Oriented Control (FOC) and Micro-Speeding Control algorithms. If a commanded speed is less than the limit, the controller uses Micro-Speeding Control algorithm. Otherwise, it uses Field Oriented Control (FOC) method. "Micro-Speeding" refers to a proprietary brushless motor control technique devised for very low speed control. The reason this technique exists is that Field Oriented Control (FOC) algorithm does not work well at low speeds with Hall Sensors, and even more so with Sensorless Control. This is due to a range of issues related to inaccuracies of measuring speed and rotor's position at low speeds. Various control instabilities start manifesting themselves whenever at low speeds. On the other hand, Micro-Speeding Control works reliably at low speeds, but does not work well at high speeds. Thus the need to switch between the control methods that compliment each other. However, Micro-Speeding Conrtol is far less energy efficient than Field Oriented Control (FOC) in terms of heat generated by the motor. To enable or disable Micro-Speeding, use a parameter named "Feature: Micro-Speeding". NOTE: The value for this limit is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller.	FLOAT32, 0x2002, 0x72, rw
10	Kickstart: Speed Limit (Electrical Frequency)	Hz	"Kickstart" refers to a method of starting sensorless brushless motors. The term "sensorless" means that such a motor is not equipped with Hall Sensors or a Motor Encoder. The controller uses a clever mathematical method called "Sensorless Observer" to deduce information about the rotor's position by sensing Back-Emf voltages produced by the motor. The "Sensorless Observer" method is used whenever controlling a sensorless brushless motor, since no other source of information about the rotor's position is available. Note that the position of the rotor has to be known to the controller to properly position magnetic fields inside the motor using Field Oriented Control (FOC) method, so that the motor produces torque. However, a magic behind the "Sensorless Observer" method works only when the rotor is moving, since that's when the motor produces measurable Back-Emf signals. If the rotor is not moving or not moving fast enough, the controller does not know the position of the rotor, and thus cannot properly position the magnetic fields inside the motor. The issue is a kind of a "chicken or egg" problem. To move the rotor, the controller needs to know the position of the rotor, but to know the position of the rotor, the rotor needs to be already moving. This challenge only pertains to "sensorless" motors, and does not apply to motors equipped with Hall sensors or encoders that give the controller a usable reading of the rotor's position at any speed. That's	FLOAT32, 0x2002, 0x52, rw

			why the Hall sensors are used in the first place. The "Kickstart" method solves the "chicken or egg" problem by initiating a motion of the rotor in a way that does not require any knowledge of the rotor's position. Once the rotor starts moving under Kickstart Control, then the physics behind the "Sensorless Observer" kicks in, and the controller switches into the efficient Field Oriented Control (FOC) method to proceed with acceleration of the rotor. It typically takes a fraction of a second for the Kickstart Control to do its thing, and hand over the control to Field Oriented Control (FOC) function. A side effect is that the "Kickstart" procedure might create ripples of torque while accelerating the motor. As those start-up ripples are not a problem for many applications, the sensorless motors are popular due to simplicity and low cost, albeit at the expense of complexity of the controller. The parameter "Kickstart: Speed Limit" defines a speed which the Kickstart procedure accelerates the rotor to, before attempting to hand over control to Field Oriented Control (FOC). An implied assumption is that the magic behind the "Sensorless Observer" method starts producing reliable estimates of the rotor's position at that speed, so that it becomes possible for Field Oriented Control (FOC) to take over. NOTE: The value for this limit is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller.	
11	Kickstart: T	sec	The parameter "Kickstart: T" defines a timeframe within which the "Kickstart" procedure accelerates a motor towards a pre-configured "Kickstart: Speed Limit". If the kickstart procedure does not succeed within the given timeframe, the controller makes another attempt, and keeps making attempts until it successfully hands over control of the motor to Field Oriented Control (FOC). Tuning intuition: • INCREASE the "Kickstart: T" parameter if the kickstart procedure does not reliably start the sensorless motor each time. This might happen, for example, if a payload attached to the motor has a moment of inertia too large to be accelerated to "Kickstart: Speed Limit" within the specified time. • DECREASE the "Kickstart: T" parameter if a torque ripple caused by the "Kickstart" procedure becomes a problem in a given application. Shorter kickstart times makes the torque ripples shorter in time too. NOTE: The value for this parameter is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller.	FLOAT32, 0x2002, 0x53, rw
12	Sensorless Observer: EMF Zero Speed Voltage (high	V	The "Sensorless Observer" method works by measuring voltages generated by the motor. Whenever the rotor is moving, permanent magnets of the rotor happen to interact with coils of the stator in such a way that measurable voltages are generated on phase lines (just like in an electric generator). A	FLOAT32, 0x2002, 0x62, rw

	watermark)		physical phenomena behind this effect is called "Back-Emf". The phenomena is a foundation for the "sensorless" sensing of the rotor's position. The controller continuously measures the Back-Emf voltages using its electronic circuits, and applies some math to the measurements to derive a position of the rotor. The math is what is actually called the "Sensorless Observer" method. The method works well at medium and high speeds. However, at low speeds, the Back-Emf voltages turn out to be too low to be reliably separated from a noise present in the electronic circuits. To counter the noise issue, the "Sensorless Observer" discards voltage readings that are lower than a certain voltage threshold. Effectively, the Sensorless Observer is designed to wait until the Kickstart procedure manages to accelerate the rotor to a speed high enough for the voltages to be reliably separated from the noise. The parameter "EMF Zero Speed Voltage (high watermark)", measured in volts, specifies a threshold that the Back-Emf readings should reach in order for Sensorless Observer to start providing reliable estimates of the rotor position. The threshold should be aligned with "Kickstart: Speed Limit" parameter since at that speed limit the motor is supposed to generate Back-Emf voltages that are higher than the noise. NOTE: The value for this parameter is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller.	
13	Sensorless Observer: EMF Zero Speed Voltage (low watermark)	V	The parameter "EMF Zero Speed Voltage (low watermark)", measured in volts, specifies a threshold that Back-Emf readings should drop to in order for Sensorless Observer to declare that its rotor position estimates are no longer reliable. NOTE: The value for this parameter is automatically computed by the "Spreadsheet" tool or by an auto-configuration procedure of the controller.	FLOAT32, 0x2002, 0x63, rw

Configuration - Features

The "Features" section provides means to customize the motor control laws to meet requirements of a specific application.

#	Parameter	Units	Description	CANopen
1	Feature: Kickstart	0 or 1	"Kickstart" refers to a method of starting "sensorless" brushless motors, the ones that do not have Hall sensors or encoders. Since the "Kickstart" procedure might create ripples of torque at start-up, there is a way to disable the function altogether by toggling the "Feature: Kickstart" parameter. If the Kickstart function is disabled, but a "sensorless" motor is used, then an external application-specific method of starting up the motor needs to be employed (e.g. pushing an electric scooter to kickstart its motor).	BOOL, 0x2004, 0x02, rw

			The Kickstart function is generally not needed or used for motors with Hall sensors or encoders, so disabling it does not change anything in that case. An exception are drives equipped with absolute quadrature encoders. A challenge with absolute quadrature encoders is that they use an INDEX signal to identify a zero position each time the drive starts up. The controller uses the same Kickstart routine to initially rotate such a motor until its quadrature encoder stumbles upon an INDEX signal. The controller then switches to Field Oriented Control (FOC) since the quadrature encoder is then providing an absolute position of the rotor.	
2	Feature: Micro-Speeding	0 or 1	"Micro-Speeding" refers to a proprietary brushless motor control technique devised for very low speed control. The reason this technique exists is that Field Oriented Control (FOC) algorithm does not work well at low speeds with Hall Sensors, and even more so with Sensorless Control. This is due to a range of issues related to inaccuracies of sensing speed or rotor position at low speeds. Various control instabilities start manifesting themselves at low speeds. On the other hand, Micro-Speeding works reliably at low speeds. However, it may cause an excessive heating of the motor. Enabling the "Micro-Speeding" feature changes the way the controller manages motion at low speeds. The motion at low speeds becomes much more precise, but at the same time not as energy efficient. Note that more heat is generated by a motor under Micro-Speeding Control, than the same motor under Field Oriented Control (FOC). This is not a problem in many applications where energy efficiency can be traded for high precision and mechanical simplicity of Micro-Speeding control. If the feature is enabled, the controller begins to dynamically switch between Field Oriented Control (FOC) and Micro-Speeding when executing commands coming from a parent control system. The controller automatically determines which method of control is the most appropriate when executing a given command. This switching behavior is influenced by a "Micro-Speeding: Speed Limit" parameter, configured in the "Control Laws" section.	BOOL, 0x2004, 0x62, rw
3	Feature: D-Q Coupling Compensation (Feed-Forward)	0 or 1	"D-Q Coupling Compensation" is an advanced motor control technique that facilitates smooth transitions between modes of operation of an electrical drive. For example, if an electric drive is running under Electronic Speed Control (ESC), and then is given a command to switch to Electronics Torque Control (ETC), such a transition might cause a sudden change in electric currents flowing through the motor due to a difference in control laws. This change might cause a ripple in torque, mechanically stress the drive, or produce an audible jolt. The "D-Q Coupling Compensation" feature, when enabled, facilitates a smooth transition between various modes of operation. The reason the feature is not enabled by default is because the feature requires that the speed of the motor is correctly measured by the controller. As that	BOOL, 0x2004, 0x42, rw

			might not be the case during initial phases of controller configuration, the feature is disabled by default, so that it does not cause random oscillations or a noise early on. The feature needs to be enabled at later phases to improve performance of the electrical drive once an initial pass of configuring its controller has been completed, and the drive is already operational. Tuning intuition: • If a drive starts experiencing speed oscillations under Electronic Speed Control (ESC) whenever the feature is enabled, DECREASE "Speed Filter: T" parameter using the "Spreadsheet" tool. • If the drive starts producing excessive noise under Electronic Speed Control (ESC) whenever the feature is enabled, INCREASE "Speed Filter: T" parameter using the "Spreadsheet" tool.	
4	Feature: Viscous Damping Compensation (Feed-Forward)	0 or 1	The feature "Viscous Damping Compensation" improves dynamics and efficiency of an electric drive if the drive experiences much viscous damping in its payload such as the mechanical resistance of water to a pump's motor. Note that "Viscous Damping Constant" needs to be experimentally measured using the controller's means, and configured prior to enabling the feature.	BOOL, 0x2004, 0x52, rw
5	Feature: Field Weakening	0 or 1	"Field Weakening" is an advanced motor control technique that allows reaching speeds higher than a rated speed of a brushless motor. It is like shifting the drive to a higher gear, but electromagnetically. Note that the higher speeds are reached at the expense of energy efficiency of the electrical drive. In normal circumstances, a maximum speed of a permanent magnet synchronous motor (PMSM) is limited by the voltage of its power supply. The higher the voltage of power supply is, the higher maximum speed a brushless motor can reach. The permanent magnets of the rotor produce Back-Emf voltage in stator coils (just like an electric generator). As the speed grows, the generated Back-Emf voltage grows too. Whenever the Back-Emf voltage matches the voltage of power supply, the brushless motor reaches its maximum speed and cannot accelerate any further. It might be not immediately obvious why it is so, but just know that it is so due to some laws of physics: input voltage limits maximum reachable speed. The permanent magnets of the rotor come with a magnetic field attached to them, the one that interacts with coils and produces the Back-Emf voltage (as well as torque). What "Field Weakening" technique does is that it drives an additional electric current through the stator coils in such a way that it creates a magnetic field in the coils that cancels out some of the magnetic field attached to the permanent magnets of the rotor. In other words, the coils are used as electromagnets to cancel out a portion of the permanent magnets' field. The net effect is that the permanent magnets become "weaker". Weaker magnets generate weaker Back-Emf voltage in coils. This means that higher speeds can be reached before the "weaker" Back-Emf voltage matches the voltage of a	BOOL, 0x2004, 0x22, rw

			power supply and the motor stops accelerating. To summarize, by enabling the Field Weakening feature, the permanent magnets of the motor are instantly made "weaker" as some of their magnetic field is canceled out by an opposing magnetic field generated by the coils. This reduces the torque of the motor, but increases the maximum reachable speed. This transformation is made instantly by toggling this configuration parameter, just like shifting a gear in a car. A drawback of Field Weakening is its energy inefficiency due excessive heat generated by the coils. What happens is that the additional electric current driven through the coils to weaken the field of permanent magnets, is not used for producing useful torque, but instead is heating the motor. This wastes some of the energy on heating rather than torque. If higher speeds are needed for a particular application, it is might be a better design decision to either increase the voltage of power supply, or swap the motor for one with a lower Back-Emf (Ke) constant. However, using the "Field Weakening" technique is appropriate for many applications.	
6	Feature: Use Servo Encoder for Speed Measurements	0 or 1	Whenever this feature is enabled, the controller uses output of a Servo Encoder to measure speed. Otherwise, the controller uses Hall Sensors, a Motor Encoder or a sensorless technique to measure speed. This feature is especially useful with Hall Sensors that have poor accuracy of speed measurements at low speeds. Make sure that Servo Encoder, Poles Number (Rotor Poles) and Gearbox Reduction Ratio are properly configured before enabling this feature. In most cases, it is beneficial to use Servo Encoder instead of Hall Sensors for speed measurements, since Hall Sensors are not suitable for measuring very low speeds. However, very high gearbox reduction ratios could dilute the advantages of Servo Encoder when it comes to precision of measuring rotor's speed as compared to measurements using Hall Sensors. Thus, this feature can be turned on or off via the configuration parameter. If the following condition holds, then using Servo Encoder for speed measurement is advantageous as compared to using Hall Sensors for speed measurement: 6 * (Poles / 2) < [Servo Encoder's Counts per Revolution] / [Gearbox Reduction Ratio]	BOOL, 0x2004, 0x72, rw

Configuration - Brake

The "Brake" function uses a drive's own electric motor to prevent a motion of the drive's shaft under influence of external forces. The controller dynamically positions electromagnetic fields inside the motor in such a way that any significant motion of the shaft is countered by an electromagnetic force working in the opposite direction. This is like

applying a brake to the shaft, but without an actual physical braking device. If there is no external force, the "Brake" function does not trigger any countering electromagnetic forces, and thus does not draw energy from the power supply.

Note that the "Brake" function allows for an amount of "backlash" of the shaft. The backlash helps reduce consumption of energy. Consider using Servo Control function if the goal is to firmly hold the shaft at a given position. For the "Brake" function to work efficiently, the controller uses Hall sensors or a "Motor Encoder" to detect that the shaft is moving due to external forces, and to dynamically apply a countering electromagnetic force.

Note that if a motor does not have Hall sensors or a "Motor Encoder", then the controller defaults to using a statically positioned magnetic field when holding the shaft of the motor. The statically positioned magnetic field requires an electric current to be continuously driven through the coils of the motor regardless of the presence of any external forces. This electric current might cause excessive heating of the sensorless motor, and cause a continuous drain of energy from its power supply. In short, special care needs given to heat management when using the "Brake" function with sensorless motors.

#	Parameter	Units	Description	CANopen
1	Brake: Backlash Threshold	rad	The controller applies a countering force that is proportional to displacement of the shaft from a braking position. The "Brake: Backlash Threshold" parameter, expressed in electrical radians, specifies how far the shaft of the motor is allowed move under the influence of external forces before the controller applies a maximum countering electromagnetic force to bring the shaft back to its original braking position. Note that the backlash value is no less than (2*PI / 6) radians (electrical) for motors with Hall sensors as this is the finest angular resolution the sensors are capable of. Tuning intuition: • REDUCE this value to reduce backlash of the motor when on a "Brake". • INCREASE the parameter to reduce power consumption in the "Brake" mode.	FLOAT32, 0x2024, 0x03, rw
2	Brake: T rising	sec	The parameter "Brake: T rising" specifies how quickly the motor responds to sudden increases in external disturbing forces that move the shaft. Tuning intuition: • DECREASE the parameter to improve responsiveness of the "Brake" function to sudden jolts of external forces. • INCREASE the parameter to reduce power consumption in the "Brake" mode.	FLOAT32, 0x2024, 0x04, rw
3	Brake: T falling	sec	The parameter "Brake: T falling" defines how quickly the motor reduces a countering electromagnetic force once an external disturbing force disappears. Tuning intuition:	FLOAT32, 0x2024, 0x05, rw

			• DECREASE the parameter to reduce power consumption in the "Brake" mode. • INCREASE the parameter if shaft oscillations or an excessive noise are observed in the "Brake" mode.	
4	Feature: Brake on Idle	0 or 1	The parameter "Feature: Brake on Idle" instructs the controller to automatically apply the "Brake" function whenever the motor is in the "Idle" mode. This feature helps prevent the force of gravity from moving the joints of machines in an event of a sudden loss of connectivity with a parent control system.	BOOL, 0x2024, 0x10, rw

Configuration - Work Zone

The term "Work Zone" defines a multi-turn range which a servo actuator's output is expected to move within. This is applicable to ROTARY, LINEAR or MULTI-TURN ROTARY encoders.

The work zone is a "multi-turn" one when a ROTARY encoder is used. In other words, the work zone is not limited to just 360 degrees of the rotary encoder's resolution. Instead, it logically spans in both positive or negative directions as many encoder counts as needed. Both Servo Control and Direct Drive Control use the logical work zone's counts at their references instead of a physical servo encoder's readings. This makes it easier to develop "multi-turn" servo applications.

If "Work Zone Limits" are enabled, the controller makes an effort to prevent the servo's output from leaving the "Work Zone" even if an erroneous command is given by a parent control system. The "Work Zone Limits" are meant to define boundaries of safe operation of a servo mechanism, so that the mechanism does not hit itself or anything else. The "Work Zone Limits" is applicable to both ROTARY and LINEAR servo actuators.

#	Parameter	Units	Description	CANopen
1	Work Zone: zero offset	counts	The "Work Zone: zero offset" parameter defines a bias measured in servo encoder's counts, that is subtracted from a servo encoder's readings to determine a position within the work zone. This parameter is used to correct for an offset in the Servo Encoder's mechanical installation.	INT32, 0x2034, 0xB0, rw
2	Feature: Enforce Work Zone Limits	0 or 1	The feature forces the controller to stop the motion of a servo actuator whenever the servo is about to leave the boundaries of the work zone.	BOOL, 0x2034, 0xB1, rw
3	Work Zone: Limit in Negative Direction	counts	This parameter, measured in Servo Encoder's counts, defines a software-controlled limit of the work zone in NEGATIVE speed direction. Note that there are hardware limit switches that work in parallel with the software ones.	INT32, 0x2034, 0xB2, rw
4	Work Zone: Limit in Positive Direction	counts	This parameter, measured in Servo Encoder's counts, defines a software-controlled limit of the work zone in POSITIVE speed direction.	INT32, 0x2034, 0xB3, rw

			Note that there are hardware limit switches that work in parallel with the software ones.	
5	Feature: Customized Work Zone Dimension	0 or 1	This parameter forces the controller to use a user-defined "Work Zone: Dimension" instead of using Servo Encoder's "counts per revolution" parameter for this purpose. This might be useful in servo application development. For example, for a CNC machine, this value can be set to match a size of the CNC machine's actual work zone measured in Servo Encoder counts. • 0: Default • 1: Customized	BOOL, 0x2034, 0xC1, rw
6	Work Zone: Dimension	counts	This parameter specifies is a size of Work Zone defined in Servo Encoder counts. Usually, this size equals the value of "counts per revolution", a parameter of a Servo Encoder. However, it can be configured to be an arbitrary number to simplify application development. For example, for a CNC machine, this value can be set to match a size of the CNC machine's actual work zone measured in Servo Encoder counts.	UINT32, 0x2034, 0xC2, rw

Configuration - Fault Management

The controller automatically stops the electric drive whenever a hardware problem (a fault) is detected. After stopping the drive, the controller latches one or more "Fault Bits" flags in telemetry, and waits for a "Reset" command to come from an a parent control system. The controller keeps the motor powered off until the "Reset" command comes that resets the fault latches.

#	Parameter	Units	Description	CANopen
1	Hide Faults	bitmask	The bit mask hides selected faults thus preventing the faults bits from latching and stopping the electric drive whenever the faults occur. Use this feature with high care since ignored faults might cause troubles. Look for a description of parameter "Fault Bits" in the telemetry section to know meanings of each of the bits.	UINT16, 0x2044, 0x02, rw
2	Overcurrent Limit: Factor	-	The controller raises an Overcurrent Fault signal whenever electric current in any of the motor's phases exceeds a pre-configured "Maximum Continuous Current" by "Overcurrent Limit: Factor". This is a software-enforced limit. It complements hardware-enforced limits. For example, if the "Maximum Continuous Current" parameter is 7 A, and the "Overcurrent Limit: Factor" is 4.0, the controller raises an Overcurrent Fault signal in "Fault Bits" whenever electrical current in any of the phases reaches 7A*4=28A.	FLOAT32, 0x2044, 0x04, rw

Configuration - Peripheral: GPIO

#	Parameter	Units	Description	CANopen

1	Emergency Stop Switch	GPIO	This parameter specifies which GPIO pin is to be used as an Emergency Stop input. Refer to datasheet for a list of available input GPIO pins. If this parameter is set to 0, this means Emergency Stop function is disabled.	UINT16, 0x3020, 0x10, rw
2	Limit Switch: Negative Speed	GPIO	This parameter specifies which GPIO pin is connected to a limit switch acting in NEGATIVE speed direction. Refer to datasheet for a list of available input GPIO pins. If this parameter is set to 0, this means the function is disabled.	UINT16, 0x3020, 0x11, rw
3	Limit Switch: Positive Speed	GPIO	This parameter specifies which GPIO pin is connected to a limit switch acting in POSITIVE speed direction. Refer to datasheet for a list of available input GPIO pins. If this parameter is set to 0, this means the function is disabled.	UINT16, 0x3020, 0x12, rw
4	Switch Type	GPIO	• 0: Normally Open • 1: Normally Closed This setting applies to both Limit Switches as well as to the Emergency Stop Switch.	BOOL, 0x3020, 0x19, rw
5	Generic Output	GPIO	This parameter tells which GPIO pin is to be used to output command-controlled discrete or PWM signal. This signal is typically used to control a solenoid of a brake. Refer to datasheet for a list of available output GPIO pins. If this parameter is set to 0, this means the function is disabled.	UINT16, 0x3020, 0x20, rw
6	Generic Input	GPIO	This parameter tells which GPIO pin is to be used as a general-purpose input that can be read out via a telemetry interface. Refer to datasheet for a list of available input GPIO pins. If this parameter is set to 0, this means the function is disabled.	UINT16, 0x3020, 0x30, rw

Configuration - Peripheral: Hall Sensors

Many brushless motors come with built-in Hall sensors. The sensors help the controller improve the electric drive's efficiency at low or zero speeds. With Hall sensors, the motor produces a reliable torque at zero speed or whenever the motor is stalled.

Note that the controller automatically configures this section when an appropriate auto-configuration procedure is launched. Typically, one would not edit this section manually.

#	Parameter	Units	Description	CANopen

1	physical sensor for logical sensor 0	0/1/2	If phase "A" is positively energized, and phases "B" and "C" are negatively energized, this logical sensor reads as "1", while others read as "0".	UINT16, 0x3004, 0x06, rw
2	physical sensor for logical sensor 1	0/1/2	If phase "B" is positively energized, and phases "A" and "C" are negatively energized, this logical sensor reads as "1", while others read as "0".	UINT16, 0x3004, 0x07, rw
3	physical sensor for logical sensor 2	0/1/2	If phase "C" is positively energized, and phases "A" and "B" are negatively energized, this logical sensor reads as "1", while others read as "0".	UINT16, 0x3004, 0x08, rw
4	Hall signals inverted	0 or 1	This parameter instructs the controller to invert readings of all Hall sensors before mapping them to logical sensors.	BOOL, 0x3004, 0x05, rw

Configuration - Peripheral: Quadrature Encoder

This section needs to be configured only if either "Motor Encoder" or "Servo Encoder" parameter in the "Datasheet" section is set to "Quadrature Encoder". Otherwise, leave this section unchanged. The controller comes with dedicated hardware, a silicon peripheral, for interfacing quadrature encoders. The peripheral has peculiarities of configuration that are addressed in this section.

A challenge with absolute quadrature encoders is that they need to use an INDEX signal to search for a zero position each time the drive is powered up. The controller uses the "Kickstart" procedure to initially rotate such a motor until its quadrature encoder stumbles upon the INDEX signal. The controller then switches to Field Oriented Control (FOC) until it gets powered off again. The search is commenced upon receiving the first motion command from a parent control system. The direction of search is derived from the received command.

Note that hardware interface to quadrature encoders allows the controller to measure not just shaft position, but also speed. This feature can be turned on or off. There are two distinct methods of how the controller's hardware (silicon) computes the speed:

1. Method #1 "UNIT DISTANCE": The controller records how much time it takes the quadrature encoder's disk to travel a pre-configured UNIT DISTANCE measured in encoder counts (quadrature edges). By dividing the UNIT DISTANCE by the recorded time, the controller arrives to the first estimate of speed. The parameter UNIT DISTANCE is configured in this section. However, at higher speeds the recorded time becomes too short thus creating a quantization issue.
2. Method #2 "UNIT TIME": The controller records a distance (measured in encoder counts) that the encoder travels in a UNIT TIME period. By dividing the recorded distance by the UNIT TIME, the controller arrives to the second estimate of speed. The parameter UNIT TIME is configured in this section. However, at lower speeds the number of edges counted within the UNIT TIME could be too small thus creating a quantization issue.
3. Since Method #1 gives reliable estimates at lower speeds, while Method #2 gives reliable estimates at higher speeds, the controller chooses one estimate or the other by comparing the estimates to a SPEED SELECTION THRESHOLD configured in this section.

#	Parameter	Units	Description	CANopen
1	counts per revolution	counts	The parameter defines a maximum resolution of the quadrature encoder. The resolution is defined in quadrature edge counts per revolution. This parameter is to be taken from the encoder's datasheet.	UINT32, 0x3011, 0x02, rw
2	encoder bias vs. electrical position	counts	This parameter needs to be set only if the encoder is used for motor control (a "Motor Encoder"). Otherwise, keep this parameter as 0. This parameter specifies a zero offset of the encoder's mechanical installation vs. an electrical position defined by an order in which the motor's phase lines are connected to the controller. The bias can be experimentally determined as the following: • Positively energize phase "A". • Negatively energize both phases "B" and "C". • This can be done by issuing a "Direct Field Control: Electrical Position" command with command parameter 0 rad and a small voltage (e.g. 0.5V). • Let the motor's rotor settle at a position. • The encoder's reading at that position is the bias. The procedure needs to be performed after correcting for a possible inversion of a mechanical installation of the encoder (see "inverted installation" parameter in this section).	UINT32, 0x3011, 0x03, rw
3	inverted installation (swap A and B signals)	0 or 1	The direction of the motor's rotation should match the direction of the encoder's rotation. If that's not the case due to an inverted way the encoder is mechanically installed, this parameter helps correct the mismatch.	BOOL, 0x3011, 0x04, rw
4	polarity inversion	0 or 1	This parameter causes the electronic circuits of the controller to invert A, B, and I signals of the quadrature encoder before feeding the signals into the software for analysis.	BOOL, 0x3011, 0x11, rw
5	incremental encoder	0 or 1	• 0: an absolute encoder with an index signal ("I" or "Z" signal is present) • 1: an incremental encoder (no index signal)	BOOL, 0x3011, 0x16, rw
6	Feature: hardware-accelerated speed measurement	0 or 1	• 1: use hardware to measure speed via UNIT DISTANCE and UNIT TIME methods • 0: use firmware to measure speed by counting pulses Before enabling this feature, please configure hardware-related parameters UPPS, CCPS, UNIT TIME, speed selection threshold.	BOOL, 0x3011, 0x17, rw
7	UNIT DISTANCE:	0-15	This parameter is used in "UNIT DISTANCE" method of computing speed. The controller records how much time it takes the encoder to travel a pre-	UINT16, 0x3011,

	UPPS		configured UNIT DISTANCE measured in encoder counts (quadrature edges). By dividing the UNIT DISTANCE by the recorded time, the controller arrives to an estimate of speed. This method of computing speed gives reliable results at lower speeds. However, at higher speeds the recorded time becomes too short thus creating a quantization issue. The UNIT DISTANCE is configured as the following silicon-specific way: UNIT DISTANCE = 2^UPPS For example, if UPPS is 4, then UNIT DISTANCE is 2^4 = 16 (counts). Note: increasing UNIT DISTANCE too much introduces latency in speed measurement since the controller has to wait longer before it can compute speed.	0x13, rw
8	UNIT DISTANCE: Divider CCPS	0-7	This parameter is used in "UNIT DISTANCE" method of computing speed. When computing speed using the UNIT DISTANCE method, the controller has to precisely measure time as explained above. The way the controller's silicon peripheral measures the time is by counting ticks of the CPU clock using a 16bits counter. The ticks arrive at the frequency of CPU which is 90 MHz (double-check this for your controller). The frequency is then divided by a prescaler circuit configured here in the following silicon-specific way: PRESCALER = 2^CCPS For example, if CCPS is 7, then PRESCALER is 2^7 = 128. This translates to counter's frequency of 90 000 00 / 128 = 703125 Hz. The reason the pre-scaler is needed is because the 16bit counter in silicon that measures time might overflow if the CPU ticks are routed to it at the full CPU frequency. Note that if the encoder's speed is too low, it takes the encoder longer time to travel the UNIT DISTANCE, thus there is a risk that the counter might overflow within that longer time. Thus the need for this pre-scaler parameter.	UINT16, 0x3011, 0x14, rw
9	UNIT TIME	sec	This parameter is used in "UNIT TIME" method of computing speed. The controller records a distance (measured in encoder counts) that the encoder travels in a UNIT TIME period. By dividing the recorded distance by the UNIT TIME, the controller arrives to an estimate of speed. This method of computing speed gives reliable results at higher speeds. However, at lower speeds the number of edges counted within the UNIT TIME could be too small thus creating a quantization issue. ATTENTION: UNIT TIME must be equal or less than 1/Max_speed, where Max_speed is the maximum expected speed to be measured by the encoder, expressed in revolutions per second.	FLOAT32, 0x3011, 0x12, rw

10	speed selection threshold	RPS	The threshold tells when to dynamically switch from UNIT DISTANCE ("lower speeds") to UNIT TIME ("higher speeds") method. • If the speed is lower than the threshold, then the controller uses the "UNIT DISTANCE" method. • If the speed is higher than the threshold, then the controller uses the "UNIT TIME" method.	FLOAT32, 0x3011, 0x15, rw

Configuration - Peripheral: SSI/BISS-C Encoder

This section needs to be configured only if either "Motor Encoder" or "Servo Encoder" parameter in the "Datasheet" section is set to "SSI/BISS-C Encoder". Otherwise leave this section unchanged.

The controller reads out data from an SSI/BISS-C encoder by sending a train of pulses via CLOCK line. The encoder sends back a single bit of data to the controller via DATA line each time it receives a pulse from the controller. By sending the train of pulses, the controller reads out all the data bits (a packet) from the encoder.

If an encoder puts a CRC field into the packet, then the controller can be configured to use a CRC verification function to detect and discard corrupted packets. If the CRC verification fails, the controller discards the packet as a corrupted one, but DOES NOT raise a "Fault Bits" flag. The controller supports a limited number of CRC formulae. If an encoder implements a CRC formula that is not supported by the controller, then the CRC verification feature needs to be turned off.

If the encoder sends an ERROR bit in a data packet, the controller stops the motor and latches a corresponding "Fault Bits" flag. The motor remains powered off until the controller receives a "Reset" command from a parent control system. A WARN bit can also be extracted from the packet for telemetry purposes. However, the WARN bit is not used by the controller itself, and does not trigger any fault-handling logic. The WARN bit is only used for telemetry purposes.

#	Parameter	Units	Description	CANopen
1	counts per revolution	counts	The parameter defines a maximum resolution of the encoder. This parameter is to be taken from the encoder's datasheet.	UINT32, 0x3013, 0x02, rw
2	encoder bias vs. electrical position	counts	This parameter needs to be set only if the encoder is used for motor control (a "Motor Encoder"). Otherwise keep this parameter as 0. This parameter specifies a zero offset of the encoder's mechanical installation vs. an electrical position defined by an order in which the motor's phase lines are connected to the controller. The bias can be experimentally determined as the following: • Positively energize phase "A". • Negatively energize both phases "B" and "C". • Let the motor's rotor settle at a position. • This can be done by issuing a "Direct Field Control: Electrical	UINT32, 0x3013, 0x03, rw

			Position" command with command parameter 0 rad and a small voltage (e.g. 0.5V). • The encoder's reading at that position is the bias. The procedure needs to be performed after correcting for a possible inversion of the mechanical installation of the encoder (see "inverted installation" parameter in this section).	
3	inverted installation	0 or 1	The direction of the motor's rotation should match the direction of the encoder's rotation. If that's not the case due to a way the encoder is mechanically installed, this parameter helps correct the mismatch.	BOOL, 0x3013, 0x04, rw
4	request frequency: divider	-	The parameter defines how often the controller reads out data from the encoder. Specifically, this parameters specifies how often pulse trains are sent by the controller to the encoder via CLOCK line. Note that the encoder sends a single bit of data back to the controller via DATA line each time it receives a pulse from the controller. By sending a train of pulses, the controller reads out all the data bits (a packet) from the encoder. The parameter specifies a divider for the controller's sampling frequency. For example: The controller has a sampling frequency of 15 kHz or 15 000 samples per second (check this for your controller in "Device Information" telemetry section). If the divider is specified as 4, then the request frequency is 15 000 / 4 = 3750 Hz = 3.75kHz. This means that the controller reads out the data from the encoder 3750 times a second. Note that the request frequency should be aligned with a maximum request frequency specified in the encoder's datasheet.	UINT16, 0x3013, 0x10, rw
5	clock frequency: divider	-	This parameter characterizes pulses within a train of pulses that are sent by the controller to the encoder via CLOCK line to read out a data packet. The pulses are generated by a silicon peripheral that has peculiarities of configuration as explained below. The parameter specifies a divider for CPU frequency of the controller. The formula for the pulse's frequency is the following: clock frequency = [Half of CPU frequency] / (divider + 1) Example: If the controller's CPU frequency is 90 MHz, and "clock frequency: divider" is configured as 89, then this results in the following clock frequency: 90 MHz / 2 / (89 + 1) = 45 Mhz / 90 = 500kHz.	UINT16, 0x3013, 0x11, rw

			Intuition for selecting the clock frequency: • The clock frequency should not be higher than a maximum clock frequency defined in the encoder's datasheet. • On the other hand, the clock frequency should be high enough, so that the entire pulse train fits in a time window between subsequent data reads. Note that the frequency of data reads is defined by "request frequency: divider" parameter in this section. • Note that the encoder may require a timeout period at the end of each pulse train. This period shall also fit in the time window between subsequent data reads in addition to the pulse train itself. • The higher the frequency, the better (lower latency).	
6	clock polarity	0 or 1	The parameter tells the controller to electrically invert output signals on the CLOCK line. The parameter is rarely changed. Leave the default setting unless an application-specific need arises.	UINT16, 0x3013, 0x12, rw
7	clock phase	0 or 1	The parameter tells the controller to delay the moment when the DATA line is sampled vs. output pulse on the CLOCK line. The parameter is rarely changed. Leave the default setting unless an application-specific need arises.	UINT16, 0x3013, 0x13, rw
8	total number of bits in packet	-	The parameter specifies the number of pulses the controller clocks out via the CLOCK line each time the controller reads the a data packet from the encoder. This parameter should be taken from the encoder's datasheet. The number of pulses matches the number of bits read out from the encoder. Note that there are 2 "empty" bits at the beginning of every SSI packet followed by an encoder-specific number of ACK bits as well as Start and CDS bits. Those empty bits, the ACK bits as well as the Start and CDS bits should be included when counting the total number of pulses to be clocked out via the CLOCK line. Furthermore, due to peculiarities of a silicon peripheral, the number of clocked out pulses is rounded up to the nearest 16. For example, if this parameter is set as 25 bits (16+9), the actual number of clocked out pulses is going to be 32 (16+16).	UINT16, 0x3013, 0x14, rw
9	POSITION field (count): start bit	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x20, rw
10	POSITION field (count): length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x21, rw
11	POSITION field (count): bit	0 or 1	This parameter defines a format of the data packet. The parameter needs to	BOOL, 0x3013,

	inversion		be taken from the encoder's datasheet.	0x22, rw
12	MULTI-TURN field (count): enable	0 or 1	This parameter enables or disables parsing of multi-turn counter. Enable this feature if the encoder is a multi-turn one. Set this to 0 if the encoder is not a multi-turn one.	BOOL, 0x3013, 0x30, rw
13	MULTI-TURN field (count): start bit	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x31, rw
14	MULTI-TURN field (count): length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x32, rw
15	MULTI-TURN field (count): bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3013, 0x33, rw
16	MULTI-TURN Count Limit (count)	turns	The parameter defines a maximum number of turns supported by a multi-turn encoder. This parameter is to be taken from the encoder's datasheet. Set this to 0 if an encoder does not support multi-turn functionality.	UINT32, 0x3013, 0x34, rw
17	ERROR bit: enable	0 or 1	This parameter specifies if the controller should analyze and react to an ERROR bit in received packets. If the parameter is enabled, the controller stops the motor and raises a "Fault Bits" flag upon receiving an ERROR bit. The motor remains powered off until the controller receives a "Reset" command from a parent control system.	BOOL, 0x3013, 0x2A, rw
18	ERROR bit: bit position	-	This parameter defines a format of the data packet.	UINT16, 0x3013, 0x2B, rw
19	ERROR bit: bit inversion	0 or 1	This parameter defines a format of the data packet.	BOOL, 0x3013, 0x2C, rw
20	WARN bit: enable	0 or 1	The WARN bit can be extracted from the data packet. However, it is not used for anything other than telemetry.	BOOL, 0x3013, 0x2D, rw
21	WARN bit: bit position	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x2E, rw

22	WARN bit: bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3013, 0x2F, rw
23	CRC field: enable	0 or 1	This parameter enables or disables CRC verification for received packets. Note that the controller supports a limited number of CRC formulae. If an encoder implements a CRC formula that is not supported by the controller, then the CRC verification function has to be turned off. Note that if CRC verification fails, the controller discards the packet as a corrupted one, but DOES NOT raise a "Fault Bits" flag.	BOOL, 0x3013, 0x23, rw
24	CRC field: start bit	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x24, rw
25	CRC field: length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x25, rw
26	CRC field: bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3013, 0x26, rw
27	CRC input: start bit	-	The CRC is computed over a particular portion of the packet as specified in the encoder's datasheet. For the purpose of this configuration procedure, the portion is called "CRC input". The CRC input may span multiple data fields across the packet. This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x27, rw
28	CRC input: length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3013, 0x28, rw
29	CRC input: bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3013, 0x29, rw

Configuration - Peripheral: SPI Encoder

This section needs to be configured only if either "Motor Encoder" or "Servo Encoder" parameter in the "Datasheet" section is set to "SPI Encoder". Otherwise leave this section unchanged.

The controller reads out data from the encoder by sending a train of pulses via SCK line. This line is sometimes called SCLK or CLOCK. The encoder sends a single bit of data back to the controller via MISO line each time it receives a

pulse from the controller. By sending a train of pulses, the controller reads out all the data bits (a packet) from the encoder.

If an encoder puts a CRC field into the packet, then the controller uses a CRC verification function to detect and discard corrupted packets. If the CRC verification fails, the controller discards the packet as a corrupted one, but DOES NOT raise a "Fault Bits" flag. The controller supports a limited number of CRC formulae. If an encoder implements a CRC formula that is not supported by the controller, then CRC verification feature needs to be turned off.

If the encoder sends an ERROR bit in a data packet, the controller stops the motor and raises a corresponding "Fault Bits" flag. The motor remains powered off until the controller receives a "Reset" command from a parent control system. A WARN bit can also be extracted from the packet for telemetry purposes, but the bit is not used by the controller, and does not trigger any fault-handling logic. The WARN bit is only used for telemetry purposes.

#	Parameter	Units	Description	CANopen
1	counts per revolution	counts	The parameter defines a maximum resolution of the encoder. This parameter is to be taken from the encoder's datasheet.	UINT32, 0x3014, 0x02, rw
2	encoder bias vs. electrical position	counts	This parameter needs to be set only if the encoder is used for motor control (a "Motor Encoder"). Otherwise, keep this parameter as 0. This parameter specifies a zero offset of the encoder's mechanical installation vs. an electrical position defined by an order in which the motor's phase lines are connected to the controller. The bias can be experimentally determined as the following: • Positively energize phase "A". • Negatively energize both phases "B" and "C". • This can be done by issuing a "Direct Field Control: Electrical Position" command with command parameter 0 rad and a small voltage (e.g. 0.5V). • Let the motor's rotor settle at a position. • The encoder's reading at that position is the bias. The procedure needs to be performed after correcting for a possible inversion of the mechanical installation of the encoder (see "inverted installation" parameter in this section).	UINT32, 0x3014, 0x03, rw
3	inverted installation	0 or 1	The direction of the motor's rotation should match the direction of the encoder's rotation. If that's not the case due to a way the encoder is mechanically installed, this parameter helps correct the mismatch.	BOOL, 0x3014, 0x04, rw
4	request frequency: divider	-	The parameter defines how often the controller reads out data from the encoder. Specifically, this parameters specifies how often pulse trains are sent by the controller to the encoder via SCK line. This line is sometimes called SCLK or CLOCK. Note that the encoder sends a single bit of data	UINT16, 0x3014, 0x10, rw

			back to the controller via MISO line each time it receives a pulse from the controller. By sending a train of pulses, the controller reads out all the data bits (a packet) from the encoder. The parameter specifies a divider for the controller's sampling frequency. For example: The controller has a sampling frequency of 15 kHz or 15 000 samples per second (check this for your controller in "Device Information" telemetry section). If the divider is specified as 4, then the request frequency is 15 000 / 4 = 3750 Hz = 3.75kHz. This means that the controller reads out the data from the encoder 3750 times a second. Note that the request frequency should be aligned with a maximum request frequency specified in the encoder's datasheet.	
5	clock frequency: divider	-	This parameter characterizes pulses within a train of pulses that are sent by the controller to the encoder via SCK line to read out a data packet. This line is sometimes called SCLK or CLOCK. The pulses are generated by a silicon peripheral that has peculiarities of configuration as explained below. The parameter specifies a divider for CPU frequency of the controller. The formula for the pulse's frequency is the following: clock frequency = [Half of CPU frequency] / (divider + 1) Example: If the controller's CPU frequency is 90 MHz, and "clock frequency: divider" is configured as 89, then this results in the following clock frequency: 90 MHz / 2 / (89 + 1) = 45 Mhz / 90 = 500kHz. Intuition for selecting the clock frequency: • The clock frequency should not be higher than a maximum clock frequency defined in the encoder's datasheet. • On the other hand, the clock frequency should be high enough, so that the entire pulse train fits in a time window between subsequent data reads. Note that the frequency of data reads is defined by "request frequency: divider" parameter. • The higher the frequency, the better (lower latency).	UINT16, 0x3014, 0x11, rw
6	clock polarity	0 or 1	The parameter tells the controller to electrically invert output signals on the SCK line. This line is sometimes called SCLK or CLOCK. Leave the default setting unless an application-specific need arises.	UINT16, 0x3014, 0x12, rw

7	clock phase	0 or 1	The parameter tells the controller to delay the moment when the MISO line is sampled vs. output pulse on the SCK line. Leave the default setting unless an application-specific need arises.	UINT16, 0x3014, 0x13, rw
8	total number of bits in packet	-	The parameter specifies the number of pulses the controller clocks out via the SCK line each time the controller reads a data packet from the encoder. This line is sometimes called SCLK or CLOCK. The number of pulses matches the number of bits read out from the encoder. This parameter should be taken from the encoder's datasheet. Furthermore, due to peculiarities of a silicon peripheral, the number of clocked out pulses is rounded up to the nearest 16. For example, if this parameter is set as 25 bits (16+9), the actual number of clocked out pulses is going to be 32 (16+16).	UINT16, 0x3014, 0x14, rw
9	POSITION field (count): start bit	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x20, rw
10	POSITION field (count): length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x21, rw
11	POSITION field (count): bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3014, 0x22, rw
12	MULTI-TURN field (count): enable	0 or 1	This parameter enables or disables parsing of multi-turn counter. Enable this feature if the encoder is a multi-turn one. Set this to 0 if the encoder is not a multi-turn one.	BOOL, 0x3014, 0x30, rw
13	MULTI-TURN field (count): start bit	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x31, rw
14	MULTI-TURN field (count): length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x32, rw
15	MULTI-TURN field (count): bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3014, 0x33, rw
16	MULTI-TURN	turns	The parameter defines a maximum number of turns supported by a multi-	UINT32,

	Count Limit (count)		turn encoder. This parameter is to be taken from the encoder's datasheet. Set this to 0 if an encoder does not support multi-turn functionality.	0x3014, 0x34, rw
17	ERROR bit: enable	0 or 1	This parameter specifies if the controller should analyze and react to an ERROR bit in received packets. If the parameter is enabled, the controller stops the motor and raises a "Fault Bits" flag upon receiving an ERROR bit. The motor remains powered off until the controller receives a "Reset" command from a parent control system.	BOOL, 0x3014, 0x2A, rw
18	ERROR bit: bit position	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x2B, rw
19	ERROR bit: bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3014, 0x2C, rw
20	WARN bit: enable	0 or 1	The WARN bit can be extracted from the data packet. However, it is not used for anything other than telemetry.	BOOL, 0x3014, 0x2D, rw
21	WARN bit: bit position	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x2E, rw
22	WARN bit: bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3014, 0x2F, rw
23	CRC field: enable	0 or 1	This parameter enables or disables CRC verification for received packets. Note that the controller supports a limited number of CRC formulae. If an encoder implements a CRC formula that is not supported by the controller, then the CRC verification function has to be turned off. Note that if CRC verification fails, the controller discards the packet as a corrupted one, but DOES NOT raise a "Fault Bits" flag.	BOOL, 0x3014, 0x23, rw
24	CRC field: start bit	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x24, rw
25	CRC field: length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x25, rw
26	CRC field: bit	0 or 1	This parameter defines a format of the data packet. The parameter needs to	BOOL,

	inversion		be taken from the encoder's datasheet.	0x3014, 0x26, rw
27	CRC input: start bit	-	The CRC is computed over a particular portion of the packet as specified in the encoder's datasheet. For the purpose of this configuration procedure, the portion is called "CRC input". The CRC input may span multiple data fields across the packet. This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x27, rw
28	CRC input: length	-	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	UINT16, 0x3014, 0x28, rw
29	CRC input: bit inversion	0 or 1	This parameter defines a format of the data packet. The parameter needs to be taken from the encoder's datasheet.	BOOL, 0x3014, 0x29, rw

Configuration - Peripheral: PWM Encoder

This section needs to be configured only if either "Motor Encoder" or "Servo Encoder" parameter in the "Datasheet" section is set to "PWM Encoder". Otherwise leave this section unchanged.

Absolute encoders with PWM output deliver position information to the controller by increasing or decreasing duty cycle of continuously sent PWM pulses. The duty cycle changes in a linear proportion to the measured position. The pulses are sent at a constant frequency, but the pulses' duty cycle changes along with absolute position measured by the encoder. The controller on its end measures the duty cycle of received pulses and knowing the resolution of the encoder, uses a mathematical proportion to extract the absolute position readings.

#	Parameter	Units	Description	CANopen
1	counts per revolution	counts	The parameter defines a maximum resolution of the encoder. This parameter is to be taken from the encoder's datasheet.	UINT32, 0x3012, 0x02, rw
2	encoder bias vs. electrical position	counts	This parameter needs to be set only if the encoder is used for motor control (a "Motor Encoder"). Otherwise keep this parameter as 0. This parameter specifies a zero offset of the encoder's mechanical installation vs. an electrical position defined by an order in which the motor's phase lines are connected to the controller. The bias can be experimentally determined as the following: • Positively energize phase "A". • Negatively energize both phases "B" and "C". • This can be done by issuing a "Direct Field Control: Electrical	UINT32, 0x3012, 0x03, rw

			Position" command with command parameter 0 rad and a small voltage (e.g. 0.5V). • Let the motor's rotor settle at a position. • The encoder's reading at that position is the bias. The procedure needs to be performed after correcting for a possible inversion of mechanical installation of the encoder (see "inverted installation" parameter).	
3	inverted installation	0 or 1	The direction of the motor's rotation should match the direction of the encoder's rotation. If that's not the case due to a way the encoder is mechanically installed, this parameter helps correct the mismatch.	BOOL, 0x3012, 0x04, rw
4	PWM capture interface	0 or 1	• 0: ENCODERS connector (DEFAULT) • 1: D-HALL connector The controller has two different hardware interfaces for capturing input PWM signals. This parameters tells the controller which of the interfaces the encoder is connected to.	UINT16, 0x3012, 0x0F, rw
5	pulse period	sec or clocks	This parameter specifies the period of the PWM pulses sent by the encoder to the controller. The period should be taken from the encoder's datasheet.	FLOAT32, 0x3012, 0x10, rw
6	max pulse width (angle=360 deg)	sec or clocks	This parameter specifies the duty cycle that corresponds to MAXIMUM position reported by the encoder. This parameter should be taken from the encoder's datasheet.	FLOAT32, 0x3012, 0x11, rw
7	min pulse width (angle=0 deg)	sec or clocks	This parameter specifies the duty cycle that corresponds to ZERO position reported by the encoder. This parameter should be taken from the encoder's datasheet.	FLOAT32, 0x3012, 0x12, rw
8	polarity inversion	0 or 1	The parameter tells the controller to electrically invert the input PWM signal. Leave the default setting unless an application-specific need arises.	BOOL, 0x3012, 0x13, rw

Configuration - Peripheral: RC PWM Input

This is a configuration section for RC PWM interface towards RC radio receivers (Futaba, etc) or microcontrollers/PLCs capable of producing PWM control signal.

If the electrical drive has a properly configured Servo Encoder, the RC PWM input is used as a reference into Servo Control. In other words, the motor is driven as a servo motor.

If the electrical drive does not have a Servo Encoder, the RC PWM input is used as a reference into Electronic Speed Control (ESC).

#	Parameter	Units	Description	CANopen
1	enable	0 or 1	Turns on the RC PWM control interface. Note that both CAN and USB interfaces stop processing commands whenever RC PWM is enabled.	BOOL, 0x3015, 0x01, rw
2	PWM capture interface	0 or 1	• 0: ENCODERS connector • 1: D-HALL connector (DEFAULT) The controller has two different hardware interfaces for capturing input PWM signals. This parameters tells the controller which of the interfaces the RC PWM signal is connected to.	UINT16, 0x3015, 0x0F, rw
3	pulse period	sec or clocks	This parameter specifies the period of the PWM pulses sent to the controller.	FLOAT32, 0x3015, 0x10, rw
4	max pulse width (angle=360 deg)	sec or clocks	This parameter specifies the duty cycle that corresponds to MAXIMUM input.	FLOAT32, 0x3015, 0x11, rw
5	min pulse width (angle=0 deg)	sec or clocks	This parameter specifies the duty cycle that corresponds to ZERO input.	FLOAT32, 0x3015, 0x12, rw
6	inverted installation	0 or 1	Swap the way the controller interprets the max pulse width and min pulse width.	BOOL, 0x3015, 0x04, rw
7	polarity inversion	0 or 1	The parameter tells the controller to electrically invert the input PWM signal. Leave the default setting unless an application-specific need arises.	BOOL, 0x3015, 0x13, rw
8	bidirectional speed	0 or 1	This setting is applicable to Electronic Speed Control (ESC) mode of operation. This settings is ignored in Servo Control mode. • 0: unidirectional speed • 1: bidirectional speed	BOOL, 0x3015, 0x0E, rw

Configuration - Peripheral: Gate Driver

#	Parameter	Units	Description	CANopen
1	Amplifier Gain Selector	0 or 1	• 0: gain = 10 • 1: gain = 40	UINT16, 0x3002, 0x03,

			If a particular motor has a low Maximum Continuous Current rating, then a dynamic range of the controller's ADC module might not be fully utilized. This under-utilization of the dynamic range might create discretization errors when measuring low phase currents. The discretization errors might reduce efficiency of the electrical drive when controlling small motors. To counter this issue, increase the gain of the amplifier whenever the motor's "Maximum Continuous Current" rating is equal or less than 5A. Re-confirm this threshold for your particular model of the controller.	rw

Configuration - Telemetry Mapping: Message 0

This section defines which telemetry parameters are included into TPDO message with COB Function Code 0x180.

#	Parameter	Units	Description	CANopen
1	Parameter 0	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A00, 0x01, rw
2	Parameter 1	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get	UINT32, 0x1A00, 0x02, rw

			the parameter's Telemetry Mapping Code and avoid doing the manual computations.	
3	Parameter 2	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A00, 0x03, rw
4	Parameter 3	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A00, 0x04, rw

Configuration - Telemetry Mapping: Message 1

This section defines which telemetry parameters are included into TPDO message with COB Function Code 0x280.

#	Parameter	Units	Description	CANopen
1	Parameter 0	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note	UINT32, 0x1A01, 0x01, rw

			hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	
2	Parameter 1	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A01, 0x02, rw
3	Parameter 2	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A01, 0x03, rw
4	Parameter 3	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index	UINT32, 0x1A01, 0x04, rw

			and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	

Configuration - Telemetry Mapping: Message 2

This section defines which telemetry parameters are included into TPDO message with COB Function Code 0x380.

#	Parameter	Units	Description	CANopen
1	Parameter 0	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A02, 0x01, rw
2	Parameter 1	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324.	UINT32, 0x1A02, 0x02, rw

			Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	
3	Parameter 2	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A02, 0x03, rw
4	Parameter 3	bits	• 0: Not used • Non-zero: Telemetry Mapping Code Telemetry Mapping Code is a decimal number that is formed from hexadecimal Index and Sub-Index values of a telemetry parameter. First, identify a telemetry parameter to be included into the telemetry message. Note hexadecimal Index and Sub-Index of the telemetry parameter. For example, Index=0x4000 and Sub-Index=0x14. Next, concatenate the hexadecimal numbers to form a single hexadecimal code. In the example, the code is hexadecimal 400014, which is 4000 concatenated with 14. Finally, convert the hexadecimal number into a corresponding decimal number. For example, hexadecimal 400014 is decimal 4194324. Hint: Just right-click on a telemetry parameter on the telemetry screen to quickly get the parameter's Telemetry Mapping Code and avoid doing the manual computations.	UINT32, 0x1A02, 0x04, rw

Configuration - Networking

#	Parameter	Units	Description	CANopen
1	CAN: bitrate	kbps	• 1000 • 500 • 250 • 125 • 100 • 50	UINT16, 0x3000, 0x02, rw

			The parameter defines bitrate in kbps for the CAN interface of the controller. This bit rate should match the bitrates of other devices on the same CAN network.	
2	CANopen: Node ID	1-127	The parameter defines the controller's Node ID on a CANopen network. Valid Node IDs range from 1 to 127 (11 bits only). An initial Node ID is automatically generated. The value can be changed here to ensure uniqueness of the Node ID within a CANopen network.	UINT32, 0x3000, 0x03, rw
3	CANopen: heartbeat timeout	sec	If the controller does not receive any messages via either CAN or USB interfaces within this time period, the device assumes that there is a problem with a parent control system. Upon detecting a heartbeat timeout event, the controller executes the following actions: 1. The controller automatically issues a "Stop" command to itself, 2. ... that stops the motor, 3. ... and depending on the configuration, might enable the "Brake" function.	FLOAT32, 0x3000, 0x04, rw
4	CANopen: telemetry frequency	Hz	The parameter specifies how often the controller sends telemetry messages (CANopen TPDO packets) to a parent control system. The controller sends a single TPDO message at a time, looping across the messages with a frequency defined by this parameter.	FLOAT32, 0x3000, 0x05, rw
5	Feature: USB2CAN routing	0 or 1	This feature turns the device into a USB-to-CAN gateway ("USB2CAN dongle" function). Note that enabling the function puts a performance penalty on the controller.	BOOL, 0x3000, 0x10, rw
6	USB2CAN: support 29bit ID frames	0 or 1	The parameter enables support for routing of 29bit frames between USB and CAN. If the feature is turned off, the built-in USB-to-CAN gateway routes 11bit frames only.	BOOL, 0x3000, 0x12, rw
7	USB: serial number	-	The USB serial number is visible to host computers and can be used to identify the device among multiple devices connected to a host. The serial number is automatically generated, but can be changed here if needed.	UINT32, 0x3000, 0x20, rw

Configuration - Product Activation

#	Parameter	Units	Description	CANopen
1	Activation Key	-	The activation key is provided by technical support team to the buyer of this device. Follow instructions from technical support team to obtain the activation key.	UINT32, 0x20FF, 0x02, rw

Telemetry Parameters

Telemetry - System Status

#	Parameter	Units	Description	CANopen
1	Fault Bits	bits	• 0: No fault • 1: Driver Chip Protection • 2: Driver Chip Overcurrent • 3: =1+2 • 4: Overheating Protection • 8: Overcurrent Protection • 16: Thermistor Overheating • 32: Hall Sensors Fault • 64: Quadrature Encoder Fault • 128: SSI Encoder Fault • 256: SPI Encoder Fault • 512: PWM Encoder Fault • 1024: RC PWM Input Fault • 16384: Emergency Stop • 32768: Activation Key is Missing Whenever a fault is detected, the controller powers off the motor, latches one or more "Fault Bits" flags, and starts waiting for a "Reset" command to come from a parent control system. Until a "Reset" command comes, the motor ignores all other commands received from the parent control system. Note that all configuration management functions (CANopen SDO functionality) keep working as usual. The parent control system is expected to continuously monitor the "Fault Bits" parameter delivered to it via CANopen TPDO mechanism. If the "Fault Bits" parameter is 0 (all bits are clear), then nothing needs to be done. Whenever one or more bits of the "Fault Bits" value indicate a fault, the parent control system is supposed to issue a "Reset" command. The "Reset" command needs to be issued once the fault is corrected, and the drive is ready to re-start operation.	UINT16, 0x4000, 0x03, ro
2	Operation Mode	-	• 0: Idle • 1: Off • 2: Fault • 3: Autoconfiguration • 4: Field Oriented Control (FOC) • 6: Electronic Speed Control (ESC) • 8: Servo Control • 9: Kickstart • 10: Brake	UINT16, 0x4000, 0x02, ro

			• 11: Direct Field Control - Rotation • 12: Direct Field Control - Electrical Position • 15: Testing - FOC Step Response • 16: Testing - Speed Step Response • 17: Testing - Servo Step Response • 19: Brushed Motor or Solenoid Control (1-2 motors) • 20: Testing - Iq Current Step Response This parameter tells what control law the controller is currently using to drive the motor.	
3	Commutation Mode	-	• 1: None (Kickstart) • 2: Sensorless • 3: Hall Sensors • 4: Motor Encoder This parameter tells which method the controller is currently using to determine electrical position of the rotor.	UINT16, 0x4000, 0x15, ro
4	Kickstart needed	0 or 1	This telemetry parameter tells if the motor needs a kickstart at its current state. A kickstart is needed if a sensorless motor is not yet moving or moving slowly, or if a Motor Encoder with Quadrature interface is yet to find its Index position.	BOOL, 0x4000, 0x14, ro
5	Electrical Position	rad	This telemetry parameter tells the position of the rotor in relation to the stator. This position is measured using Hall sensors or a motor encoder, or estimated by "Sensorless Observer" method.	FLOAT32, 0x4000, 0x11, ro
6	Speed (Electrical Frequency)	Hz	This telemetry parameter tells the speed of the motor's rotor. The speed is defined in electrical revolutions per second (Hz). To convert electrical revolutions per second (Hz) to motor shaft's revolutions per second, just divide it by the number of pole pairs. For example, assuming the speed is 20 Hz (electrical), and Poles Number is 8, then the corresponding speed in motor shaft's revolutions per second is 20 / (8/2) = 5.0 (revolutions per second), which is 5 * 60 = 300 RPM.	FLOAT32, 0x4000, 0x10, ro
7	Work Zone: Speed	RPS	This is the speed of rotation of payload measured using a Servo Encoder or estimated using a Motor Encoder, Hall Sensors or via a sensorless technique.	FLOAT32, 0x4000, 0x16, ro
8	Work Zone: Count	counts	The "Work Zone: Count" telemetry is the same as "Servo Encoder Counts" except that it continuously spans in both positive and negative directions. This telemetry output is affected by "Work Zone: zero offset" setting as well as by "inverted installation" and "encoder bias vs. electrical position" parameters of	INT32, 0x4000, 0x19, ro

			the encoder's configuration. The "Work Zone Count" is not limited to Servo Encoder's resolution. Instead, it logically spans in both positive or negative directions as many encoder counts as needed. Both Servo Control and Direct Drive Control use the logical work zone's counts at their references instead of the physical servo encoder's readings. This makes it easier to develop "multi-turn" servo applications.	
9	Work Zone: Count (Legacy Compatibility)	counts	This is the same telemetry parameter as "Work Zone: Count" except that it uses a FLOAT32 data type. This works correctly with encoders that have resolution of 20bits or less only. The telemetry parameter is kept for backward compatibility with existing applications. Consider using "Work Zone: Count" that uses INT32 data type and does not apply restrictions on resolution of the encoders.	FLOAT32, 0x4000, 0x18, ro
10	Work Zone: Turn Count	turns	This telemetry parameter counts full revolutions using the following formula: "Work Zone: Turn Count" = "Work Zone: Count" div "Work Zone: Dimension".	INT32, 0x4000, 0x1B, ro
11	Work Zone: Modulo Count	counts	This is a measurement of a partial revolution as computed using the following modulo division formula: "Work Zone: Modulo" = "Work Zone: Count" mod "Work Zone: Dimension".	UINT32, 0x4000, 0x1A, ro
12	Work Zone: Dimension	counts	This is a size of Work Zone defined in Servo Encoder counts. The value is used for computing "Work Zone: Modulo" and "Work Zone: Turn Count". Usually, this size equals the value of "counts per revolution", a parameter of a Servo Encoder. However, it can be configured to be an arbitrary number to simplify application development. For example, for a CNC machine, this value can be set to match a size of the CNC machine's actual work zone measured in Servo Encoder counts.	UINT32, 0x4000, 0x1C, ro
13	Motor Encoder: Counts	counts	This is an ABSOLUTE POSITION read out from an encoder that plays the "Motor Encoder" role.	UINT32, 0x4000, 0x33, ro
14	Servo Encoder: Counts	counts	This is an ABSOLUTE POSITION read out from an encoder that plays the "Servo Encoder" role.	UINT32, 0x4000, 0x32, ro
15	Reference: Current	A	This is the currently active reference, a commanded target, for current flowing through the motor.	FLOAT32, 0x4000, 0x20, ro
16	Reference: Speed	Hz	This is the currently active reference, a commanded target, for the motor's	FLOAT32,

	(Electrical Frequency)		speed.	0x4000, 0x21, ro
17	Reference: Work Zone Count	counts	This is the currently active reference, a commanded target, for the servo actuator.	INT32, 0x4000, 0x23, ro
18	Target Reached	0 or 1	• 0: Not reached • 1: Reached This indicator tells if a reference servo position has been reached. This indicator does not latch.	BOOL, 0x4000, 0x41, ro
19	Sample Number	-	This is a continuously incremented counter of samples taken by an ADC module of the controller. The counter is incremented with the sampling frequency of the controller.	UINT32, 0x4000, 0x04, ro

Telemetry - ADC

#	Parameter	Units	Description	CANopen
1	Voltage: DC bus (Udc)	V	Measured voltage of input power supply.	FLOAT16, 0x5001, 0x04, ro
2	Current: Phase A (Ia)	A	Measured electric current flowing through phase A.	FLOAT32, 0x5001, 0x05, ro
3	Current: Phase B (Ib)	A	Measured electric current flowing through phase B.	FLOAT32, 0x5001, 0x06, ro
4	Current: Phase C (Ic)	A	Measured electric current flowing through phase C.	FLOAT32, 0x5001, 0x07, ro
5	CPU Temperature	C	Measured temperature of the controller's CPU.	INT16, 0x5001, 0x0C, ro
6	ADC mode	0 or 1	• 0: Calibration is ongoing • 1: Operational	BOOL, 0x5001, 0x03, ro

Telemetry - Field Oriented Control (FOC)

#	Parameter	Units	Description	CANopen
1	Torque	N*m	This is an estimate of TORQUE that the motor is currently generating. Note that this estimation is only available whenever the motor runs under Field Oriented Control (FOC).	FLOAT32, 0x4001, 0x02, ro
2	FOC: Iq Current	A	This is an estimate of Iq current derived from phase currents measured by the ADC.	FLOAT32, 0x4001, 0x03, ro
3	FOC: Id Current	A	This is an estimate of Id current derived from phase currents measured by the ADC.	FLOAT32, 0x4001, 0x04, ro
4	FOC: Id Reference	V	This is a currently commanded Id reference. This telemetry parameter is useful for tuning "Feature: Field Weakening".	FLOAT32, 0x4001, 0x09, ro
5	FOC: Uq Voltage	V	This is a Uq voltage, an output of a FOC control law.	FLOAT32, 0x4001, 0x05, ro
6	FOC: Uq Integral	V	This is an integral sum of a PI controller that commands Uq voltage to stabilize Iq current.	FLOAT32, 0x4001, 0x07, ro
7	FOC: Ud Voltage	V	This is a Ud voltage, an output of a FOC control law.	FLOAT32, 0x4001, 0x06, ro
8	FOC: Ud Integral	V	This is an integral sum of a PI controller that commands Ud voltage to stabilize Id current.	FLOAT32, 0x4001, 0x08, ro
9	Moment of Inertia (Rotor and Payload)	kg*m2	This is an estimated (measured) "Moment of Inertia of Rotor and Payload".	FLOAT32, 0x4001, 0x0D, ro
10	Viscous Damping Constant	Nm/Hz	This is an estimated (measured) "Viscous Damping Constant".	FLOAT32, 0x4001, 0x0E, ro

Telemetry - Direct Field Control

#	Parameter	Units	Description	CANopen
1	Direct Field Control: Electrical Position	rad	This telemetry parameter tells an estimate of position of the rotor in relation to the stator.	FLOAT32, 0x4008, 0x03, ro
2	Direct Field Control: Speed (Electrical Frequency)	Hz	This telemetry parameter tells an estimate of speed of the motor. The speed is defined in electrical revolutions per second (Hz). To convert electrical revolutions per second (Hz) to motor shaft's revolutions per second, just divide it by the number of pole pairs. For example, assuming the speed is 20 Hz (electrical), and Poles Number is 8, then the corresponding speed in motor shaft's revolutions per second is 20 / (8/2) = 5.0 Hz (revolutions per second), which is 5 * 60 = 300 RPM.	FLOAT32, 0x4008, 0x04, ro

Telemetry - Sensorless Observer

#	Parameter	Units	Description	CANopen
1	Sensorless Observer: mode	-	• 0: Undefined state • 1: Zero speed • 2: Operational speed	UINT16, 0x4007, 0x02, ro
2	Sensorless Observer: Speed (Electrical Frequency)	Hz	This telemetry parameter tells an estimated speed of the "sensorless" motor. The speed is defined in electrical revolutions per second (Hz). To convert electrical revolutions per second (Hz) to motor shaft's revolutions per second, just divide it by the number of pole pairs. For example, assuming the speed is 20 Hz (electrical), and Poles Number is 8, then the corresponding speed in motor shaft's revolutions per second is 20 / (8/2) = 5.0 Hz (revolutions per second), which is 5 * 60 = 300 RPM.	FLOAT32, 0x4007, 0x03, ro
3	Sensorless Observer: Electrical Position	rad	This telemetry parameter tells an estimated position of the rotor in relation to the stator.	FLOAT32, 0x4007, 0x04, ro
4	Sensorless Observer: Emf alpha	V	This is an estimate of Back-Emf voltage (the "sine" component).	FLOAT32, 0x4007, 0x05, ro

5	Sensorless Observer: Emf beta	V	This is an estimate of Back-Emf voltage (the "cosine" component).	FLOAT32, 0x4007, 0x06, ro
6	Sensorless Observer: Back-Emf Constant (Ke)	V/ (rad/s)	This is an estimate of "Back-Emf Constant (Ke)" measured in V (peak, line-to-neutral) per electrical rad/s.	FLOAT32, 0x4007, 0x07, ro
7	Sensorless Observer: Technical Speed Limit	Hz	This is an estimate of MAXIMUM speed the Sensorless Observer is capable of measuring. The limit is caused by finite performance of CPU, and a finite sampling frequency of ADC.	FLOAT32, 0x4007, 0x09, ro
8	Sensorless Observer: Zero Speed Threshold	Hz	This is an estimate of MINIMUM speed at which the Sensorless Observer is still capable of sensing Back-Emf voltages in the presence of background noise.	FLOAT32, 0x4007, 0x08, ro

Telemetry - Hall Sensors Observer

#	Parameter	Units	Description	CANopen
1	Hall Observer: mode	-	• 0: Zero speed • 1: Low speed • 2: Operational speed	UINT16, 0x4002, 0x02, ro
2	Hall Observer: Hall code	-	Hall sensors generate 6 codes (counts) per electrical revolution. This telemetry parameter tells the current Hall code.	UINT16, 0x4002, 0x03, ro
3	Hall Observer: Speed (Electrical Frequency)	Hz	This telemetry parameter tells an estimated speed of the motor. The speed is defined in electrical revolutions per second (Hz). To convert electrical revolutions per second (Hz) to motor shaft's revolutions per second, just divide it by the number of pole pairs. For example, assuming the speed is 20 Hz (electrical), and Poles Number is 8, then the corresponding speed in motor shaft's revolutions per second is 20 / (8/2) = 5.0 Hz (revolutions per second), which is 5 * 60 = 300 RPM.	FLOAT32, 0x4002, 0x04, ro
4	Hall Observer: Electrical Position	rad	This telemetry parameter tells an estimated position of the rotor in relation to the stator.	FLOAT32, 0x4002, 0x05, ro
5	Hall Observer: between codes counter	samples	This is a counter that is reset each time a Hall code changes.	UINT32, 0x4002, 0x07, ro

6	Hall Observer: Hall errors counter	-	This telemetry parameter counts occurrences of erroneous combinations of Hall sensors' readings. Note that the combinations with all "ones" or all "zeros" correspond to faulty or disconnected Hall sensors.	UINT32, 0x4002, 0x08, ro
7	Hall Observer: positioning error estimate	rad	Whenever a Hall code changes, an electrical position that corresponds to the code is compared to an estimate of the electrical position made by the Observer, and the difference is shown here. The smaller the difference, the better.	FLOAT32, 0x4002, 0x09, ro
8	Hall Observer: count	-	Hall sensors generate 6 counts per electrical revolution. This telemetry parameter tells the current count as if Hall Sensors were an absolute encoder.	UINT16, 0x4002, 0x0B, ro
9	Hall Observer: Technical Speed Limit	Hz	This is an estimate of MAXIMUM speed the Hall Observer is capable of measuring. The limit is caused by finite performance of CPU, and a finite sampling frequency of ADC.	FLOAT32, 0x4002, 0x06, ro
10	Hall Observer: Zero Speed Threshold	Hz	This is an estimate of MINIMUM speed the Hall Observer is capable of measuring. The limit is caused by a low resolution of Hall Sensors that generate just 6 counts per electrical revolution.	FLOAT32, 0x4002, 0x0A, ro

Telemetry - Peripheral: Hall Sensors

#	Parameter	Units	Description	CANopen
1	Hall Sensor #0	-	Reading of physical Hall sensor 0	UINT16, 0x5005, 0x03, ro
2	Hall Sensor #1	-	Reading of physical Hall sensor 1	UINT16, 0x5005, 0x04, ro
3	Hall Sensor #2	-	Reading of physical Hall sensor 2	UINT16, 0x5005, 0x05, ro
4	Hall Sensors Fault	0 or 1	Note that combinations with all "ones" or all "zeros" correspond to faulty or disconnected Hall sensors.	BOOL, 0x5005, 0x02, ro

Telemetry - Peripheral: Quadrature Encoder

#	Parameter	Units	Description	CANopen
1	Quadrature: count	counts	The encoder's ABSOLUTE POSITION expressed in counts (quadrature edge counts).	UINT32, 0x500A, 0x06, ro
2	Quadrature: direction	0 or 1	The direction of the rotation of the encoder's disk	BOOL, 0x500A, 0x05, ro
3	Quadrature: index detected	0 or 1	This bit latches whenever the quadrature encoder detects its INDEX position for the first time. Until the INDEX signal is detected, the encoder cannot measure an absolute position of the shaft since it does not have a reference point that corresponds to a zero position.	BOOL, 0x500A, 0x0A, ro
4	Quadrature: signal A	0 or 1	Reading of signal "A"	BOOL, 0x500A, 0x02, ro
5	Quadrature: signal B	0 or 1	Reading of signal "B"	BOOL, 0x500A, 0x03, ro
6	Quadrature: signal I	0 or 1	Reading of INDEX signal	BOOL, 0x500A, 0x04, ro
7	Quadrature: phase errors counter	-	This is a counter of phase errors in quadrature signals.	UINT32, 0x500A, 0x0B, ro
8	Quadrature: encoder speed	RPS	The speed of quadrature disk rotation expressed in revolutions per second (Hz). This speed is estimated using "UNIT TIME" or "UNIT DISTANCE" method (see "Peripheral: Quadrature Encoder" configuration section for details).	FLOAT32, 0x500A, 0x08, ro
9	Quadrature: encoder speed via UNIT DISTANCE method	RPS	The speed of quadrature disk rotation computed using "UNIT DISTANCE" method.	FLOAT32, 0x500A, 0x0C, ro
10	Quadrature: encoder	RPS	The speed of quadrature disk rotation computed using "UNIT TIME"	FLOAT32,

	speed via UNIT TIME method		method.	0x500A, 0x0D, ro
11	Quadrature: time to travel UNIT DISTANCE	sec	This parameter helps monitor performance of UNIT DISTANCE speed computation method.	FLOAT32, 0x500A, 0x0F, ro
12	Quadrature: distance traveled in UNIT TIME	counts	This parameter helps monitor performance of UNIT TIME speed computation method.	FLOAT32, 0x500A, 0x0E, ro

Telemetry - Peripheral: SSI/BISS-C Encoder

#	Parameter	Units	Description	CANopen
1	SSI/BISS-C Encoder: packets counter	-	This is a counter of data packets received by the controller from the encoder.	UINT32, 0x5008, 0x02, ro
2	SSI/BISS-C Encoder: count	counts	This is the latest ABSOLUTE POSITION reported by the encoder.	UINT32, 0x5008, 0x03, ro
3	SSI/BISS-C Encoder: is count valid	0 or 1	This bit tells if the latest absolute position is valid, meaning there is no ERROR bit and no CRC error.	BOOL, 0x5008, 0x04, ro
4	SSI/BISS-C Encoder: turn count	turns	This is the latest MULTI-TURN count reported by the encoder.	UINT32, 0x5008, 0x0B, ro
5	SSI/BISS-C Encoder: is turn count valid	0 or 1	This bit tells if the latest MULTI-TURN count is valid.	BOOL, 0x5008, 0x0C, ro
6	SSI/BISS-C Encoder: error bit	0 or 1	This is a value of latest ERROR bit reported by the encoder.	BOOL, 0x5008, 0x05, ro
7	SSI/BISS-C Encoder: warn bit	0 or 1	This is a value of latest WARN bit reported by the encoder.	BOOL, 0x5008, 0x06, ro
8	SSI/BISS-C Encoder: extracted CRC	-	This is a value of latest CRC field reported by the encoder.	UINT16, 0x5008, 0x07, ro

9	SSI/BISS-C Encoder: computed CRC	-	This is the latest CRC value computed by the controller.	UINT16, 0x5008, 0x08, ro
10	SSI/BISS-C Encoder: CRC mismatch error	0 or 1	This bit is raised whenever CRC verification fails.	BOOL, 0x5008, 0x09, ro
11	SSI/BISS-C Encoder: CRC mismatch counter	-	This counter is increased each time CRC verification fails.	UINT32, 0x5008, 0x0A, ro

Telemetry - Peripheral: SPI Encoder

#	Parameter	Units	Description	CANopen
1	SPI Encoder: packets counter	-	This is a counter of data packets received by the controller from the encoder.	UINT32, 0x5006, 0x02, ro
2	SPI Encoder: count	counts	This is the latest ABSOLUTE POSITION reported by the encoder.	UINT32, 0x5006, 0x03, ro
3	SPI Encoder: is count valid	0 or 1	This bit tells if the latest absolute position is valid, meaning there is no ERROR bit and no CRC error.	BOOL, 0x5006, 0x04, ro
4	SPI Encoder: turn count	turns	This is the latest MULTI-TURN count reported by the encoder.	UINT32, 0x5006, 0x0B, ro
5	SPI Encoder: is turn count valid	0 or 1	This bit tells if the latest MULTI-TURN count is valid.	BOOL, 0x5006, 0x0C, ro
6	SPI Encoder: error bit	0 or 1	This is a value of latest ERROR bit reported by the encoder.	BOOL, 0x5006, 0x05, ro
7	SPI Encoder: warn bit	0 or 1	This is a value of latest WARN bit reported by the encoder.	BOOL, 0x5006, 0x06, ro
8	SPI Encoder: extracted CRC	-	This is a value of latest CRC field reported by the encoder.	UINT16, 0x5006, 0x07,

				ro
9	SPI Encoder: computed CRC	-	This is the latest CRC value computed by the controller.	UINT16, 0x5006, 0x08, ro
10	SPI Encoder: CRC mismatch error	0 or 1	This bit is raised whenever CRC verification fails.	BOOL, 0x5006, 0x09, ro
11	SPI Encoder: CRC mismatch counter	-	This counter is increased each time CRC verification fails.	UINT32, 0x5006, 0x0A, ro

Telemetry - Peripheral: PWM Encoder

#	Parameter	Units	Description	CANopen
1	PWM Encoder: state	-	• 0: Undefined • 1: No interrupts • 2: Operational • 3: Fault - No Pulse • 4: Fault - PWM Duty Out of Range	UINT16, 0x5007, 0x02, ro
2	PWM Encoder: sample counter	-	This counter is incremented each time a pair of pulses is sampled.	UINT32, 0x5007, 0x03, ro
3	PWM Encoder: count	counts	This is the latest ABSOLUTE POSITION reported by the encoder.	UINT32, 0x5007, 0x08, ro
4	PWM Encoder: ratio	0 to 1	Received signal as a value in the range [0:1].	FLOAT32, 0x5007, 0x0F, ro
5	PWM Encoder: pulse period	sec	Measured period of the latest pulse	FLOAT32, 0x5007, 0x0B, ro
6	PWM Encoder: pulse width	sec	Measured duty cycle (in seconds) of the latest pulse	FLOAT32, 0x5007, 0x0A, ro
7	PWM Encoder: pulse period (CAP2)	CPU ticks	A silicon-specific value	UINT32, 0x5007, 0x05, ro

8	PWM Encoder: pulse width (CAP1)	CPU ticks	A silicon-specific value	UINT32, 0x5007, 0x04, ro
9	PWM Encoder: pulse period (CAP4)	CPU ticks	A silicon-specific value	UINT32, 0x5007, 0x07, ro
10	PWM Encoder: pulse width (CAP3)	CPU ticks	A silicon-specific value	UINT32, 0x5007, 0x06, ro
11	PWM Encoder: fault detected	0 or 1	A silicon-specific value	BOOL, 0x5007, 0x09, ro
12	PWM Encoder: overflow errors counter	-	This counter is incremented each time a period of a pulse cannot be measured due to a timeout while waiting for a pulse's edge.	UINT32, 0x5007, 0x0C, ro

Telemetry - Peripheral: RC PWM Input

#	Parameter	Units	Description	CANopen
1	RC PWM Input: state	-	• 0: Undefined • 1: No interrupts • 2: Operational • 3: Fault - No Pulse • 4: Fault - PWM Duty Out of Range	UINT16, 0x500B, 0x02, ro
2	RC PWM Input: sample counter	-	This counter is incremented each time a pair of pulses is sampled.	UINT32, 0x500B, 0x03, ro
3	RC PWM Input: ratio	0 to 1	Received signal as a value in the range [0:1].	FLOAT32, 0x500B, 0x0F, ro
4	RC PWM Input: pulse period	sec	Measured period of the latest pulse	FLOAT32, 0x500B, 0x0B, ro
5	RC PWM Input: pulse width	sec	Measured duty cycle (in seconds) of the latest pulse	FLOAT32, 0x500B, 0x0A, ro
6	RC PWM Input: pulse	CPU	A silicon-specific value	UINT32,

	period (CAP2)	ticks		0x500B, 0x05, ro
7	RC PWM Input: pulse width (CAP1)	CPU ticks	A silicon-specific value	UINT32, 0x500B, 0x04, ro
8	RC PWM Input: pulse period (CAP4)	CPU ticks	A silicon-specific value	UINT32, 0x500B, 0x07, ro
9	RC PWM Input: pulse width (CAP3)	CPU ticks	A silicon-specific value	UINT32, 0x500B, 0x06, ro
10	RC PWM Input: fault detected	0 or 1	A silicon-specific value	BOOL, 0x500B, 0x09, ro
11	RC PWM Input: overflow errors counter	-	This counter is incremented each time a period of a pulse cannot be measured due to a timeout while waiting for a pulse's edge.	UINT32, 0x500B, 0x0C, ro

Telemetry - Peripheral: GPIO

#	Parameter	Units	Description	CANopen
1	Emergency Stop Switch	0 or 1	Status of Emergency Stop Signal	UINT16, 0x5009, 0x04, ro
2	Limit Switch: Negative Speed	0 or 1	Status of Limit Switch in NEGATIVE speed direction.	UINT16, 0x5009, 0x05, ro
3	Limit Switch: Positive Speed	0 or 1	Status of Limit Switch in POSITIVE speed direction.	UINT16, 0x5009, 0x06, ro
4	Generic Input	0 or 1	Latest value read out from a dedicated GPIO input.	UINT16, 0x5009, 0x10, ro

Telemetry - Peripheral: Inverter

#	Parameter	Units	Description	CANopen
1	Inverter mode	0 or 1	• 0: Inverter PWM is OFF	BOOL,

			• 1: Inverter PWM is ON	0x5002, 0x02, ro
2	CMPR A	CPU ticks	A silicon-specific value	UINT16, 0x5002, 0x03, ro
3	CMPR B	CPU ticks	A silicon-specific value	UINT16, 0x5002, 0x04, ro
4	CMPR C	CPU ticks	A silicon-specific value	UINT16, 0x5002, 0x05, ro

Telemetry - Peripheral: Gate Driver

#	Parameter	Units	Description	CANopen
1	driver: mode	0 or 1	• 0: Gate Driver is OFF • 1: Gate Driver is ON	BOOL, 0x5004, 0x02, ro
2	driver: fault	0 or 1	This is a latest reading of the "FAULT" signal coming from the Gate Driver chip.	BOOL, 0x5004, 0x03, ro
3	driver: overcurrent or overtemperature	0 or 1	This is a latest reading of the "OVERCURRENT or OVERTEMPERTATURE" signal coming from the Gate Driver chip.	BOOL, 0x5004, 0x04, ro

Telemetry - Networking

#	Parameter	Units	Description	CANopen
1	CAN: received packets	-	This counter is incremented each time the device receives a CAN packet.	UINT32, 0x5010, 0x02, ro
2	CAN: sent packets	-	This counter is incremented each time the device sends a CAN packet.	UINT32, 0x5010, 0x03, ro
3	USB: status	0 or 1	• 0: Disconnected • 1: Connected	BOOL, 0x5010, 0x12, ro
4	USB: received packets	-	This counter is incremented each time the device receives a USB packet.	UINT32,

				0x5010, 0x13, ro
5	USB: sent packets	-	This counter is incremented each time the device sends a USB packet.	UINT32, 0x5010, 0x14, ro
6	USB: sending errors	-	This counter is incremented each time the device is not able to send a USB packet for any reason.	UINT32, 0x5010, 0x15, ro
7	USB2CAN: can->usb packets	-	This counter is incremented each time a packet received from CAN is forwarded to a USB host.	UINT32, 0x5010, 0x17, ro
8	USB2CAN: usb->can packets	-	This counter is incremented each time a USB packet is forwarded via CAN.	UINT32, 0x5010, 0x18, ro

Telemetry - Device Information

#	Parameter	Units	Description	CANopen
1	Device Type	-	The value is used when ACTIVATING the device. Please send this number to technical support team to obtain an activation key.	UINT32, 0x5000, 0x9F, ro
2	Device Serial Number	-	The serial number is used when ACTIVATING the device. Please send this number to technical support team to obtain an activation key.	UINT32, 0x5000, 0xA0, ro
3	Firmware Serial Number	-	The serial number is used when ACTIVATING the device. Please send this number to technical support team to obtain an activation key.	UINT32, 0x5000, 0xA1, ro
4	Firmware Version	-	Firmware release date. For example: 20151230 means 2015-12-30.	UINT32, 0x5000, 0xA7, ro
5	Sampling Frequency	Hz	This is the sampling frequency as well as the control loop frequency of the servo drive.	FLOAT32, 0x5000, 0xA2, ro
6	Inverter PWM Period	CPU ticks	This is a period of the inverter's PWM signals of the controller.	UINT16, 0x5000, 0xA3, ro

7	ADC ISR Execution Time	CPU ticks	Measured performance of an ADC interrupt servicing routine	UINT32, 0x5000, 0xA4, ro
8	Timer ISR Execution Time	CPU ticks	Measured performance of a Timer interrupt servicing routine	UINT32, 0x5000, 0xA5, ro
9	USB RX ISR Execution Time	CPU ticks	Measured performance of a USB interrupt servicing routine	UINT32, 0x5000, 0xA6, ro

Commands

Command - Electronic Speed Control (ESC), Hz

The "Electronic Speed Control (ESC), Hz" command instructs the controller to drive a brushless/brushed motor at a constant speed. The controller automatically increases or decreases torque to maintain the constant speed.

Note that the speed here is defined in electrical revolutions per second (Hz). To convert the electrical revolutions per second (Hz) to motor shaft's revolutions per second, just divide it by the number of pole pairs. For example, assuming the speed is 20 Hz (electrical), and Poles Number is 8, then the corresponding speed in motor shaft's revolutions per second is 20 / (8/2) = 5.0 Hz (revolutions per second), which is 5 * 60 = 300 RPM.

	Parameter	Units	Description	Data type	Position in Payload
1	Speed (Electrical Frequency)	Hz	This is a speed reference defined in electrical revolutions per second.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x20.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Electronic Speed Control (ESC), RPM

The "Electronic Speed Control (ESC), RPM" command instructs the controller to drive a brushless/brushed motor at a constant speed defined in motor shaft's revolutions per minute (RPM). The controller automatically increases or decreases torque to maintain the constant speed.

Under the hood the controller converts the RPM reference into an electrical revolutions per second (Hz), and issues itself an "Electronic Speed Control (ESC)" command. Properly set the "Poles Number" parameter in the "Datasheet" section before using this command, since the configuration parameter is used to perform the conversion.

	Parameter	Units	Description	Data type	Position in Payload
1	Speed (Revolutions per Minute)	RPM	This is a speed reference defined in motor shaft's revolutions per minute.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x24.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Servo (Legacy Compatibility)

This command is the same as the "Servo" command except that it takes a FLOAT32 as a parameter rather than an INT32. This command works with servo encoders that have resolution equal or less than 20bit only. This command is maintained for compatibility with legacy systems. Consider using the "Servo" command instead.

	Parameter	Units	Description	Data type	Position in Payload
1	Work Zone Position	counts	This parameter specifies a target position the servo actuator is going to move to. Note that the position is defined in Work Zone "multi-turn" counts meaning that the servo might make multiple revolutions to reach the target position. The position can be a positive or a negative value.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x30.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Current Control / Field Oriented Control (FOC)

This command instructs the controller to drive a constant electrical current through a brushless/brushed motor. Use this command to directly control electrical current flowing through a motor. The constant electrical current means a constant torque generated by the motor. The command might causes the motor to continuously accelerate.

	Parameter	Units	Description	Data type	Position in Payload
1	Current	A	The "Current" parameter is the commanded electrical current to be driven through the brushless/brushed motor. This value should be equal or less than the "Maximum Continuous Current" configuration parameter.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x10.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Electronic Torque Control (ETC)

This commands instructs the controller to drive a brushless motor in such a way that the motor generates a specific constant torque. Use this command to directly control torque of a brushless motor. Note that the command might cause the motor to continuously accelerate.

Under the hood the controller converts the torque reference into an electrical current reference, and issues itself a "Field Oriented Control (FOC)" command. Properly set the "Back-Emf Constant (Ke)" and "Poles Number" parameters in the "Datasheet" section before using this command since the parameters are needed when converting the torque reference to an electrical current reference.

	Parameter	Units	Description	Data type	Position in Payload
1	Torque	N*m	The "Torque" parameter is the constant torque that the brushless motor is commanded to produce.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x14.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Direct Field Control: Rotation

The "Direct Field Control: Rotation" command instructs the controller to use the coils of a brushless motor to create a magnetic field inside the motor, and then ROTATE the magnetic field with a given speed. What happens next is that permanent magnets of the rotor get attracted to the rotating magnetic field of the coils. The rotor starts following the rotation of the magnetic field. Note that this way of moving the rotor is inefficient from energy point of view as compared to Field Orieted Control (FOC), and can lead to heating the motor. However this mode of operation is useful in some practical applications such as gyro-stabilization of optical payloads.

	Parameter	Units	Description	Data type	Position in Payload
1	Voltage	V	This parameter specifies a voltage that the controller applies to the coils of the stator to create a magnetic field inside the motor. The higher the voltage is, the stronger the electric current in the coils is, the stronger the rotor is attracted to the rotating magnetic field. ATTENTION: Wrong voltage burns brushless motors. The coils of brushless motors tend to have low electrical resistance. Even a small voltage (0.20-0.30 V) can create an electric current strong enough to burn the coils. If you are not sure what a safe voltage is, start with a small voltage (0.10 V), and gradually raise it, while observing electrical currents flowing through the motor's phases using the controller's telemetry screen. The electrical currents should not be stronger than "Maximum Continuous Current" limit of the motor, a datasheet value.	FLOAT16	2
2	Speed (Electrical Frequency)	Hz	This parameter defines a speed with which the controller rotates the magnetic field created inside the motor using the coils of the motor.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x40.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Direct Field Control: Electrical Position

The "Direct Field Control: Electrical Position" command instructs the controller to use the coils of a brushless motor to create a STATIC magnetic field inside the motor. The rotor of the motor aligns itself with such a static magnetic field. Note that this way of positioning the rotor is inefficient from the energy point of view, and can lead to heating the motor. However this mode of operation is useful any many practical applications such as gyro-stabilization of optical payloads.

	Parameter	Units	Description	Data type	Position

					in Payload
1	Voltage	V	This parameter specifies a voltage that the controller applies to the coils of the stator to create a STATIC magnetic field inside the motor. The higher the voltage is, the stronger the electrical current in the coils is, the stronger the rotor is attached to the static magnetic field. ATTENTION: Wrong voltage burns brushless motors. The coils of brushless motors tend to have low electrical resistance. Even a small voltage (0.20-0.30 V) can create an electric current strong enough to burn the coils. If you are not sure what a safe voltage is, start with a small voltage (0.10 V), and gradually raise it, while observing electrical currents flowing through the motor's phases using the controller's telemetry screen. The electrical currents should not be stronger than "Maximum Continuous Current" limit of the motor, a datasheet value.	FLOAT16	2
2	Electrical Position	rad	This parameter specifies an electrical position of the static magnetic field to be created using the coils of the stator.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x44.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Kickstart

The "Kickstart" command accelerates a "sensorless" brushless motor to a target speed using a control technique that that does require any knowledge of the rotor's position. This command is intended for use with "sensorless" brushless motors. Note the "Kickstart" procedure is configured in "Control Laws" section.

	Parameter	Units	Description	Data type	Position in Payload
1	Speed (Electrical Frequency)	Hz	This parameter defines a target speed. The speed is expressed in electrical revolutions per second (Hz).	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0x58.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Reset

The command clears "Fault Bits" latches, powers off the motor, resets the inverter circuitry, and resets the Work Zone multi-turn position. Use this command to clear fault flags whenever Fault Bits telemetry indicates a fault, to reset servo position within the work zone, or as a "panic button" to power off the motor in an emergency.

Whenever a fault is detected, the controller powers off the motor, latches one or more "Fault Bits" flags, and starts waiting for a "Reset" command to come from a parent control system. Until a "Reset" command comes, the motor ignores all other commands received from the parent control system. Note that all configuration management functions (CANopen SDO functionality) keep working as usual. The parent control system is expected to continuously monitor the "Fault Bits" parameter streamed to it via CANopen TPDO mechanism. If the "Fault Bits" parameter is 0 (all bits are clear), then nothing needs to be done. Whenever one or more bits of the "Fault Bits" telemetry indicate a fault, the parent control system is supposed to issue a "Reset" command once the fault is addressed, and the drive is ready to re-start operation.

This command does not have parameters.

The Function Code for COB-ID is 0x200.

The Command Code is 0x01.

It is generally not required to continuously send this command to the device.

Command - Reset Work Zone

The command programmatically resets the multi-turn Work Zone position counter back to the zero turn.

This command does not have parameters.

The Function Code for COB-ID is 0x200.

The Command Code is 0xE0.

It is generally not required to continuously send this command to the device.

Command - Brake

The "Brake" command instructs the controller to start using the drive's electric motor to prevent motion of the drive's shaft under influence of external forces. The controller starts dynamically positioning electromagnetic fields inside the motor in such a way that any significant motion of the shaft is countered by an electromagnetic force working in the opposite direction. This is like applying a brake to the shaft, but without an actual physical braking device. If there is no external force, the "Brake" command does not trigger any countering electromagnetic forces, and thus does not draw energy from the power supply.

For the braking function to work efficiently, the controller uses Hall sensors or a "Motor Encoder" to detect that the shaft is moving due to external forces, and to dynamically apply a countering electromagnetic force. Note that if a motor does not have Hall sensors or a "Motor Encoder", then the controller defaults to using a statically positioned magnetic field when holding the shaft of the motor. The statically positioned magnetic field requires an electric current to be continuously driven through the coils of the motor regardless of the presence of external forces. This electric current might cause excessive heating of the sensorless motor, and cause a continuous drain of energy from its power supply. In short, special care needs to be taken when using the "Brake" command with sensorless motors.

This command does not have parameters.

The Function Code for COB-ID is 0x200.

The Command Code is 0x50.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Stop

This command instructs the controller to stop a running motor in a controllable way.

This command does not have parameters.

The Function Code for COB-ID is 0x200.

The Command Code is 0x04.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Off

This command powers off the inverter circuitry which means all connected motors or solenoids are powered off.

This command does not have parameters.

The Function Code for COB-ID is 0x200.

The Command Code is 0x06.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - GPIO: PWM output

This command increases or decreases duty cycle of a PWM signal on a dedicated GPIO output pin. The command is typically used to control solenoids/brakes connected via the GPIO output pin.

	Parameter	Units	Description	Data type	Position in Payload
1	Duty Cycle	0.0-1.0	This parameter specifies a duty cycle of the PWM signal. The value must be a real number between 0.0 (fully open) and 1.0 (fully closed). For example, Duty Cycle=0.40 means 40% duty cycle of the output PWM signal.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0xA4.

It is generally not required to continuously send this command to the device.

Command - Testing: Iq Current Step Response

This command initiates a load test of PI controller of Iq current. The load test is typically run to verify that the relevant control laws are configured properly. The load test routine continuously changes commanded CURRENT reference ("Iq current") according to a SQUARE WAVE profile. Note that the motor may produce clicking sounds while the test is running. Use "Stop" or "Reset" command to terminate the testing procedure.

	Parameter	Units	Description	Data type	Position in

					Payload
1	Period	sec	The parameter specifies a period of a SQUARE WAVE that produces an "Iq current" reference for the testing procedure.	FLOAT16	2
2	Amplitude: Iq current	A	The parameter specifies an amplitude of a SQUARE WAVE that produces an "Iq current" reference for the testing procedure.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0xBC.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Testing: Field Oriented Control (FOC) Step Response

This command initiates a load test of "Field Oriented Control (FOC)" function of the controller. The load test is typically run to verify that the relevant control laws are configured properly. The load test routine continuously changes commanded CURRENT reference ("Iq current") according to a SQUARE WAVE profile. Note that the motor accelerates and decelerates repeatedly while the test is running. Use "Stop" or "Reset" command to terminate the testing procedure.

	Parameter	Units	Description	Data type	Position in Payload
1	Period	sec	The parameter specifies a period of a SQUARE WAVE that produces an "Iq current" reference for the testing procedure.	FLOAT16	2
2	Amplitude: Iq current	A	The parameter specifies an amplitude of a SQUARE WAVE that produces an "Iq current" reference for the testing procedure.	FLOAT32	4

The Function Code for COB-ID is 0x200.

The Command Code is 0xB0.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Testing: Electronic Speed Control (ESC) Step Response

This command initiates a load test of "Electronic Speed Control (ESC)" function of the controller. The load test is typically run to verify that the relevant control laws are configured properly. The load test routine continuously changes commanded SPEED reference according to a SQUARE WAVE profile. Note that the motor accelerates and decelerates repeatedly while the test is running. Use "Stop" or "Reset" command to terminate the testing procedure.

	Parameter	Units	Description	Data type	Position in Payload
1	Period	sec	The parameter specifies a period of a SQUARE WAVE that produces an SPEED reference for the testing procedure.	FLOAT16	2
2	Amplitude:	Hz	The parameter specifies an amplitude of a SQUARE WAVE that	FLOAT32	4

	Speed		produces an SPEED reference for the testing procedure.		

The Function Code for COB-ID is 0x200.

The Command Code is 0xB4.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Servo

The "Servo" command instructs the controller to move the output of a servo actuator to a specified target position defined in Work Zone Counts, and keep that position upon reaching it. Note that a speed with which the servo actuator moves is defined by "Servo: Speed Limit" configuration parameter that can be changed in runtime if needed.

	Parameter	Units	Description	Data type	Position in Payload
1	Target: Work Zone Count	counts	This parameter specifies a target position the servo actuator is going to move to. Note that the position is defined in Work Zone "multi-turn" counts meaning that the servo might make multiple revolutions to reach the target position. The position can be a positive or a negative value.	INT32	4

The Function Code for COB-ID is 0x300.

The Command Code is 0x31.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Servo: Modulo

This command moves the output shaft of a servo actuator to a specified target position defined in Work Zone Modulo Counts. This command is similar to the "Servo" command except that it takes a single-turn rotary target position.

	Parameter	Units	Description	Data type	Position in Payload
1	Target: Work Zone Modulo	counts	This parameter specifies a target cyclical position the servo actuator is going to move to.	INT32	4
2	Direction	0,1,2,3	• 0: NORMAL, similar to linear actuator, same as "Servo" command • 1: motion in NEGATIVE direction • 2: motion in POSITIVE direction • 3: take a SHORTEST way (optimized)	INT16	2

The Function Code for COB-ID is 0x300.

The Command Code is 0x32.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Servo: Turns and Modulo

This command moves the output shaft of a servo actuator to a specified target position defined in Work Zone Turns and Modulo Counts. This command is similar to the "Servo" command.

	Parameter	Units	Description	Data type	Position in Payload
1	Target: Work Zone: Modulo	counts	This parameter specifies a target cyclical position the servo actuator is going to move to.	INT32	4
2	Target: Work Zone: Turns	turns	Multi-turn target position.	INT16	2

The Function Code for COB-ID is 0x300.

The Command Code is 0x34.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Brushed: Open Loop Control (1-2 motors)

The purpose of this command is to control 1-2 brushed motors or solenoids in an open-loop way. The motors/solenoids are independently controlled. Both direct and reverse directions of speed are supported.

The brushed motors/solenoids need to be connected to the controller in the following way:

- Brushed Motor #1 shall be connected to terminals "A" and "B".
- Brushed Motor #2 (if exists) shall be connected to terminals "C" and "B".
- Note that both motors/solenoids share the terminal "B".

	Parameter	Units	Description	Data type	Position in Payload
1	Voltage: Channel #1	V	A commanded output voltage for channel #1	FLOAT16	2
2	Voltage: Channel #2	V	A commanded output voltage for channel #2	FLOAT16	4

The Function Code for COB-ID is 0x300.

The Command Code is 0x90.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Testing: Servo Step Response

This command initiates a load test of "Servo" function of the controller. The load test is typically run to verify that the relevant control laws are configured properly. The testing procedure forces the output shaft of a servo actuator to continuously transit back-and-forth between two discrete Work Zone positions, a positive one and a negative one. Use "Stop" or "Reset" command to terminate the testing procedure.

	Parameter	Units	Description	Data type	Position in Payload
1	Period	sec	This parameter specifies a time period between transitions of the	FLOAT16	2

			servo actuator's position.		
2	Amplitude: Work Zone Counts	counts	This parameter defines an amplitude of servo transitions. For example, if the amplitude is 1000, the servo transits between the positions [-1000] and [1000]. The positions are defined in Work Zone counts.	INT32	4

The Function Code for COB-ID is 0x300.

The Command Code is 0xB8.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Autoconfiguration: Brushless Motor

Use this command when commissioning a new brushless motor. The command launches an auto-configuration procedure that automatically measures various characteristics of a brushless motor, computes optimal parameters for relevant control laws, and updates stored configuration of the controller. The updated configuration is saved to a persistent storage of the controller (Flash).

ATTENTION: The brushless motor makes various moves while the procedure is ongoing, including a rapid acceleration. The beginning and an end of the procedure are indicated by "beep" sounds produced by the motor.

Note that in order to launch the auto-configuration procedure, the user needs to have prior knowledge of "Maximum Continuous Current" and "Poles Number" characteristics of the brushless motor. Those characteristics has to be taken from the motor's datasheet or determined experimentally prior to launching the auto-configuration procedure. Read descriptions of the parameters "Maximum Continuous Current" and "Poles Number" in the "Datasheet" section for details about the parameters.

The auto-configuration procedure does the following:

1. Measures "Phase Resistance (Line-to-Line)" of the brushless motor
2. Measures "Phase Inductance (Line-to-Line)" of the brushless motor
3. Measures "Back-Emf Constant (Ke)" of the brushless motor
4. Measures "Moment of Inertia of Rotor and Payload"
5. Detects if Hall sensors are properly wired to the controller, and updates the "Hall Sensors" configuration parameter in the "Datasheet" section accordingly. If Hall sensors have been detected, the procedure automatically configures the section "Peripheral: Hall Sensors"
6. Writes all the measured parameters to the "Datasheet" section
7. Computes optimal parameters for various control laws, and writes the parameters to "Control Laws" section
8. Save to Flash: all the updated configuration parameters are automatically saved to a persistent storage of the controller

NOTE: The parameter "Moment of Inertia of Rotor and Payload" is automatically measured by the controller during the auto-configuration procedure. However, the procedure assumes that Viscous Damping is not present. If that turns out to be not the case, the auto-configuration procedure overestimates the moment of inertia which might lead to vibrations or noise in the drive whenever the drive is operated under Electronic Speed Control (ESC).

	Parameter	Units	Description	Data type	Position in Payload
1	Maximum Limit on Continuous Current (Line-to-Line)	A	This is a maximum current the motor can handle indefinitely without over-heating. Do not confuse this with a short-term peak current which could be much higher. The "Maximum Limit on Continuous Current (Line-to-Line)" is one of the most critical performance and safety parameters in the "Datasheet" section. This parameter should be found in the motor's datasheet. On one hand, the parameter defines the maximum torque the electric drive can produce. The higher this limit is set, the more electric current is allowed to be driven through the motor by the controller, the more torque the motor produces, the better the dynamics of the electric drive is. On the other hand, driving more current through the motor means generating more heat in the motor's winding. The heat is what burns electric motors. This means that making a mistake and setting this parameter too high might have a fatal consequences for the motor. Setting this parameter too low would mean that the motor is not used to its full capacity in terms of torque. In short, it is important to set this parameter right. Note that if a particular application does not require all the torque the motor can produce, it would be wise to set the limit lower than a nominal value suggested by the manufacturer. This would establish a safety margin at the expense of torque.	FLOAT32	4
2	Poles Number (Rotor Poles)	an even number	The Poles Number parameter should be taken from the motor's datasheet, or it can be determined experimentally. Note that the number of rotor poles is always an EVEN number since magnet poles always come in pairs. The number of poles can be determined by looking inside the motor. Just count the number of coils in the motor, divide it by 3, and then multiply by 2. For example, if a motor has 21 stator coils, this means that Poles Number = (21 / 3) * 2 = 14. If one cannot peek inside the motor, there is a simple procedure that experimentally determines the Poles Number.	UINT16	2

The Function Code for COB-ID is 0x400.

The Command Code is 0xD0.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - Autoconfiguration: Brushed Motor

Use this command when commissioning a new brushed motor. The command launches an auto-configuration procedure that automatically measures various characteristics of a brushed motor, computes optimal parameters for relevant control laws, and updates stored configuration of the controller. The updated configuration is saved to a persistent storage of the controller (Flash).

	Parameter	Units	Description	Data type	Position in Payload
1	Maximum Limit on Continuous Current (Line-to-Line)	A	The "Maximum Limit on Continuous Current" is one of the most critical performance and safety parameters in the "Datasheet" section. This parameter should be found in the motor's datasheet. On one hand, the parameter defines the maximum torque the electric drive can produce. The higher this limit is set, the more electric current is allowed to be driven through the motor by the controller, the more torque the motor produces, the better the dynamics of the electric drive is. On the other hand, driving more current through the motor means generating more heat in the motor's winding. The heat is what burns electric motors. This means that making a mistake and setting this parameter too high might have a fatal consequences for the motor. Setting this parameter too low would mean that the motor is not used to its full capacity in terms of torque. In short, it is important to set this parameter right. Note that if a particular application does not require all the torque the motor can produce, it would be wise to set the limit lower than a nominal value suggested by the manufacturer. This would establish a safety margin at the expense of torque.	FLOAT32	4

The Function Code for COB-ID is 0x400.

The Command Code is 0xD4.

This command should be sent CONTINUOUSLY at regular intervals to avoid heartbeat timeout on the device side.

Command - GPIO: Generic Output

This command sets output (0 or 1) on a dedicated GPIO output pin.

	Parameter	Units	Description	Data type	Position in Payload
1	Output	0 or 1	This parameter provides a value (0 or 1) to be set as an output on a dedicated GPIO output pin.	UINT16	2

The Function Code for COB-ID is 0x400.

The Command Code is 0xA0.

It is generally not required to continuously send this command to the device.

Telemetry Mappings (TPDO)

Telemetry Message with COB-ID 0x180

	Name	Units	Position in Payload (byte #)	Data type	Index	Subindex
1	Fault Bits	bits	0	UINT16	0x4000	0x03
2	Voltage: DC bus (Udc)	V	2	FLOAT16	0x5001	0x04
3	Speed (Electrical Frequency)	Hz	4	FLOAT32	0x4000	0x10

Telemetry Message with COB-ID 0x280

	Name	Units	Position in Payload (byte #)	Data type	Index	Subindex
1	Operation Mode	-	0	UINT16	0x4000	0x02
2	Commutation Mode	-	2	UINT16	0x4000	0x15
3	Work Zone: Count	counts	4	INT32	0x4000	0x19

Telemetry Message with COB-ID 0x380

	Name	Units	Position in Payload (byte #)	Data type	Index	Subindex
1	Current: Phase A (Ia)	A	0	FLOAT32	0x5001	0x05
2	Current: Phase B (Ib)	A	4	FLOAT32	0x5001	0x06