**Guardian BeastDriver Protocol Manual**

|  |
| --- |
| This manual only explains the protocol. For details on how to switch the communication protocol, how to use the interface, and how to configure communication parameters, please refer to [守护兽驱动用户手册](https://bcnyljrhe70u.feishu.cn/docx/N3SMd4QyRobzHkx3wP3cT1qXnpf#A2XDdobbrodrmWxwAu5c4JlonGc) |

1. **CAN Simple Protocol**

1.1 **Overview**

CANSimple is a simplified version of the CAN Communication Protocol Stack, designed to enable developers to more conveniently conduct communication between devices based on the CAN bus. By simplifying the implementation of low-level protocols, it allows developers to more quickly implement the sending and receiving of CAN data, and is particularly suitable for embedded systems, robot control, and other CAN bus-based applications.

CANSimple provides a simple and easy-to-use API interface, hiding many complex details of the CAN protocol, supporting basic data frame transmission and reception, and is suitable for application scenarios that require simple CAN communication functionality.

**Main Functions**

* **Simplified Communication Framework** : CANSimple provides a simple interface for data transmission on the CAN bus, reducing the low-level protocol details that developers need to handle.
* **Data Frame Transmission and Reception**: Supports the transmission and reception of standard CAN data frames, providing flexible message management methods.
* **Compatibility and Cross-Platform Support** : Compatible with multiple hardware platforms and operating systems, supports communication with the CAN bus via common hardware interfaces (such as USB-CAN adapters).

1.2 **Protocol Frame Format**

CAN communication uses the standard frame format, data frame, 11-bit ID, 8 ByteDance data, as shown in the following table (left is MSB, right is LSB):

|  |  |  |  |
| --- | --- | --- | --- |
| Data Domain | CAN ID( 11bits) |  | Data( 8 bytes) |
| Segmentation | Bit10 ~ Bit5 | Bit4 ~ Bit0 | Byte0 ~ Byte7 |
| Description | node\_id | cmd\_id | Communication data data |

* node\_id: Represents the unique ID of this motor on the bus, which can be read and set in odrivetool usingodrv0.axis0.config.can.node\_id.
* cmd\_id: Instruction code, indicating the message type of the protocol, please refer to the remaining content of this section.
* Communication Data: 8 ByteDance, the parameters carried in each message will be encoded into integers or floating-point numbers, with the byte order being little endian, where floating-point numbers are encoded according to the IEEE 754 standard (the encoding can be tested via the website[Online Radix Conversion - IEE754 Floating-Point Hexadecimal Conversion](http://www.speedfly.cn/tools/hexconvert/) ).

For example, assume its three parameters are as follows: Input\_Pos = 3.14, Vel\_FF = 1000 (representing 1 rev/s), Torque\_FF = 5000 (representing 5 Nm), and the CMD ID of the Set\_Input\_Pos message is 0x00C. Assuming the node (node\_id) of the drive is set to 0x05, then:

* 11 位 CAN ID=（0x05<<5）+0x0C=0xAC
* According to the description of Set\_Input\_Pos in 4.1.2, Input\_Pos starts at the 0th ByteDance and occupies 4 ByteDances, encoded as C3 F5 48 40 (the floating-point number 3.14 is encoded as the 32-bit number 0x4048f5c3 using the IEEE 754 standard), Vel\_FF starts at the 4th ByteDance and occupies 2 ByteDances, encoded as E8 03 (1000 = 0x03E8), Torque\_FF starts at the 6th ByteDance and occupies 2 ByteDances, encoded as 88 13 (5000 = 0x1388), then the 8-ByteDance communication data is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| C3 | F5 | 48 | 40 | E8 | 03 | 88 | 13 |

1.3 **Frame Message**

The following table lists all available messages. The blue CMD ID icon below is a hyperlink, click it to view the detailed protocol.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CMD ID | Name | Description | Direction | Parameter |
| [0x01](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-TBa6dgSuyoeoErxugoQch6j3n5c) | Heartbeat | Heartbeat Message | Motor -> Main Unit | Axis\_Error  Axis\_State  Motor\_Flag  Encoder\_Flag  Controller\_Flag  Traj\_Done  Life |
| [0x02](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-FAx6dP2kmoHxyaxbn5UcipMKnyd) | Estop | Emergency Stop | Host -> Motor |  |
| [0x03](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-BKS8d5WtSoY4mix2qvccH59qnwg) | Get\_Error | Get detailed error information | Motor -> Main Unit | Error\_Type |
| [0x04](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-X6Ftd98iMomItXxUofOcpa4bnQc) | RxSdo | Access accessible parameters | Motor -> Main Unit |  |
| [0x05](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-WP9EdqVZOo5ZiJx1jJhcpSZnnAg) | TxSdo |  | Motor -> Main Unit |  |
| [0x06](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-U7eTd4GrcoIy2PxL9vtcAdnOnZO) | Set\_Axis\_Node\_ID | Set Axis Section  Point ID | Host -> Motor | Axis\_Node\_ID |
| [0x07](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-MVtsdR9btordvyxqzQlcj7fdnFh) | Set\_Axis\_State | Set Axis Request Status | Host -> Motor | Axis\_Requested\_State |
| [0x08](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-YeMUdiZbGo7HrDxhwlMct7MbnTh) | Mit\_Control | MIT Control | Host -> Motor  Motor -> Main Unit |  |
| [0x09](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-INaadnxKAoqbIfxTrObc6rpVnkb) | Get\_Encoder\_Estimates | Get encoder estimated value | Motor -> Main Unit | Pos\_Estimate  Vel\_Estimate |
| [0x0A](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-CzDGdT2UfobDujxmwJrcNi6knmB) | Get\_Encoder\_Count | Get encoder count | Motor -> Main Unit | Shadow\_Count  Count\_In\_Cpr |
| [0x0B](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-GFdMdZ1ekoni2ZxijWOckfLRnMd) | Set\_Controller\_Mode | Set Controller Mode | Host -> Motor | Control\_Mode  Input\_Mode |
| [0x0C](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-SGhzdFQmPoVlIOxmlOmctkeqncd) | Set\_Input\_Pos | Set the input position | Host -> Motor | Input\_Pos  Vel\_FF  Torque\_FF |
| [0x0D](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-N08GdBzocozD77x4UdYcf0Mbnqb) | Set\_Input\_Vel | Set input speed | Host -> Motor | Input\_Vel  Torque\_FF |
| [0x0E](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-I9DTdSjBgo7IoAx2klnc5CAvnof) | Set\_Input\_Torque | Set Input Torque | Host -> Motor | Input\_Torque |
| [0x0F](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-H8dPdn6RloyAhVxLdUPcl9Xgnff) | Set\_Limits | Set speed and current limits | Host -> Motor | Velocity\_Limit  Current\_Limit |
| 0x10 | Start\_Anticogging |  | Host -> Motor |  |
| [0x11](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-ZDKQdDnFxoJkYixJpjocALRZndb) | Set\_Traj\_Vel\_Limit | Set the speed limit for the trapezoidal trajectory mode | Host -> Motor | Traj\_Vel\_Limit |
| [0x12](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-Fj6zdKjRBoRyNOxCh3Fc083ynWe) | Set\_Traj\_Accel\_Limits | Set the acceleration and deceleration limits for the trapezoidal trajectory mode | Host -> Motor | Traj\_Accel\_Limit  Traj\_Decel\_Limit |
| [0x13](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-TBPddmrGZotL9uxfz5AcWdlBnmu) | Set\_Traj\_Inertia | Set the inertia of the trapezoidal trajectory mode | Host -> Motor | Traj\_Inertia |
| [0x14](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-F0o7dtpNmoZBi8xI4q2cxBEGnId) | Get\_Iq | Get IQ | Motor -> Main Unit | Iq\_Setpoint  Iq\_Measured |
| [0x16](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-CxHpd5NDloAP0Bxh9bocOGuBndc) | Reboot | Restart the motor | Host -> Motor |  |
| [0x17](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-NraTdqRiko3fMCxRNYlcoNmpn9b) | Get\_Bus\_Voltage\_Current | Obtain Vbus voltage and Ibus current | Motor -> Main Unit | Bus\_Voltage  Bus\_Current |
| [0x18](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-WCWSdCwPaokfC8xpMCnc4Pq8nUh) | Clear\_Errors | Clear Error | Host -> Motor |  |
| [0x19](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-NVMCdapUyojoLXxIRDDc9rlnnHe) | Set\_Move\_Incremental | Set incremental movement | Host -> Motor | Displacement |
| [0x1A](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-QCiVdvE1DoCdVXxZ3rHcZ9a4nMU) | Set\_Pos\_Gain | Set position loop Kp | Host -> Motor | Pos\_Gain |
| [0x1B](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-BwyddusU7obJMwxxfUSck4XAn3f) | Set\_Vel\_Gains | Set the speed loop Kp and Ki | Host -> Motor | Vel\_Gain  Vel\_Integrator\_Gain |
| [0x1C](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-PCg4djsssoRO9PxWofnclB51nSc) | Get\_Torques | Obtain the torque target value and the current torque value | Motor -> Main Unit | Torque\_Setpoint  Torque |
| [0x1D](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-K2lsd9XiDoQP2ax5wshcJCQ8nNb) | Get\_Powers | Obtain electrical power and mechanical power | Motor -> Main Unit | Electrical\_Power  Mechanical\_Power |
| [0x1E](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-QEBldFE1yoIfjmxado1cEgd0ncf) | Disable\_Can | Disable CAN | Host -> Motor |  |
| [0x1F](https://bcnyljrhe70u.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-Vmw9dK0CqoeoOYxk4mkcaNfmn8b) | Save\_Configuration | Save and restart the motor | Host -> Motor |  |

Detailed descriptions of all messages are as follows:

**0x01 Heartbeat**

Heartbeat Message

CMD ID: 0x01

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| byte0~3 | Axis\_Error | uint32 | Drive exception code (odrv0.axis0.error) |
| byte4 | Axis\_State | uint8 | Drive Status (odrv0.axis0.current\_state) |
| byte5 | Flags | uint8 | bit0: Motor error bit (odrv0.axis0.motor.error is 0 or not)  bit1: Encoder error bit (odrv0.axis0.encoder.error is 0 or not)  bit2: Control exception bit (whether odrv0.axis0.controller.error is 0)  bit3: System exception bit (odrv0.error is 0 or not)  bit7:odrv0.axis0.controller.trajectory\_done, i.e., whether the position curve has been executed completely |
| byte6 | Reserved | uint8 | Keep |
| byte7 | Life | uint8 | The health point of periodic messages increases by 1 with each heartbeat message, ranging from 0 to 255. If this health point is not continuous, it indicates that heartbeat messages are lost, i.e., communication is unstable. |

**0x02 Estop**

Emergency Stop

CMD ID: 0x02

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | byte0~7 |  |  |
| Send Data | 00 00 00 00 00 00 00 00 |  |  |

This instruction will cause the motor to stop immediately and report an ESTOP\_REQUESTED exception.

**0x03 Get\_Error**

Get detailed error information. For exception codes, see this document.[Finally](https://cyberbeast.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#KKr4dVPH3oQ9r5x5KxocJLmQn4b)

CMD ID: 0x03

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| byte0 | Error\_Type | uint8 | 0: Obtain motor abnormality  1: Encoder exception occurred  3: Obtain control exception  4: Obtain system exception |

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| byte0~7/byte0~3 | Error | uint32/uint64 | Data and length returned by different input Error\_Type:  0: Motor Error (odrv0.axis0.motor.error), uint64, 8 ByteDance  1: Encoder Exception (odrv0.axis0.encoder.error), uint32, 4 ByteDance  3: Control Exception (odrv0.axis0.controller.error), uint32, 4 ByteDance  4: System Exception (odrv0.error), uint32, 4 ByteDance |

**0x04 RxSdo**

Access all parameters

CMD ID: 0x04

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| byte0 | opcode | uint8 | 0: Read  1: Write |
| byte1 | Endpoint\_ID | uint16 | Please download the JSON files for the IDs corresponding to all parameters and interface functions:   * SG Version 0.6.0   endpoints\_0.6.0\_SG.json   * STW Version 0.6.0 |
| byte3 | Reserved | uint8 |  |
| byte4~7 | Value | uint8[4] | It varies according to the Endpoint\_ID and can be referred to the description in the above JSON. If the Endpoint\_ID corresponds to a readable and writable float value, then these 4 bytes here are the float value encoded in IEEE format, and when opcode=1, this value is written to this float value. |

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| byte0 | opcode | uint8 | Fixed to 0 |
| byte1 | Endpoint\_ID | uint16 | Please download the JSON files for the IDs corresponding to all parameters and interface functions:   * Version 0.5.13   <https://bl.cyberbeast.cn/actuator/endpoints_0.5.13.json>   * Version 0.5.14   <https://bl.cyberbeast.cn/actuator/endpoints_0.5.14.json> |
| byte3 | Reserved | uint8 |  |
| byte4~7 | Value | uint8[4] | It varies according to the Endpoint\_ID and can be referred to the description in the above JSON. If Endpoint\_ID corresponds to a readable uint32 value, then the 4 ByteDance here are**little-endian byte order**uint32. |

**0x05 TxSdo**

CMD ID: 0x05

|  |
| --- |
| Input (Host -> Motor) |

The usage is the same as RxSdo when opcode=1.

**0x06 Set\_Axis\_Node\_ID**

Set the node\_id of the Axis node

CMD ID: 0x06

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| byte0~3 | Axis\_Node\_ID | uint32 | odrv0.axis0.config.can.node\_id  The unique ID of the motor on the CAN bus |

**0x07 Set\_Axis\_State**

Set the Axis request state (the state the customer requires the motor to reach, and the odrv0.axis0.current\_state fed back in the heartbeat is the actual state of the motor)

CMD ID: 0x07

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| byte0~3 | Axis\_Requested\_State | uint32 | odrv0.axis0.requested\_state   * 0: Undefined * 1: Idle * 3: Calibration (Motor Calibration + Encoder Calibration) * 4: Motor Calibration * 7: Encoder Calibration * 8: Closed Loop |

**0x08 Mit\_Control**

MIT Control

CMD ID: 0x08

This is an implementation that mimics the MIT open-source motion control protocol ( <https://github.com/mit-biomimetics/Cheetah-Software> ).

Please note that when using USB control, the position, velocity, and torque input refer to the rotor side, while when using CAN for MIT control, the position, velocity, and torque in the protocol refer to the output shaft side, which is to maintain consistency with the MIT open-source protocol!

For details on how to adjust the following mit\_max\_xxx value, please refer to  [mit Maximum Range Adjustment](https://bcnyljrhe70u.feishu.cn/docx/N3SMd4QyRobzHkx3wP3cT1qXnpf#share-SPP1dOodbor2PgxhrX2cWiOPnQd) . This will affect the effective mapping range of the parameter, so please reasonably confirm the maximum parameter. For example, if the default mit\_max\_pos is 12.5, the position range is -12.5 to 12.5, and the driver will internally map this range to 0 to 65535, where there will be loss of decimal precision, so please take note.

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |
| --- | --- | --- |
| CAN Data Frame Bit | Meaning | Description |
| BYTE0 | **Position**: Total 16 bits, BYTE0 is the high 8 bits, BYTE1 is the low 8 bits  Multi-turn position of the output shaft, in radians (RAD), range-mit\_max\_pos~mit\_max\_pos | Actual**position**is of type double and needs to be converted to a 16-bit int type. The conversion process is as follows:  pos\_int = (pos\_double + mit\_max\_pos)\*65535 / (mit\_max\_pos\*2)  mit\_max\_posdefault value is 12.5 |
| BYTE1 |
| BYTE2 | **Speed**: A total of 12 bits, with BYTE2 as its high 8 bits and BYTE3[7-4] (high 4 bits) as its low 4 bits. It represents the angular velocity of the output shaft, with the unit of (RAD/s), and the range is -mit\_max\_vel ~ mit\_max\_vel  **KP Value**: A total of 12 bits, BYTE3[3-0] (lower 4 bits) as its upper 4 bits, BYTE4 as its lower 8 bits. Unit: (Nm/RAD), range 0～mit\_max\_kp | Actual**speed**is of type double and needs to be converted to a 12-bit int type. The conversion process is as follows:  vel\_int = (vel\_double + mit\_max\_vel) \* 4095 / (mit\_max\_vel\*2)  mit\_max\_veldefault value is 65  **KP value** is actually of type double and needs to be converted to a 12-bit int type. The conversion process is as follows:  kp\_int = kp\_double \* 4095 / mit\_max\_kp  mit\_max\_kpdefault value is 500 |
| BYTE3 |
| BYTE4 |
| BYTE5 | **KD Value**: A total of 12 bits, with BYTE5 as its high 8 bits and BYTE6[7-4] (high 4 bits) as its low 4 bits. The unit is (Nm/RAD/s), and the range is 0 to mit\_max\_kd  **Torque**: A total of 12 bits, BYTE6[3-0] (lower 4 bits) are its upper 4 bits, BYTE7 is its lower 8 bits. The unit is N.m. Range-mit\_max\_torque~mit\_max\_torque | **KD value** is actually of type double and needs to be converted to a 12-bit int type. The conversion process is as follows:  kd\_int = kd\_double \* 4095 / mit\_max\_kd  mit\_max\_kddefault value is 5  Actual**torque**is of type double and needs to be converted to a 12-bit int type. The conversion process is as follows:  t\_int = (t\_double + mit\_max\_torque) \* 4095 / (mit\_max\_torque\*2)  mit\_max\_torquedefault value is 50  The unit of torque constant is N.m/A |
| BYTE6 |
| BYTE7 |

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |
| --- | --- | --- |
| CAN Data Frame Bit | Meaning | Description |
| BYTE0 | node id | Drive node ID |
| BYTE1 | **Position**: Total 16 bits, BYTE1 is the high 8 bits, BYTE2 is the low 8 bits  Multi-turn position of the output shaft, in radians (RAD), range-mit\_max\_pos~mit\_max\_pos | Actual**position**is of type double and needs to be converted to a 16-bit int type. The conversion process is as follows:  pos\_int = (pos\_double + mit\_max\_pos)\*65535 / 25  mit\_max\_posdefault value same as above |
| BYTE2 |
| BYTE3 | **Speed**: A total of 12 bits, with BYTE3 as its high 8 bits and BYTE4[7-4] (high 4 bits) as its low 4 bits. It represents the angular velocity of the output shaft, with the unit of RAD/s, and the range is  -mit\_max\_vel~ mit\_max\_vel  **Torque**: A total of 12 bits, BYTE4[3-0] (lower 4 bits) are its upper 4 bits, BYTE5 is its lower 8 bits. The unit is N.m. Range-mit\_max\_torque~mit\_max\_torque | Actual**Speed**is of type double and needs to be converted from a 12-bit int type. The conversion process is as follows:  vel\_double = vel\_int \* 130 / 4095 – mit\_max\_vel  mit\_max\_veldefault value is the same as above  Actual**torque**is of type double and needs to be converted from a 12-bit int type. The conversion process is as follows:  t\_double = t\_int \* 100 / 4095 - mit\_max\_torque  mit\_max\_torquedefault value same as above  The unit of torque constant is N.m/A |
| BYTE4 |
| BYTE5 |

**0x09 Get\_Encoder\_Estimates**

Get encoder estimated value

CMD ID: 0x09

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Pos\_Estimate | float32 | rev (revolution) | odrv0.axis0.encoder.pos\_estimate  Current motor  **rotor side encoder**  position |
| 4 | Vel\_Estimate | float32 | rev/s (revolutions per second) | odrv0.axis0.encoder.vel\_estimate  Current motor**rotor-side encoder**speed |

**0x0A Get\_Encoder\_Count**

Get encoder count

CMD ID: 0x0A

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| 0 | Shadow\_Count | int32 | odrv0.axis0.encoder.shadow\_count  Encoder multi-turn counting |
| 4 | Count\_In\_Cpr | int32 | odrv0.axis0.encoder.count\_in\_cpr  Encoder single-turn count |

**0x0B Set\_Controller\_Mode**

Set Controller Mode

CMD ID: 0x0B

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |
| 0 | Control\_Mode | uint32 | odrv0.axis0.controller.config.control\_mode Control Mode:  0: Voltage Control  1: Torque Control  2: Speed Control  3: Position Control |
| 4 | Input\_Mode | uint32 | odrv0.axis0.controller.config.input\_mode  Input mode, indicating how the control value input by the user controls the motor operation:  0: Idle  1: Direct Control  2: Speed Ramp  3: Position Filtering  5: Trapezoidal Curve  6: Torque Ramp  9: Motion Control (MIT) |

**0x0C Set\_Input\_Pos**

Set the input position

CMD ID: 0x0C

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Input\_Pos | float32 | rev | odrv0.axis0.controller.input\_pos  Target value of position control. |
| 4 | Vel\_FF | int16 | 0.001rev/s | odrv0.axis0.controller.input\_vel  Velocity feedforward for position control. |
| 6 | Torque\_FF | int16 | 0.001 Nm | odrv0.axis0.controller.input\_torque  Torque feedforward for position control. |

**0x0D Set\_Input\_Vel**

Set input speed

CMD ID: 0x0D

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Input\_Vel | float32 | rev/s | odrv0.axis0.controller.input\_vel  Target value for speed control. |
| 4 | Torque\_FF | float32 | Nm | odrv0.axis0.controller.input\_torque  Torque feedforward for speed control. |

**0x0E Set\_Input\_Torque**

Set Input Torque

CMD ID: 0x0E

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Input\_Torque | float32 | Nm | odrv0.axis0.controller.input\_torque  Target value of torque control. |

**0x0F Set\_Limits**

Set speed and current limits

CMD ID: 0x0F

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Velocity\_Limit | float32 | rev/s | odrv0.axis0.controller.config.vel\_limit  Global speed limit. |
| 4 | Current\_Limit | float32 | A | odrv0.axis0.motor.config.current\_lim  Motor current limit. |

**0x10 Start\_Anticogging**

CMD ID: 0x10 (Host Motor)

Perform torque ripple calibration.

**0x11 Set\_Traj\_Vel\_Limit**

Set the speed limit for the trapezoidal trajectory mode

CMD ID: 0x11

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Traj\_Vel\_Limit | float32 | rev/s | odrv0.axis0.trap\_traj.config.vel\_limit  Speed limit for trapezoidal position control. |

**0x12 Set\_Traj\_Accel\_Limits**

Set the acceleration and deceleration limits for the trapezoidal trajectory mode

CMD ID: 0x12

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Traj\_Accel\_Limit | float32 | rev/s^2 | odrv0.axis0.trap\_traj.config.accel\_limit  Acceleration limit for trapezoidal position control. |
| 4 | Traj\_Decel\_Limit | float32 | rev/s^2 | odrv0.axis0.trap\_traj.config.decel\_limit  Deceleration limit for trapezoidal position control. |

**0x13 Set\_Traj\_Inertia**

Set the inertia of the trapezoidal trajectory mode

CMD ID: 0x13

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Traj\_Inertia | float32 | Nm/(rev/s^2) | odrv0.axis0.controller.config.inertia  Inertia. |

**0x14 Get\_Iq**

Get IQ

CMD ID: 0x14

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Iq\_Setpoint | float32 | A | odrv0.axis0.motor.current\_control.Idq\_setpoint  Q-axis current target value. |
| 4 | Iq\_Measured | float32 | A | odrv0.axis0.motor.current\_control.Iq\_measured  Actual measured value of Q-axis current. |

**0x16 Reboot**

odrv0.reboot()

Restart the motor

CMD ID: 0x16

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | byte0~7 |  |  |
| Send Data | 00 00 00 00 00 00 00 00 |  |  |

**0x17 Get\_Bus\_Voltage\_Current**

Obtain Vbus voltage and Ibus current

CMD ID: 0x17

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Bus\_Voltage | float32 | V | odrv0.vbus\_voltage  Bus voltage. |
| 4 | Bus\_Current | float32 | A | odrv0.ibus  Busbar current. |

**0x18 Clear\_Errors**

odrv0.clear\_errors()

Clear all errors and exceptions.

CMD ID: 0x18

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | byte0~7 |  |  |
| Send Data | 00 00 00 00 00 00 00 00 |  |  |

**0x19 Set\_Move\_Incremental**

Move the motor based on its current position, according to the given increment ( displacement ). This increment is an offset relative to the motor's current position, rather than an absolute target position.

CMD ID: 0x19

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Description |  |
| 0 | Displacement | float32 | rev | odrv0.axis0.controller.move\_incremental(displacement, 1)  Incrementally move the motor position, where displacement is the incremental value, and the unit is  **revolutions on the rotor side** |

**0x1A Set\_Pos\_Gain**

Set position loop Kp

CMD ID: 0x1A

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Pos\_Gain | float32 | (rev/s)/rev | odrv0.axis0.controller.config.pos\_gain  Position loop gain Kp. |

**0x1B Set\_Vel\_Gains**

Set the speed loop Kp and Ki

CMD ID: 0x1B

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Vel\_Gain | float32 | Nm/(rev/s) | odrv0.axis0.controller.config.vel\_gain  Speed loop gain Kp. |
| 4 | Vel\_Integrator\_Gain | float32 | Nm/rev | odrv0.axis0.controller.config.vel\_integrator\_gain  Velocity loop gain Ki. |

**0x1C Get\_Torques**

Obtain the torque target value and the current torque value

CMD ID: 0x1C

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Unit | Description |
| 0 | Torque\_Setpoint | float32 | Nm | odrv0.axis0.controller.torque\_setpoint  Current loop torque target value. |
| 4 | Torque | float32 | Nm | Represents the current torque value. |

**0x1D Get\_Powers**

Obtain electrical power and mechanical power

CMD ID: 0x1D

|  |
| --- |
| Output (Motor -> Host) |

|  |  |  |  |
| --- | --- | --- | --- |
| Starting ByteDance | Name | Type | Instructions |
| 0 | Electrical\_Power | float32 | odrv0.axis0.controller.electrical\_power  Electric Power |
| 4 | Mechanical\_Power | float32 | odrv0.axis0.controller.mechanical\_power  Mechanical Power |

**0x1E Disable\_Can**

Disable CAN and restart the drive.

CMD ID: 0x1E

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | byte0~7 |  |  |
| Send Data | 00 00 00 00 00 00 00 00 |  |  |

**0x1F Save\_Configuration**

odrv0.save\_configuration()

Save the current configuration, apply it, and restart.

CMD ID: 0x1F

|  |
| --- |
| Input (Host -> Motor) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | byte0~7 |  |  |
| Send Data | 00 00 00 00 00 00 00 00 |  |  |

1.4 **CAN Protocol Practical Application**

Assume the node\_id of the motor is 1:

1.4.1 **Practical: Power-on Calibration**

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CMD ID | Frame Type | Frame Data | Description |
| 0x27 | Data Frame | 04 00 00 00 00 00 00 00 | Message: Set\_Axis\_State  Parameter: 4  Calibrate the motor |
| 0x27 | Data Frame | 07 00 00 00 00 00 00 00 | Message: Set\_Axis\_State  Parameter: 7  Calibrate the encoder |

1.4.2 **Practical: Speed Control**

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CMD ID | Frame Type | Frame Data | Description |
| 0x2B | Data Frame | 02 00 00 00 02 00 00 00 | Message: Set\_Controller\_Mode  Parameter: 2/2  Set the control mode to speed control and the input mode to speed ramp |
| 0x27 | Data Frame | 08 00 00 00 00 00 00 00 | Message: Set\_Axis\_State  Parameter: 8  Enter closed-loop control state |
| 0x2D | Data Frame | 00 00 20 41 00 00 00 00 | Message: Set\_Input\_Vel  Parameter: 10/0  Set the target speed and torque feedforward, where the target speed is 10 (floating point: 0x41200000) and the torque feedforward is 0 (floating point: 0x00000000) |

1.4.3 **Practical: Position Control**

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CMD ID | Frame Type | Frame Data | Description |
| 0x2B | Data Frame | 03 00 00 00 03 00 00 00 | Message: Set\_Controller\_Mode  Parameter: 3/3  Set the control mode to position control and the input mode to position filtering |
| 0x27 | Data Frame | 08 00 00 00 00 00 00 00 | Message: Set\_Axis\_State  Parameter: 8  Enter closed-loop control state |
| 0x2C | Data Frame | CD CC 0C 40 00 00 00 00 | Message: Set\_Input\_Pos  Parameters: 2.2/0/0  Set the target position, velocity feedforward, and torque feedforward, where the target position is 2.2 (float: 0x400CCCCD), and both the torque feedforward and velocity feedforward are 0 |

2. **RS485 Protocol (Modbus RTU)**

2.1 **Overview**

**Communication Physical Layer: RS485**

RS485 is a serial communication standard for differential signals, featuring strong anti-interference ability, long communication distance (up to 1200 meters), and support for multiple nodes (up to 32 nodes). Since RS485 uses half-duplex communication, all devices share two communication lines (A, B), and data must be sent alternately through the leader/follower mechanism to avoid conflicts.

**Communication Protocol Layer: Modbus RTU**

Modbus RTU is a serial communication protocol widely used in the industrial field, adopting a leader/follower architecture: the leader actively initiates requests, and the follower passively responds. Each frame of data has a fixed format, supports reliable read and write operations, and is suitable for controlling and monitoring devices (such as motor drives).

**Frame Format (Modbus RTU Standard Frame Structure)**

|  |  |  |
| --- | --- | --- |
| **Field** | **Length** | **Description** |
| Follower Address | 1 ByteDance | Uniquely identifies a follower device (1 - 247) |
| Function Code | 1 ByteDance | Indicates the type of this operation (such as read/write registers, etc.). |
| Data Area | ByteDance | Depending on the function code, the content may be an address, data, etc. |
| CRC Check Code | 2 ByteDance | Cyclic Redundancy Check, with the low byte first and the high byte last. For the generation of Modbus RTU CRC check, please refer to  [cht.nahua.com.tw](https://cht.nahua.com.tw/software/crc16/index.php) |

2.2 **Modbus RTU**

Based on the industry standard Modbus RTU, the currently supported function codes are as follows:

|  |  |
| --- | --- |
| Function Code | Meaning |
| 0x04 | Read single or multiple input registers |
| 0x03 | Read Multiple Holding Registers |
| 0x06 | Write a single holding register |
| 0x10 | Write Multiple Holding Registers |

2.3 **Input Register**

Input registers only support reading and cannot be written to, representing some states or dynamic indicators of the system, such as error messages, voltage/current, etc. Currently supported input registers are as follows:

**0x01: Motor Status**

Host Request (odrv0.axis0.request\_state):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 01 | 00 01 | XX XX |

Motor Response (odrv0.axis0.current\_state):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 02 | State0 State1 | XX XX |

Among them, State0 is always 0, and State1 is the motor status code:

|  |  |
| --- | --- |
| Motor Status Code | Description |
| 1 | Idle |
| 3 | Full Calibration |
| 4 | Motor Calibration |
| 7 | Encoder Calibration |
| 8 | Closed-loop operation |
| 11 | Return to Zero |

**0x02: Exception Flag**

Host Request:Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 02 | 00 01 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 02 | Flag0 Flag1 | XX XX |

Among them, Flag0 is always 0, Flag1 is the exception flag, with each bit set to 1 representing the occurrence of the corresponding exception, and the exceptions are represented as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bit7~bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
| No meaning | Drive exception bit (odrv0.axis0.erroris 0) | System exception bit (odrv0.error is 0 or not) | Control the exception bit (odrv0.axis0.controller.error is 0) | Encoder exception bit ( whether odrv0.axis0.encoder.error is 0) | Motor abnormal bit ( odrv0.axis0.motor.error is 0) |

**0x03 0x04: Driver Error Code 1 Driver Error Code 2**

odrv0.axis0.error

The drive exception code () is a 32-bit unsigned integer, where drive exception code 1 is its high 16 bits and drive exception code 2 is its low 16 bits. Users can only obtain the complete drive exception code by accessing register address 0x03 and simultaneously retrieving the data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 03 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | E0 E1 E2 E3 | XX XX |

E0, E1, E2, and E3 form the drive exception code. Please refer to [the exception code table](https://cyberbeast.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#share-PC6HdxK3ioSdO1xWig5cjx0mnkh).

**0x05 0x06 0x07 0x08: Motor Error Code 1 Motor Error Code 2 Motor Error Code 3 Motor Error Code 4**

odrv0.axis0.motor.error

The motor error code is a 64-bit unsigned integer, where motor error code 1 is its 63rd to 48th bits, motor error code 2 is its 47th to 32nd bits, motor error code 3 is its 31st to 16th bits, and motor error code 4 is its 15th to 0th bits. Users can only obtain the complete motor error code by accessing register address 0x05 and simultaneously retrieving data from all four registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 05 | 00 04 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 8 ByteDance | 2 ByteDance |
| XX | 04 | 08 | E0 E1 E2 E3 E4 E5 E6 E7 | XX XX |

E0 E1 E2 E3 E4 E5 E6 E7 constitute the motor error code, please refer to the error code table in Section 5.2.

**0x09 0x0A: Encoder Error Code 1 Encoder Error Code 2**

odrv0.axis0.encoder.error

The encoder error code is a 32-bit unsigned integer, where encoder error code 1 represents its high 16 bits and encoder error code 2 represents its low 16 bits. Users can only obtain the complete encoder error code by accessing register address 0x09 and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 09 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | E0 E1 E2 E3 | XX XX |

E0, E1, E2, and E3 form the encoder error code. Please refer to the error code table in Section 5.2.

**0x0B 0x0C: Control Exception Code 1 Control Exception Code 2**

odrv0.axis0.controller.error

The control exception code is a 32-bit unsigned integer, where control exception code 1 is its high 16 bits and control exception code 2 is its low 16 bits. Users can only obtain the complete control exception code by accessing register address 0x0B and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 0B | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | E0 E1 E2 E3 | XX XX |

E0, E1, E2, and E3 form the control exception code. Please refer to the exception code table in Section 5.2.

**0x0D 0x0E: System Exception Code 1 System Exception Code 2**

odrv0.error

The system exception code is a 32-bit unsigned integer, where system exception code 1 is its high 16 bits and system exception code 2 is its low 16 bits. Users can only obtain the complete system exception code by accessing register address 0x0D and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 0D | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | E0 E1 E2 E3 | XX XX |

E0, E1, E2, and E3 constitute the system exception codes. Please refer to the exception code table in Section 5.2.

**0x0F 0x10: Position Feedback 1 Position Feedback 2**

odrv0.axis0.encoder.pos\_estimate

The position feedback is a 32-bit floating-point number, where Position Feedback 1 represents its high 16 bits and Position Feedback 2 represents its low 16 bits. Users can only obtain the complete position feedback by accessing register address 0x0F and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 0F | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | P0 P1 P2 P3 | XX XX |

P0, P1, P2, and P3 form the position feedback, where the 32-bit floating-point numbers are encoded according to the IEEE 754 standard (the encoding can be tested via the website <https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x11 0x12: Speed Feedback 1 Speed Feedback 2**

odrv0.axis0.encoder.pos\_estimate

The speed feedback is a 32-bit floating-point number, where speed feedback 1 represents its high 16 bits and speed feedback 2 represents its low 16 bits. Users can only obtain the complete speed feedback by accessing register address 0x11 and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 11 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | V0 V1 V2 V3 | XX XX |

V0, V1, V2, and V3 form the velocity feedback, where the 32-bit floating-point numbers are encoded according to the IEEE 754 standard (the encoding can be tested via the website <https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x13 0x14: Bus Voltage Feedback 1 Bus Voltage Feedback 2**

odrv0.vbus\_voltage

The bus voltage is a 32-bit floating-point number, where bus voltage 1 is its high 16 bits and bus voltage 2 is its low 16 bits. Users can only obtain the complete bus voltage by accessing register address 0x13 and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 13 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | V0 V1 V2 V3 | XX XX |

V0, V1, V2, and V3 form the bus voltage, where the 32-bit floating-point numbers are encoded according to the IEEE 754 standard (the encoding can be tested via the website <https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x15 0x16: Bus Current Feedback 1 Bus Current Feedback 2**

odrv0.ibus

The bus current feedback is a 32-bit floating-point number, where bus current feedback 1 represents its upper 16 bits and bus current feedback 2 represents its lower 16 bits. Users can only obtain the complete bus current feedback by accessing register address 0x15 and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 15 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | I0 I1 I2 I3 | XX XX |

I0, I1, I2, and I3 form the bus current feedback, where the 32-bit floating-point numbers are encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x17 0x18: Q-axis current target value 1 Q-axis current target value 2**

odrv0.axis0.motor.current\_control.Idq\_setpoint

The Q-axis current target value is a 32-bit floating-point number, where Q-axis current target value 1 represents its high 16 bits, and Q-axis current target value 2 represents its low 16 bits. Users can only obtain the complete Q-axis current target value by accessing register address 0x17 and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 17 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | I0 I1 I2 I3 | XX XX |

I0, I1, I2, and I3 form the Q-axis current target value, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website <https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x19 0x1A: Q-axis current feedback 1 Q-axis current feedback 2**

odrv0.axis0.motor.current\_control.Iq\_measured

The Q-axis current feedback is a 32-bit floating-point number, where Q-axis current feedback 1 represents its upper 16 bits and Q-axis current feedback 2 represents its lower 16 bits. Users can only obtain the complete Q-axis current feedback by accessing register address 0x19 and simultaneously retrieving data from both registers. It is not allowed to access any single register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 19 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | I0 I1 I2 I3 | XX XX |

I0, I1, I2, and I3 form the Q-axis current feedback, where the 32-bit floating-point numbers are encoded according to the IEEE 754 standard (the encoding can be tested via the website <https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x31 0x32: Random Endpoint Access Return Value 1 Random Endpoint Access Return Value 2**

Similar to RxSdo in cansimple.

Random endpoint access means that users can read/write any supported parameter. The so-called "endpoint" refers to a parameter, and each parameter has a unique endpoint number. When a user wants to read a certain parameter, they can obtain the endpoint number of that parameter through the endpoint description file, write it into holding register 0xA1, and then read input register 0x31 to obtain the parameter value. Please refer to the description of holding register 0xA1 to obtain the address for downloading the endpoint description file.

The return value of random endpoint access is a 32-bit number, which may be a 16-bit integer, a 32-bit integer, or a 32-bit floating-point number. Among them, random endpoint access return value 1 is its high 16 bits, and random endpoint access return value 2 is its low 16 bits. Users can only obtain the complete return value (parameter value) by accessing register address 0x31 and reading the data of two registers simultaneously. It is not allowed to access any register separately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 00 31 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 04 | 04 | V0 V1 V2 V3 | XX XX |

Among them, V0, V1, V2, and V3 are the 4 bytes from high to low of the return value of random endpoint access, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

2.4 **Keep Register**

Keep the registers readable and writable. The currently supported registers and their functions are as follows:

**0x61: Emergency Stop**

Writing any value to this register will cause the motor to stop immediately.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 61 | XX XX | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 61 | XX XX | XX XX |

**0x62: Node ID**

odrv0.axis0.config.can.node\_id

Write the node ID to this node using Function Code 06, or read the current node ID using Function Code 03.

Write Node ID Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 62 | ID0 ID1 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 62 | ID0 ID1 | XX XX |

Among them, ID0 and ID1 are the high 8 bits and low 8 bits of the new node ID.

Read Current Node ID Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 62 | 00 01 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 02 | ID0 ID1 | XX XX |

**0x63: Status Control**

odrv0.axis0.requested\_state

Use Function Code 06 to write the motor status (change the motor status) to this node.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 63 | S0 S1 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 63 | S0 S1 | XX XX |

Among them, S0 and S1 are the high 8 bits and low 8 bits of the motor status. The status definitions are as follows:

|  |  |
| --- | --- |
| Motor Status Code | Description |
| 1 | Idle |
| 3 | Full Calibration |
| 4 | Motor Calibration |
| 7 | Encoder Calibration |
| 8 | Closed-loop operation |
| 11 | Return to Zero |

For example, setting the motor status to 8 (closed-loop operation) is generally equivalent to "turning on the motor"; setting the motor status to 1 (idle) is generally equivalent to "turning off the motor".

**0x64: Control Mode**

odrv0.axis0.controller.config.control\_mode

The user can use Function Code 06 to modify the value of this register to enter torque control, speed control, or position control mode, or use Function Code 03 to read the current control mode.

Modify Control Mode Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 64 | M0 M1 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 64 | M0 M1 | XX XX |

Read Control Mode Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 64 | 00 01 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 02 | M0 M1 | XX XX |

Among them, M0 and M1 are the high 8 bits and low 8 bits of the control mode. The control mode is defined as follows:

|  |  |
| --- | --- |
| Control Mode | Description |
| 1 | Torque control |
| 2 | Speed Control |
| 3 | Position Control |

**0x65: Input Mode**

odrv0.axis0.controller.config.input\_mode

Users can use Function Code 06 to modify the value of this register to select different target value input modes, such as direct input, filtered input, etc., or use Function Code 03 to read the current input mode.

Modify Input Mode Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 65 | M0 M1 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 65 | M0 M1 | XX XX |

Read Input Mode Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 65 | 00 01 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 02 | M0 M1 | XX XX |

Among them, M0 and M1 are the high 8 bits and low 8 bits of the input mode. The input mode is defined as follows:

|  |  |  |
| --- | --- | --- |
| Control Mode | Description | Detailed Explanation |
| 1 | Direct Input | The target value entered by the user is directly used as the target value of the drive |
| 2 | Speed Ramp | The speed target value of the driver will reach the speed target value input by the user at a certain slope within a certain period of time, and is only used for speed control |
| 3 | Position Filtering | The position target value of the drive is the value obtained after the position target value input by the user has been filtered, and is only used for position control |
| 5 | Trapezoidal Curve | The driver will drive the motor to reach the target value input by the user according to a trapezoidal speed curve, where the rising slope (acceleration), falling slope (deceleration), and coasting speed of the trapezoid can all be set. Please refer to subsequent registers 0x70 to 0x75. |
| 6 | Torque Ramp | The torque target value of the driver will reach the target torque value input by the user at a certain slope within a certain period of time, and is only used for torque control |

**0x66 0x67: Position Control Target 1 Position Control Target 2**

odrv0.axis0.controller.input\_pos

The position control target is a 32-bit floating-point number, where Position Control Target 1 represents its high 16 bits, and Position Control Target 2 is its low 16 bits. Users can only access register address 0x66 to read/write data from/to both registers simultaneously, thereby obtaining/modifying the complete position control target. It is not allowed to access any register separately.

Modify/Update Position Control Target Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 66 | 00 02 | 04 | P0 P1 P2 P3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 66 | 00 02 | XX XX |

Read Position Control Target Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 66 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | P0 P1 P2 P3 | XX XX |

Among them, P0, P1, P2, and P3 are the 4 ByteDance from high to low of the position control target, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x68 0x69: Speed Control Target 1 Speed Control Target 2**

odrv0.axis0.controller.input\_vel

The speed control target is a 32-bit floating-point number, where speed control target 1 represents its high 16 bits and speed control target 2 represents its low 16 bits. Users can only access register address 0x68 to read/write the data of both registers simultaneously, thereby obtaining/modifying the complete speed control target. It is not allowed to access any single register separately.

Modify/Update Speed Control Target Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 68 | 00 02 | 04 | V0 V1 V2 V3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 68 | 00 02 | XX XX |

Read speed control target request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 68 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | V0 V1 V2 V3 | XX XX |

Among them, V0, V1, V2, and V3 are the 4 ByteDances from high to low of the speed control target, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x6A 0x6B: Torque Control Target 1 Torque Control Target 2**

odrv0.axis0.controller.input\_torque

The torque control target is a 32-bit floating-point number, where Torque Control Target 1 represents its upper 16 bits and Torque Control Target 2 represents its lower 16 bits. Users can only access register address 0x6A to read/write data from/to both registers simultaneously, thereby obtaining/modifying the complete torque control target. It is not allowed to access any single register separately.

Modify/Update Torque Control Target Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 6A | 00 02 | 04 | T0 T1 T2 T3 | XX XX |

Motor Response:

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 6A | 00 02 | XX XX |

Read torque control target request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 6A | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | T0 T1 T2 T3 | XX XX |

Among them, T0, T1, T2, and T3 are the 4 bytes from high to low of the torque control target, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x6C 0x6D: Speed Limit 1 Speed Limit 2**

odrv0.axis0.controller.config.vel\_limit

The speed limit is a 32-bit floating-point number, where speed limit 1 is its high 16 bits and speed limit 2 is its low 16 bits. Users can only obtain/modify the complete speed limit by accessing register address 0x6C and simultaneously reading/writing data from/to both registers. It is not allowed to access any register individually.

Modify/Update Speed Limit Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 6C | 00 02 | 04 | V0 V1 V2 V3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 6C | 00 02 | XX XX |

Read speed limit request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 6C | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | V0 V1 V2 V3 | XX XX |

Among them, V0, V1, V2, and V3 are the 4 ByteDance from highest to lowest of the speed limit, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x6E 0x6F: Current Limit 1 Current Limit 2**

odrv0.axis0.motor.config.current\_lim

The current limit is a 32-bit floating-point number, where Current Limit 1 represents its high 16 bits and Current Limit 2 represents its low 16 bits. Users can only obtain/modify the complete current limit by accessing register address 0x6E and simultaneously reading/writing data from/to both registers. It is not allowed to access any register individually.

Modify/Update Current Limit Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 6E | 00 02 | 04 | I0 I1 I2 I3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 6E | 00 02 | XX XX |

Read Current Limit Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 6E | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | I0 I1 I2 I3 | XX XX |

Among them, I0, I1, I2, and I3 are the 4 ByteDance from high to low of the current limit, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x70 0x71: Trapezoidal Position Control Speed Limit 1 Trapezoidal Position Control Speed Limit 2**

odrv0.axis0.trap\_traj.config.vel\_limit

The trapezoidal position control speed limit is a 32-bit floating-point number, where trapezoidal position control speed limit 1 is its high 16 bits, and trapezoidal position control speed limit 2 is its low 16 bits. Users can only access register address 0x70 to read/write data from/to both registers simultaneously, thereby obtaining/modifying the complete trapezoidal position control speed limit. It is not allowed to access any register individually.

Modify/Update Trapezoidal Position Control Speed Limit Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 70 | 00 02 | 04 | V0 V1 V2 V3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 70 | 00 02 | XX XX |

Read trapezoidal position control speed limit request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 70 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | V0 V1 V2 V3 | XX XX |

Among them, V0, V1, V2, and V3 are the 4 bytes from high to low of the trapezoidal position control speed limit, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x72 0x73: Trapezoidal Position Control Acceleration Limit 1 Trapezoidal Position Control Acceleration Limit 2**

odrv0.axis0.trap\_traj.config.accel\_limit

The trapezoidal position control acceleration limit is a 32-bit floating-point number, where trapezoidal position control acceleration limit 1 represents its upper 16 bits, and trapezoidal position control acceleration limit 2 represents its lower 16 bits. Users can only access register address 0x72 to read/write data from both registers simultaneously, thereby obtaining/modifying the complete trapezoidal position control acceleration limit. It is not allowed to access any single register separately.

Modify/Update Trapezoidal Position Control Acceleration Limit Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 72 | 00 02 | 04 | A0 A1 A2 A3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 72 | 00 02 | XX XX |

Read trapezoidal position control acceleration limit request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 72 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | A0 A1 A2 A3 | XX XX |

Among them, A0, A1, A2, and A3 are the 4 bytes from high to low of the trapezoidal position control acceleration limit, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x74 0x75: Trapezoidal Position Control Deceleration Limit 1 Trapezoidal Position Control Deceleration Limit 2**

odrv0.axis0.trap\_traj.config.decel\_limit

The trapezoidal position control deceleration limit is a 32-bit floating-point number, where trapezoidal position control deceleration limit 1 is its high 16 bits, and trapezoidal position control deceleration limit 2 is its low 16 bits. Users can only obtain/modify the complete trapezoidal position control deceleration limit by accessing register address 0x74 and simultaneously reading/writing data from/to both registers. It is not allowed to access any single register separately.

Modify/Update Trapezoidal Position Control Deceleration Limit Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 74 | 00 02 | 04 | D0 D1 D2 D3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 74 | 00 02 | XX XX |

Read trapezoidal position control deceleration limit request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 74 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | D0 D1 D2 D3 | XX XX |

Among them, D0, D1, D2, and D3 are the 4 ByteDance from high to low of the trapezoidal position control deceleration limit, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x76 0x77: Inertia 1 Inertia 2**

odrv0.axis0.controller.config.inertia

The inertia is a 32-bit floating-point number, where Inertia 1 represents its high 16 bits and Inertia 2 represents its low 16 bits. Users can only access register address 0x76 to read/write the data of both registers simultaneously, thereby obtaining/modifying the complete inertia. It is not allowed to access any single register separately.

Modify/Update Inertia Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 76 | 00 02 | 04 | I0 I1 I2 I3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 76 | 00 02 | XX XX |

Read Inertia Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 76 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | I0 I1 I2 I3 | XX XX |

Among them, I0, I1, I2, and I3 are the 4 bytes of inertia from highest to lowest, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x78 0x79: Position Gain 1 Position Gain 2**

odrv0.axis0.controller.config.pos\_gain

The position gain is a 32-bit floating-point number, where position gain 1 is its high 16 bits and position gain 2 is its low 16 bits. Users can only obtain/modify the complete position gain by accessing register address 0x78 and simultaneously reading/writing data from/to both registers. It is not allowed to access any single register separately.

Modify/Update Position Gain Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 78 | 00 02 | 04 | G0 G1 G2 G3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 78 | 00 02 | XX XX |

Read position gain request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 78 | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | G0 G1 G2 G3 | XX XX |

Among them, G0, G1, G2, and G3 are the 4 bytes of the position gain from high to low, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x7A 0x7B: Speed Gain 1 Speed Gain 2**

odrv0.axis0.controller.config.vel\_gain

The speed gain is a 32-bit floating-point number, where speed gain 1 is its high 16 bits and speed gain 2 is its low 16 bits. Users can only obtain/modify the complete speed gain by accessing register address 0x7A and simultaneously reading/writing data from/to both registers. It is not allowed to access any register individually.

Modify/Update Speed Gain Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 7A | 00 02 | 04 | G0 G1 G2 G3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 7A | 00 02 | XX XX |

Read Speed Gain Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 7A | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | G0 G1 G2 G3 | XX XX |

Among them, G0, G1, G2, and G3 are the 4 bytes of the speed gain from high to low, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x7C 0x7D: Speed Integral Gain 1 Speed Integral Gain 2**

odrv0.axis0.controller.config.vel\_integrator\_gain

The speed integral gain is a 32-bit floating-point number, where speed integral gain 1 is its high 16 bits and speed integral gain 2 is its low 16 bits. Users can only obtain/modify the complete speed integral gain by accessing register address 0x7C and simultaneously reading/writing data from/to both registers. It is not allowed to access any single register separately.

Modify/Update Speed Integral Gain Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 7C | 00 02 | 04 | G0 G1 G2 G3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 7C | 00 02 | XX XX |

Read speed integral gain request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 00 7C | 00 02 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 03 | 04 | G0 G1 G2 G3 | XX XX |

Among them, G0, G1, G2, and G3 are the 4 bytes of the speed integral gain from high to low, where the 32-bit floating-point number is encoded according to the IEEE 754 standard (the encoding can be tested via the website<https://www.h-schmidt.net/FloatConverter/IEEE754.html>).

**0x7E: Reboot**

odrv0.reboot()

Writing any value to this register will cause the motor driver to restart.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 7E | XX XX | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 7E | XX XX | XX XX |

**0x7F: Clear Exception**

odrv0.clear\_errors()

Writing any value to this register will clear all errors and exceptions.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 7F | XX XX | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 7F | XX XX | XX XX |

**0x80: Storage Configuration**

odrv0.save\_configuration()

Writing any value to this register will save all current parameters to the drive's Flash for permanent storage.

Host Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 80 | XX XX | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 80 | XX XX | XX XX |

**0xA1: Random Endpoint Access Endpoint Number**

Random endpoint access means that users can read/write any supported parameter. The so-called "endpoint" refers to a parameter, and each parameter has a unique endpoint number. When a user wants to read/write a certain parameter, they can obtain the endpoint number of that parameter through the endpoint description file, write it into holding register 0xA1, and then read input register 0x31 to obtain the parameter value, or write it into holding register 0xA2 to update the parameter value.

The endpoint number is a 16-bit unsigned integer. Please download the JSON file for the endpoint numbers corresponding to all parameters and interface functions:

* Version 0.5.13: https://bl.cyberbeast.cn/actuator/endpoints\_0.5.13.json
* Version 0.5.14: https://bl.cyberbeast.cn/actuator/endpoints\_0.5.14.json

Write Random Endpoint Number Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 A1 | E0 E1 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 06 | 00 A1 | E0 E1 | XX XX |

Among them, E0 and E1 are the two bytes of the 16-bit endpoint number from high to low.

**0xA2 0xA3: Random Endpoint Access Write Value 1 Random Endpoint Access Write Value 2**

The random endpoint access write value is a 32-bit number, which may be a 16-bit integer, a 32-bit integer, or a 32-bit floating-point number. Among them, random endpoint access write value 1 is its high 16 bits, and random endpoint access write value 2 is its low 16 bits. Users can only update the endpoint value (parameter value) by accessing register address 0xA2 and writing data to two registers simultaneously. It is not allowed to access any register individually.

Write Request:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Byte Length | Register Value | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 1 ByteDance | 4 ByteDance | 2 ByteDance |
| XX | 10 | 00 A2 | 00 02 | 04 | V0 V1 V2 V3 | XX XX |

Motor Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Function Code | Register Address | Number of Registers | Checksum |
| 1 ByteDance | 1 ByteDance | 2 ByteDance | 2 ByteDance | 2 ByteDance |
| XX | 10 | 00 A2 | 00 02 | XX XX |

Among them, V0, V1, V2, and V3 are the 4 bytes of the written value from high to low. If this endpoint is a 32-bit floating-point number, it is encoded according to the IEEE 754 standard (the encoding can be tested via the website [IEEE-754 Floating Point Converter](https://www.h-schmidt.net/FloatConverter/IEEE754.html) ).

2.5 **Modbus Protocol in Practice**

Assume the address of the motor is 1.

2.5.1 **Practical: Speed Control**

The sequence of transmitted frames is as follows:

|  |  |  |
| --- | --- | --- |
| Frame Data | Register | Description |
| 01 06 00 64 00 02 XX XX | Keep register 0x64 | Set the control mode to 2 (speed control) |
| 01 06 00 65 00 02 XX XX | Keep register 0x65 | Set the input mode to 2 (Speed Ramp) |
| 01 06 00 63 00 08 XX XX | Keep register 0x63 | Change the motor state to 8 (closed-loop operation). |
| 01 10 00 68 00 02 04 40 13 33 33 XX XX | Keep register 0x68 | Set the target speed to 2.3 r/s, which converts to 40 13 33 33 as a 32-bit floating point number. |

2.5.2 **Practical: Position Control**

The sequence of transmitted frames is as follows:

|  |  |  |
| --- | --- | --- |
| Frame Data | Register | Description |
| 01 06 00 64 00 03 XX XX | Keep register 0x64 | Set the control mode to 3 (position control) |
| 01 06 00 65 00 03 XX XX | Keep register 0x65 | Set the input mode to 3 (position filtering) |
| 01 06 00 63 00 08 XX XX | Keep register 0x63 | Change the motor state to 8 (closed-loop operation). |
| 01 10 00 66 00 02 04 40 49 0F F9 XX XX | Keep register 0x66 | Set the targetpositionto 3.1416r, which converts to 40 49 0F F9 in 32-bit floating point. |

3. **CANOpen Protocol**

3.1 **Overview**

CANOpen is a communication protocol specification for the automation field, and this product is implemented in accordance with the CiA301, CiA302, and CiA402 specifications.

**Main Functions**

* **Standardized Motion Control Modes**: CiA 402 defines multiple operating modes suitable for different motion control requirements:

Profile Position Mode (轮廓位置模式): Used for precise position control (e.g., robotic arm positioning).

Cyclic Synchronous Velocity Mode (周期同步速度模式): Simple speed control (no complex trajectory planning required).

Cyclic Synchronous Torque Mode (Periodic Synchronous Torque Mode): Simple torque control (no complex trajectory planning required).

Cyclic Synchronous Position Mode (周期同步位置模式): Simple position control (no complex trajectory planning required).

Homing Mode: Used to find the mechanical zero point (such as the machine tool returning to the reference point).

* **Finite-State Machine Management**: CiA 402 defines the state transition logic of the drive to ensure the safe operation of the device.

3.2 **Protocol Frame Format**

The following table shows the frame format: CANOpen communication uses the standard frame format, data frame, 11-bit ID, 8 ByteDance data, as shown in the following table (left is MSB, right is LSB):

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Domain** | **COB-ID / CAN ID( 11bits)** |  | **Data( 8 bytes)** |
| Segmentation | Bit10 ~ Bit7 | Bit6 ~ Bit0 | Byte0 ~ Byte7 |
| Description | Function Code | node\_id | Communication Data |

* Function Code: Represents the type and priority of the CANOpen message. The smaller the function code, the higher the priority. See the table below.
* node\_id: Represents the unique ID of this motor on the bus, which can be read and set in odrivetool usingodrv0.axis0.config.can.node\_id.
* Communication data: 8 ByteDance, byte order is little endian, content depends on the function code type, see 4.3 - 4.8.

The function code table is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Communication Object** | **Function Code** | **Node Address/Node-ID** | **COB-ID** | **Corresponding Object Index** |
| Network Management | 0000b | 0 | 0h | - |
| Synchronized Object | 0001b | 0 | 80h | 1005h, 1006h |
| Emergency Message Object | 0001b | 1-127 | 80h + node\_id | 1014h |
| RPDO1 | 0100b | 1-127 | 200h + node\_id | 1400h |
| RPDO2 | 0110b | 1-127 | 300h + node\_id | 1401h |
| RPDO3 | 1000b | 1-127 | 400h + node\_id | 1402h |
| RPDO4 | 1010b | 1-127 | 500h + node\_id | 1403h |
| TPDO1 | 0011b | 1-127 | 180h + node\_id | 1800h |
| TPDO2 | 0101b | 1-127 | 280h + node\_id | 1801h |
| TPDO3 | 0111b | 1-127 | 380h + node\_id | 1802h |
| TPD04 | 1001b | 1-127 | 480h + node\_id | 1803h |
| TSDO | 1100b | 1-127 | 0x600+node\_id | 1200h |
| RSDO | 1011b | 1-127 | 0x580+node\_id | 1200h |
| NMT (Heartbeat) | 1110b | 1-127 | 0x700+node\_id | 1017h |

3.3 **NMT (Network Management)**

NMT is used by the master station (Master) to centrally manage the operating status of all slave stations (Slave). Its COB-ID is always 0, and the data contains 2 ByteDance, as shown in the following table:

|  |  |  |
| --- | --- | --- |
| **COB-ID** | **Data Byte0** | **Data Byte1** |
| 0x000 | Status Command Word | node\_id |

When node\_id is 0, it indicates a broadcast message, which is valid for all slave devices in the network.

The status command word is used to directly control the operating status of the slave device, and its meanings are as shown in the following table:

|  |  |
| --- | --- |
| Status Command Word | Description |
| 0x01 | Status changed to "operational" |
| 0x02 | Status changed to "stopped" |
| 0x80 | Status changed to "pre-operational" |
| 0x81 | Reset node "reset node" |
| 0x82 | Reset communication “reset communication” |

For example, after the master station connects to the drive, it first sends the following NMT command to start all nodes on the bus:

|  |  |
| --- | --- |
| COB-ID | Data |
| 0x000 | 01 00 |

3.4 **SYNC (Synchronization Signal)**

SYNC (Synchronization Signal) is used by the Master to synchronize the time of all nodes on the bus, mainly for synchronizing PDO messages among nodes. Its COB-ID is fixed at 0x80, and the data field is empty.

For example, after the host uses the NMT command to start the nodes on the bus, it generally sends a SYNC message to make the nodes start transmitting PDO data.

3.5 **Emergency Message**

Currently not supported.

3.6 **PDO**

PDO (Process Data Object) is a communication method prepared for real-time data transmission. When users wish to periodically acquire node data or report data under event-driven conditions, PDO is used for transmission. PDO is only used when the node state is in the "operational" state, so users must first use NMT messages to switch to the "operational" state before using PDO. Its COB-ID follows the table below, and the data field length is fixed at 0 to 8 ByteDance, with the specific content determined by PDO Mapping.

According to the difference between reception and transmission, PDOs can be divided into RPDOs (sent from the master to the slave) and TPDOs (sent from the slave to the master). The final transmission method and content of PDOs are jointly determined by communication parameters and mapping parameters. This drive uses 4 RPDOs and 4 TPDOs to implement PDO transmission. The relevant object list is shown in the following table, and the specific mapping content can be found in the object dictionary.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name |  | COB-ID | Communication Object | mapping object |
| RPDO | 1 | 200h + Node\_ID | 1400h | 1600h |
| 2 | 300h + Node\_ID | 1401h | 1601h |
| 3 | 400h + Node\_ID | 1402h | 1602h |
| 4 | 500h + Node\_ID | 1403h | 1603h |
| TPDO | 1 | 180h + Node\_ID | 1800h | 1A00h |
| 2 | 280h + Node\_ID | 1801h | 1A01h |
| 3 | 380h + Node\_ID | 1802h | 1A02h |
| 4 | 480h + Node\_ID | 1803h | 1A03h |

3.6.1 **PDO Transmission Framework**

PDO belongs to process data, which means unidirectional transmission, and does not require the receiving node to respond to the CAN message for confirmation. In terms of communication terminology, it belongs to the "producer-consumer" model.

3.6.2 **PDO Communication Configuration**

* **PDO Transmission Type**

The transmission type of PDO is located at sub-index 02 of the communication parameters (RPDO: 1400h - 1403h, TPDO: 1800h - 1803h). Different values represent different transmission types, which define the method by which the drive triggers TPDO transmission or processes the received RPDO. The specific correspondence is shown in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Communication Type Value | Synchronize |  | Asynchronous |
| Loop | Non-cyclic |  |
| 0 |  | √ |  |
| 1~240 | √ |  |  |
| 241~253 | / | / | / |
| 254、255 |  |  | √ |

When the transmission type of TPDO is 0, if the mapped data changes and a synchronization frame is received, the TPDO is sent; when the transmission type of TPDO is 1 to 240, the TPDO is sent upon receiving the corresponding number of synchronization frames.

When the transmission type of TPDO is 254 or 255, the TPDO is sent when the mapped data changes or the event timer expires. Note: In this type, the sub-index 03 inhibit time of the drive's TPDO communication parameters (TPDO: 1800h - 1803h) cannot be 0.

When the transmission type of RPDO is 0 to 255, the received data is directly updated to the application.

* **Prohibited Time**

A prohibit time is set for TPDO and stored in sub-index 03 of communication parameters (1800h - 1803h) to prevent the CAN network from being continuously occupied by lower-priority PDOs. After setting the value, the transmission interval of the same TPDO shall not be less than the time corresponding to this parameter.

* **Event Timer**

For TPDOs of asynchronous transmission (transmission type 254 or 255), an event timer is defined, located at sub-index 05 of communication parameters (1800h - 1803h). The event timer can also be regarded as a trigger event, which will also trigger the corresponding TPDO transmission. If other events such as data changes occur within the timer's operating cycle, the TPDO will also be triggered, and the event timer will be immediately reset.

3.6.3 **PDO Mapping Parameters**

PDO mapping refers to the process of associating variables in the Object Dictionary with PDOs. The data length of each PDO signal can reach up to 8 ByteDance, and one or more objects can be mapped simultaneously. Through mapping, the specific content and format of PDO transmission can be defined.

Sub-index 0 of the PDO mapping parameters (RPDO: 1600h - 1603h, TPDO: 1A00h - 1A03h) records the number of objects specifically mapped to this PDO, while sub-indexes 1 - 8 are the mapping content, referring to pointers to variables in the object dictionary that can be mapped to the PDO, including the index, sub-index, and the length of the mapped object.

The index and sub-index together determine the position of the object in the object dictionary, and the object length specifies the specific bit length of the object, represented in hexadecimal, i.e.:

|  |  |
| --- | --- |
| Object Length | Bit Length |
| 08h | 8-bit |
| 10h | 16-bit |
| 20h | 32-bit |

For example:

The mapping content of sub-index 1 of RPDO1 is 60710010, indicating that the 16-bit target torque 6071h-00h is mapped to RPDO1.

If the host wants to send a target torque with a value of 1000,

According to the mapping, during the transmission process, RPDO1 has 2 ByteDance in the data segment, and the transmission signal is E8 03 00 00 00 00 00 00.

3.7 **SDO**

Using SDO (Service Data Object) allows reading or writing object data in the object dictionary. Generally, SDO is only used for parameter configuration, not for data exchange during operation. For real-time operational data exchange, PDO is recommended. The COB-ID of SDO follows the table below, with the data field length fixed at 8 ByteDance, and the specific content also follows the table below.

3.7.1 **SDO Transmission Framework**

The SDO transmission mode follows the Client-Server model, i.e., the one-request-one-response mode. Initiated by the SDO Client in the CAN bus network, the SDO Server responds. Therefore, data exchange between SDOs requires at least two CAN messages to be realized, and the COB-IDs of the two CAN messages are inconsistent.

3.7.2 **SDO Transfer Message**

Currently, SDO transmission only supports object data transmission of no more than 4 ByteDance, that is, only accelerated SDO transmission is supported. The SDO transmission message is shown in the following table. The data segment uses little-endian mode, that is, the low-order bits are arranged first and the high-order bits are arranged last. All SDO message data segments must be 8 ByteDance. The format of the SDO transmission message is as shown in the following table:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| COB-ID | Data |  |  |  |  |  |  |  |
| 580h+Node\_ID/  600h+Node\_ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Command Code | Index |  | Subindex | Data |  |  |  |

Among them, the command code specifies the transfer type and transfer data length of this segment of SDO, the index and sub-index are the position of the object in the object dictionary, and the data is the value of the object.

* SDO Accelerated Write Transfer Message

For writes of no more than 4 ByteDance, accelerated SDO transfer (also known as upload) is used. Depending on the read/write method and the length of the content data, the transfer messages vary. The accelerated SDO write messages are as shown in the following table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | COB-ID | Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| Client→ | 0x600+  node\_id | 0x23 | Index |  | Subindex | Data |  |  |  |
| 0x27 | Index |  | Subindex | Data |  |  | / |
| 0x2B | Index |  | Subindex | Data |  | / | / |
| 0x2F | Index |  | Subindex | Data | / | / | / |
| ←Server | 0x580+  node\_id | 0x60 | Index |  | Subindex | / | / | / | / |
| 0x80 | Index |  | Subindex | Abort Code |  |  |  |

Note: "/" indicates that there is data but it is not considered. When writing data, it is recommended to write 0. The same applies hereinafter.

* SDO Accelerated Read Transfer Message

For reads of no more than 4 ByteDance, accelerated SDO transmission (also known as download) is used. Depending on the read/write method and the length of the content data, the transmission messages vary. The accelerated SDO read messages are as shown in the following table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | COB-ID | Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| Client→ | 0x600+  node\_id | 0x40 | Index |  | Subindex | / | / | / | / |
| ←Server | 0x580+  node\_id | 0x43 | Index |  | Subindex | Data |  |  |  |
| 0x47 | Index |  | Subindex | Data |  |  | / |
| 0x4B | Index |  | Subindex | Data |  | / | / |
| 0x4F | Index |  | Subindex | Data | / | / | / |
| 0x80 | Index |  | Subindex | Abort Code |  |  |  |

3.8 **NMT (Heartbeat)**

Heartbeat messages are a mechanism used to monitor the status of nodes. Slave devices will periodically send heartbeat packets to the Master or the entire network to report their current operating status. The frame format is as follows:

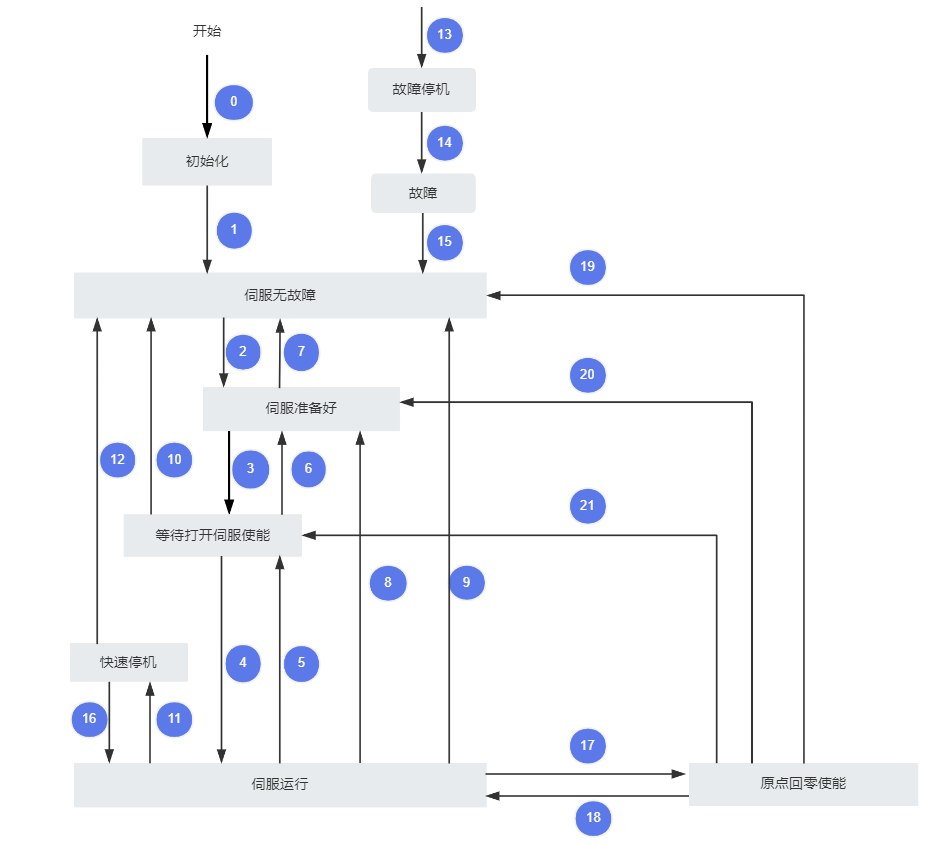
|  |  |
| --- | --- |
| COB-ID | Data Byte0 |
| 0x700+node\_id | Status Word |

The status word displays the current operating status of the slave, and its meaning is shown in the following table

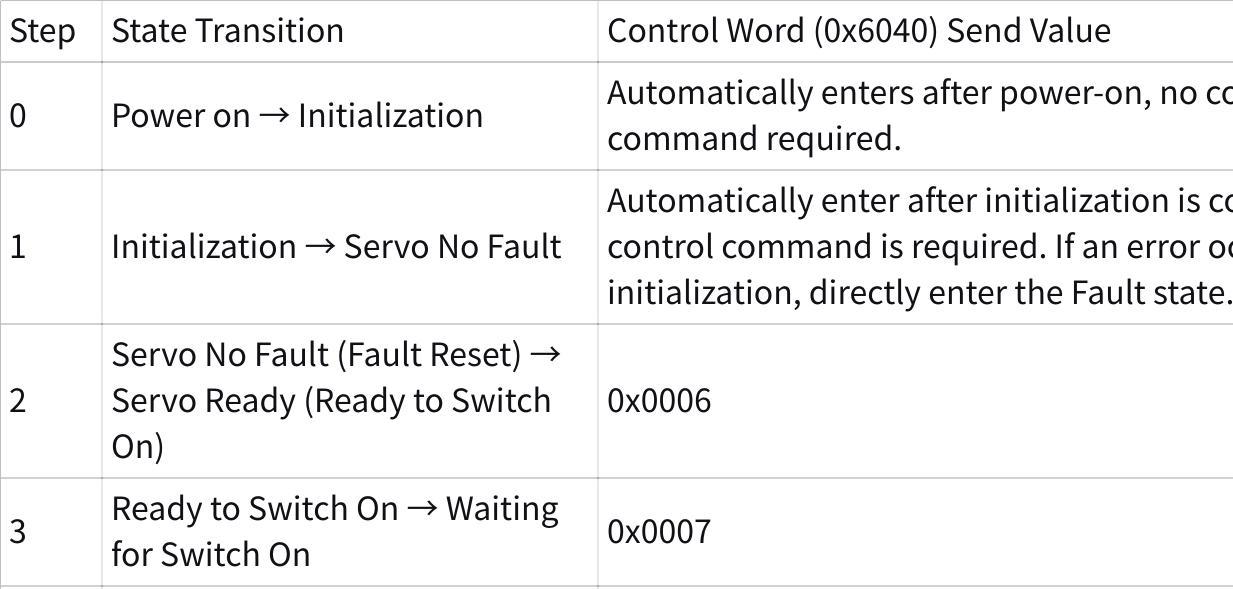
|  |  |  |
| --- | --- | --- |
| Status Value (Hex) | Status Name | Description |
| 0x00 | Boot-up (Startup) | Node initialization completed |
| 0x04 | Stopped（Stopped） | The node is in the NMT stopped state and only responds to NMT messages |
| 0x05 | operational (Operation) | Node allows all communication services |
| 0x7F | Pre-operational (Pre-operation) | The node can perform SDO communication but does not process PDO |
| 0x01-0x03, 0x06-0x7E | - | Retained value, normal nodes should not send,  Sending indicates a problem with the implementation of the protocol stack |

3.9 **Servo Status**

To operate the servo drive in the specified state, it is necessary to guide the servo drive according to the process specified in the CiA402 protocol when using this drive.



The explanation of control words and state switching is as shown in the following table:



**点击图片可查看完整电子表格**

Since bit10~bit15 (bit14 is meaningless) of status word 6041h are related to the operating status of each servo mode, they are all represented as "0" in the above table. For the specific status of each bit, please refer to each servo operating mode.

3.10 **CANOpen Protocol Practical Application**

Assume the motor is a slave, with Node\_ID being 1. Note that in actual combat, the unit of CAN signals is the command unit. If conversion to user units is required, please refer to [the conversion factor section of the Supplementary Manual for CANOpen Drive of the Guardian Beast](https://bcnyljrhe70u.feishu.cn/wiki/RkxswgglNiX2mckH9CdcVNmQnXe).

3.10.1 **Practical: Periodic Synchronous Torque Mode**

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CAN ID | Frame Type | Frame Data | Frame Data |
| 00 | Data Frame | 01 01 | Message: NMT  Parameter: 01  The NMT state of the motor with Node\_ID 1 switches to "operational" |
| 601 | Data Frame | 2F 60 60 00 0A 00 00 00 | Message: Operating Mode  Parameter: 10  Set the control mode to torque control and the input mode to ramp torque |
| 601 | Data Frame | 2B 40 60 00 06 00 00 00 | Message: Control Word  Parameter: 6  Set the servo status to servo ready |
| 601 | Data Frame | 2B 40 60 00 07 00 00 00 | Message: Control Word  Parameter: 7  Set the servo status to waiting to enable the servo |
| 601 | Data Frame | 2B 40 60 00 0F 00 00 00 | Message: Control Word  Parameter: 16  Set the servo state to servo operation |
| 501 | Data Frame | 0F 00 64 00 | Message: Target Torque  Parameter: 0.5  Set the target torque of the third and fourth ByteDance to 0.5 Nm via PDO |

3.10.2 **Practical: Periodic Synchronous Velocity Mode**

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CAN ID | Frame Type | Frame Data | Frame Data |
| 00 | Data Frame | 01 01 | Message: NMT  Parameter: 01  The NMT state of the motor with Node\_ID 1 switches to "operational" |
| 601 | Data Frame | 2F 60 60 00 09 00 00 00 | Message: Operating Mode  Parameter: 9  Set the control mode to speed control and the input mode to ramp speed |
| 601 | Data Frame | 2B 40 60 00 06 00 00 00 | Message: Control Word  Parameter: 6  Set the servo status to servo ready |
| 601 | Data Frame | 2B 40 60 00 07 00 00 00 | Message: Control Word  Parameter: 7  Set the servo status to waiting to enable the servo |
| 601 | Data Frame | 2B 40 60 00 0F 00 00 00 | Message: Control Word  Parameter: 16  Set the servo state to servo operation |
| 401 | Data Frame | 0F 00 00 08 00 00 | Message: Target Speed  Parameter: 0.125  Set the target speed of the third, fourth, fifth, and sixth ByteDance to 0.125 revolutions per second through PDO |

3.10.3 **Practical: Cyclic Synchronous Position Mode**

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CAN ID | Frame Type | Frame Data | Frame Data |
| 00 | Data Frame | 01 01 | Message: NMT  Parameter: 01  The NMT state of the motor with Node\_ID 1 switches to "operational" |
| 601 | Data Frame | 2F 60 60 00 08 00 00 00 | Message: Operating Mode  Parameter: 8  Set the control mode to position control and the input mode to filtered position |
| 601 | Data Frame | 2B 40 60 00 06 00 00 00 | Message: Control Word  Parameter: 6  Set the servo status to servo ready |
| 601 | Data Frame | 2B 40 60 00 07 00 00 00 | Message: Control Word  Parameter: 7  Set the servo status to waiting to enable the servo |
| 601 | Data Frame | 2B 40 60 00 0F 00 00 00 | Message: Control Word  Parameter: 16  Set the servo state to servo operation |
| 301 | Data Frame | 0F 00 00 40 00 00 | Message: Target Location  Parameter: 1  Set the target position of the third, fourth, fifth, and sixth ByteDance to 1 revolution via PDO |

3.10.4 **Actual Combat: Contour Position Mode**

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CAN ID | Frame Type | Frame Data | Frame Data |
| 00 | Data Frame | 01 01 | Message: NMT  Parameter: 01  The NMT state of the motor with Node\_ID 1 switches to "operational" |
| 601 | Data Frame | 2F 60 60 00 01 00 00 00 | Message: Operating Mode  Parameter: 1  Set the control mode to position control and the input mode to trapezoidal curve position |
| 601 | Data Frame | 23 81 60 00 00 80 0C 00 | Message: Contour Velocity  Parameter: 50  Set the contour speed to 50 revolutions per second |
| 601 | Data Frame | 23 83 60 00 00 80 02 00 | Message: Contour Acceleration  Parameter: 16  Set the contour acceleration to 10 rev/s^2 |
| 601 | Data Frame | 23 83 60 00 00 80 02 00 | Message: Contour Deceleration  Parameter: 16  Set the contour deceleration to 10 rev/s^2 |
| 601 | Data Frame | 2B 40 60 00 06 00 00 00 | Message: Control Word  Parameter: 6  Set the servo status to servo ready |
| 601 | Data Frame | 2B 40 60 00 07 00 00 00 | Message: Control Word  Parameter: 7  Set the servo status to waiting to enable the servo |
| 601 | Data Frame | 2B 40 60 00 0F 00 00 00 | Message: Control Word  Parameter: 16  Set the servo state to servo operation |
| 301 | Data Frame | 0F 00 00 40 00 00 | Message: Target Location  Parameter: 1  Set the target position of the third, fourth, fifth, and sixth ByteDance to 1 revolution via PDO |

3.10.5 **Practical: Zero Return Mode**

The sequence for sending CAN messages is as follows: The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CAN ID | Frame Type | Frame Data | Frame Data |
| 00 | Data Frame | 01 01 | Message: NMT  Parameter: 01  The NMT state of the motor with Node\_ID 1 switches to "operational" |
| 601 | Data Frame | 2F 60 60 00 06 00 00 00 | Message: Operating Mode  Parameter: 6  Set the control mode to the homing mode |
| 601 | Data Frame | 2F 98 60 00 11 00 00 00 | Message: Zero Return Mode  Parameter: 17  Set the zero return mode to 17, and it can only be set to 17; other modes are not currently supported |
| 601 | Data Frame | 23 7C 60 00 00 40 01 00 | Message: Zero Return Offset  Parameter: 5  Set the zero return offset to 5 revolutions |
| 601 | Data Frame | 23 99 60 01 00 20 00 00 | Message: Return-to-zero speed  Parameter: 16  Set the contour deceleration to 0.5 rev/s |
| 601 | Data Frame | 23 9A 60 00 00 00 04 00 | Message: Zero Return Acceleration  Parameter: 0.5  Set the contour deceleration to 16 rev/s^2 |
| 601 | Data Frame | 2B 40 60 00 06 00 00 00 | Message: Control Word  Parameter: 6  Set the servo status to servo ready |
| 601 | Data Frame | 2B 40 60 00 07 00 00 00 | Message: Control Word  Parameter: 7  Set the servo status to waiting to enable the servo |
| 601 | Data Frame | 2B 40 60 00 0F 00 00 00 | Message: Control Word  Parameter: 16  Set the servo state to servo operation |
| 601 | Data Frame | 2B 40 60 00 1F 00 00 00 | Message: Control Word  Parameter: 31  Set the servo status to homing in progress |

**3.9.6 Practical Exercise: Zeroing Mode**

If it is found that the current position value is prone to overflow during use, you can configure a suitable conversion factor yourself, or execute this zeroing mode

Note: Prerequisites for the zeroing operation:

1. The motor is in an idle state (unenabled state)

2. In homing mode

3. The homing mode is 35

When the above three conditions are met, perform the zeroing operation and set the current position as the zero point

The sequence for sending CAN messages is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| CAN ID | Frame Type | Frame Data | Frame Data |
| 601 | Data Frame | 2F 60 60 00 06 00 00 00 | Message: Operating Mode  Parameter: 6  Set the control mode to the homing mode |
| 601 | Data Frame | 2F 98 60 00 23 00 00 00 | Message: Zero Return Mode  Parameter: 17  Set the zero return mode to 35, and clear all actual values of the speed loop and position loop |
| 601 | Data Frame | 2F 60 60 00 01 00 00 00 | Message: Operating Mode  Parameter: 1  After completing the zeroing operation, remember to exit the homing mode. Here is an example: set the control mode to contour position mode |

3.11 **Object Dictionary**

The Object Dictionary, i.e., the EDS file, describes the device's functions and communication parameters. To communicate with this product via CANOpen, the EDS file of this product must be imported. Please download it via the following link: <https://bl.cyberbeast.cn/actuator/cb_0.1.eds>

3.11.1 **Allocation Overview of Object Group 1000h**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 1000h | - | Device Type | RO | NO | Uint32 | VAR | Uint32 | 0x20192 |
| 1001h | - | Error Register | RO | RPDO | Uint8 | VAR | Uint8 | 0x0 |
| 1003h | 0h | Number of Errors | RO | NO | Uint32 | ARR | Uint8 | - |
| 1~10h | Standard Error Field | RW | NO | Uint32 | - | Uint32 | 0 |
| 1005h | - | Synchronous Message COB-ID | RW | NO | Uint32 | VAR | Uint32 | 0x80 |
| 1006h | - | Synchronous Cycle Period | RW | NO | Uint32 | VAR | Uint32 | 1000 |
| 1008h | - | Manufacturer Device Name | CONST | NO | String | VAR | String | MW6010-8 |
| 1009h | - | Hardware Version | CONST | NO | String | VAR | String | 3.10 |
| 100Ah | - | Software Version | CONST | NO | String | VAR | String | 0.5.16 |
| 1010h | 0h | Save parameters | RO | NO | Uint8 | ARR | Uint8 | 4 |
| 1h | Save all object parameters | RW | NO | Uint32 | - | - | 0 |
| 2h | Save communication object parameters | RW | NO | Uint32 | - | - | 0 |
| 3h | Save application object parameters | RW | NO | Uint32 | - | - | 0 |
| 4h | Save sub-protocol area object parameters | RW | NO | Uint32 | - | - | 0 |
| 1011h | 0h | Restore default parameters | RW | NO | Uint8 | ARR | - | 4 |
| 1h | Restore default parameters for all objects | RW | NO | Uint32 | - | - | 0 |
| 2h | Restore default parameters of communication object | RW | NO | Uint32 | - | - | 0 |
| 3h | Restore application object parameters | RW | NO | Uint32 | - | - | 0 |
| 4h | Restore sub-protocol area object parameters | RW | NO | Uint32 | - | - | 0 |
| 1014h | - | Emergency Message COB-ID | RW | NO | Uint32 | VAR | Uint32 | 0x80+  NODEID |
| 1016h | 0h | Consumer heartbeat time | RO | NO | Uint8 | ARR | - | 8 |
| 1~8h | Consumer heartbeat time | RW | NO | Uint32 | - | Uint32 | 0 |
| 1017h | - | Producer heartbeat time | RW | NO | Uint16 | VAR | Uint16 | 500 |
| 1018h | 0h | Device Object Description | RO | NO | Uint8 | REC | - | 4 |
| 1h | Manufacturer ID | RO | NO | Uint32 | - | Uint32 | 0 |
| 2h | Device Code | RO | NO | Uint32 | - | Uint32 | 0 |
| 3h | Device Revision Number | RO | NO | Uint32 | - | Uint32 | 0 |
| 4h | Serial Number | RO | NO | Uint32 | - | Uint32 | 0 |
| 1200h | 0h | SDO Server Parameters | RO | NO | Uint8 | REC | Uint8 | 2 |
| 1h | Client to Server COB-ID | RO | NO | Uint32 | - | Uint32 | 0x600+  Node\_ID |
| 2h | Server to Client COB-ID | RO | NO | Uint32 | - | Uint32 | 0x580+  Node\_ID |
| 1400h | 0h | RPDO1 Parameters | RO | NO | Uint8 | REC | Uint8 | 4 |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x200  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0xFF |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 0 |
| 4h | Reserved Field | RW | NO | Uint8 | - | Uint8 | 0 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 0 |
| 1401h | 0h | RPDO2 Parameters | RW | NO | Uint8 | REC | Uint8 | - |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x300  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0xFF |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 0 |
| 4h | Reserved Field | RW | NO | Uint8 | - | Uint8 | 0 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 0 |
| 1402h | 0h | RPDO3 Parameters | RW | NO | Uint8 | REC | Uint8 | - |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x400  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0xFF |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 0 |
| 4h | Reserved Field | RW | NO | Uint8 | - | Uint8 | 0 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 0 |
| 1403h | 0h | RPDO4 Parameters | RW | NO | Uint8 | REC | Uint8 | - |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x500  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0xFF |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 0 |
| 4h | Reserved Field | RW | NO | Uint8 | - | Uint8 | 0 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 0 |
| 1600h | 0h | RPDO1 Mapping Parameters | RW | NO | Uint8 | REC | Uint8 | 3 |
| 1h | RPDO1 Mapping Object 1 | RW | NO | Uint32 | - | Uint32 | 0x60400010 |
| 2h | RPDO1 Mapping Object 2 | RW | NO | Uint32 | - | Uint32 | 0x60600008 |
| 3h | RPDO1 Mapping Object 3 | RW | NO | Uint32 | - | Uint32 | 0 |
| 1601h | 0h | RPDO2 Mapping Parameters | RW | NO | Uint8 | REC | Uint8 | 3 |
| 1h | RPDO2 Mapping Object 1 | RW | NO | Uint32 | - | Uint32 | 0x60400010 |
| 2h | RPDO2 Mapping Object 2 | RW | NO | Uint32 | - | Uint32 | 0x607A0020 |
| 3h | RPDO2 Mapping Object 3 | RW | NO | Uint32 | - | Uint32 | 0 |
| 1602h | 0h | RPDO3 Mapping Parameters | RW | NO | Uint8 | REC | Uint8 | 3 |
| 1h | RPDO3 Mapping Object 1 | RW | NO | Uint32 | - | Uint32 | 0x60400010 |
| 2h | RPDO3 Mapping Object 2 | RW | NO | Uint32 | - | Uint32 | 0x60FF0020 |
| 3h | RPDO3 Mapping Object 3 | RW | NO | Uint32 | - | Uint32 | 0 |
| 1603h | 0h | RPDO4 Mapping Parameters | RW | NO | Uint8 | REC | Uint8 | 3 |
| 1h | RPDO4 Mapping Object 1 | RW | NO | Uint32 | - | Uint32 | 0x60400010 |
| 2h | RPDO4 Mapping Object 2 | RW | NO | Uint32 | - | Uint32 | 0x60710010 |
| 3h | RPDO4 Mapping Object 3 | RW | NO | Uint32 | - | Uint32 | 0 |
| 1800h | 0h | TPDO1 Communication Parameters | RO | NO | Uint8 | REC | - | 5 |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x180  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0xFF |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 200 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 500 |
| 1801h | 0h | TPDO2 Communication Parameters | RO | NO | Uint8 | REC | - | 5 |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x280  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0xFF |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 200 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 500 |
| 1802h | 0h | TPDO3 Communication Parameters | RO | NO | Uint8 | REC | - | 5 |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x380  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0xFF |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 200 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 500 |
| 1803h | 0h | TPDO4 Communication Parameters | RO | NO | Uint8 | REC | - | 5 |
| 1h | COB-ID | RW | NO | Uint32 | - | Uint32 | 0x480  +Node\_ID |
| 2h | Transfer Type | RW | NO | Uint8 | - | Uint8 | 0x01 |
| 3h | Prohibited Time | RW | NO | Uint16 | - | Uint16 | 0 |
| 5h | Event Timer | RW | NO | Uint16 | - | Uint16 | 0 |
| 1A00h | 0h | TPDO1 Mapping Parameters | RO | NO | Uint8 | REC | Uint8 | 3 |
| 1h | Mapping Object 1 of TPDO1 | RW | NO | Uint32 | - | Uint32 | 0x60410010 |
| 2h | Mapping Object 2 of TPDO1 | RW | NO | Uint32 | - | Uint32 | 0x60610008 |
| 3h | Mapping Object 3 of TPDO1 | RW | NO | Uint32 | - | Uint32 | 0 |
| 1A01h | 0h | TPDO2 Mapping Parameters | RO | NO | Uint8 | REC | Uint8 | 3 |
| 1h | Mapping Object 1 of TPDO2 | RW | NO | Uint32 | - | Uint32 | 0x60410010 |
| 2h | Mapping Object 2 of TPDO2 | RW | NO | Uint32 | - | Uint32 | 0x60640020 |
| 3h | Mapping Object 3 of TPDO2 | RW | NO | Uint32 | - | Uint32 | 0 |
| 1A02h | 0h | TPDO3 Mapping Parameters | RO | NO | Uint8 | REC | Uint8 | 3 |
| 1h | Mapping Object 1 of TPDO3 | RW | NO | Uint32 | - | Uint32 | 0x60410010 |
| 2h | Mapping Object 2 of TPDO3 | RW | NO | Uint32 | - | Uint32 | 0x606C0020 |
| 3h | Mapping Object 3 of TPDO3 | RW | NO | Uint16 | - | Uint16 | 0 |
| 1A03h | 0h | TPDO4 Mapping Parameters | RO | NO | Uint8 | REC | Uint8 | 3 |
| 1h | Mapping Object 1 of TPDO4 | RW | NO | Uint32 | - | Uint32 | 0x60410010 |
| 2h | Mapping Object 2 of TPDO4 | RW | NO | Uint32 | - | Uint32 | 0x60770010 |
| 3h | Mapping Object 3 of TPDO4 | RW | NO | Uint32 | - | Uint32 | 0 |

**3.10.2 Object Group 2000h Allocation Overview**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 2000h | 0h | Motor Parameters | RO | NO | Uint8 | - | - | 13 |
| 1h | Number of pole pairs | RW | NO | Uint32 | - | - | 0 |
| 2h | Torque Constant | RW | NO | Float32 | - | - | 0 |
| 3h | Interphase Inductance | RW | NO | Float32 | - | - | 0 |
| 4h | Interphase resistance | RW | NO | Float32 | - | - | 0 |
| 5h | Current Loop Bandwidth | RW | NO | Uint16 | - | - | 0 |
| 6h | Motor Name | RW | NO | String | - | - | MW6010-8 |
| 7h | Maximum Temperature | RW | NO | Uint8 | - | - | 90 |
| 8h | Gear ratio | RW | NO | Float32 | - | - | 1 |
| 9h | Rotation Direction Selection | RW | TPDO | Uint8 | - | - | 0 |
| 0Ah | Can ID | RW | NO | Uint32 | - | - | 1 |
| 0Bh | CAN Baud Rate | RW | NO | Uint32 | kbps | - | 500 |
| 0Ch | Undefined | RW | NO | Uint8 | - | - | 0 |
| 0Dh | Undefined\_1 | RW | NO | Uint8 | - | - | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 2001h | 0h | Speed PID Control | RO | NO | Uint8 | - | - | 8 |
| 1h | kp | RW | TPDO | Float32 | - | - | 0 |
| 2h | ki | RW | TPDO | Float32 | - | - | 0 |
| 3h | kd | RW | TPDO | Float32 | - | - | 0 |
| 4h | Undefined | RW | NO | Uint8 | - | - | 0 |
| 5h | Undefined\_1 | RW | NO | Uint8 | - | - | 0 |
| 6h | Undefined\_2 | RW | NO | Uint8 | - | - | 0 |
| 7h | Undefined\_3 | RW | NO | Uint8 | - | - | 0 |
| 8h | Undefined\_4 | RW | NO | Uint8 | - | - | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 2002h | 0h | Position PID Control | RO | NO | Uint8 | - | - | 8 |
| 1h | kp | RW | TPDO | Float32 | - | - | 0 |
| 2h | ki | RW | TPDO | Float32 | - | - | 0 |
| 3h | kd | RW | TPDO | Float32 | - | - | 0 |
| 4h | Undefined | RW | NO | Uint8 | - | - | 0 |
| 5h | Undefined\_1 | RW | NO | Uint8 | - | - | 0 |
| 6h | Undefined\_2 | RW | NO | Uint8 | - | - | 0 |
| 7h | Undefined\_3 | RW | NO | Uint8 | - | - | 0 |
| 8h | Undefined\_4 | RW | NO | Uint8 | - | - | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 2003h | 0h | Torque PID Control | RO | NO | Uint8 | - | - | 8 |
| 1h | kp | RW | TPDO | Float32 | - | - | 0 |
| 2h | ki | RW | TPDO | Float32 | - | - | 0 |
| 3h | kd | RW | TPDO | Float32 | - | - | 0 |
| 4h | Current Loop Bandwidth | RW | NO | Uint16 | - | - | 0 |
| 5h | Undefined\_1 | RW | NO | Uint8 | - | - | 0 |
| 6h | Undefined\_2 | RW | NO | Uint8 | - | - | 0 |
| 7h | Undefined\_3 | RW | NO | Uint8 | - | - | 0 |
| 8h | Undefined\_4 | RW | NO | Uint8 | - | - | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 2004h | 0h | Encoder Parameters | RO | NO | Uint8 | - | - | 8 |
| 1h | Main Encoder Status | RW | TPDO | Uint32 | - | - | 0 |
| 2h | Absolute encoder rotation count data | RO | TPDO | Int32 | - | - | 0 |
| 3h | Undefined\_1 | RW | NO | Uint8 | - | - | 0 |
| 4h | Undefined\_2 | RW | NO | Uint8 | - | - | 0 |
| 5h | Undefined\_3 | RW | NO | Uint8 | - | - | 0 |
| 6h | Undefined\_4 | RW | NO | Uint8 | - | - | 0 |
| 7h | Undefined\_5 | RW | NO | Uint8 | - | - | 0 |
| 8h | Undefined\_6 | RW | NO | Uint8 | - | - | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 2005h | 0h | Second Encoder Parameter | RO | NO | Uint8 | - | - | 8 |
| 1h | Second Encoder State | RW | TPDO | Uint32 | - | - | 0 |
| 2h | Undefined | RW | NO | Uint8 | - | - | 0 |
| 3h | Undefined\_1 | RW | NO | Uint8 | - | - | 0 |
| 4h | Undefined\_2 | RW | NO | Uint8 | - | - | 0 |
| 5h | Undefined\_3 | RW | NO | Uint8 | - | - | 0 |
| 6h | Undefined\_4 | RW | NO | Uint8 | - | - | 0 |
| 7h | Undefined\_5 | RW | NO | Uint8 | - | - | 0 |
| 8h | Undefined\_6 | RW | NO | Uint8 | - | - | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 2006h | 0h | Temperature Parameter | RO | NO | Uint8 | - | - | 4 |
| 1h | Motor Temperature | RW | TPDO | Float32 | - | - | 0 |
| 2h | MOSFET Temperature | RW | TPDO | Float32 | - | - | 0 |
| 3h | Undefined | RW | NO | Float32 | - | - | 0 |
| 4h | Undefined\_1 | RW | NO | Float32 | - | - | 0 |

**3.10.3 Object Group 6000h Allocation Overview**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Subindex | Name | Accessibility | Can it be mapped? | Data Type | Unit | Data Range | Factory Settings |
| 6040h | - | Control Word | RW | YES | Uint16 | - | 0~65535 | 0 |
| 6041h | - | Status Word | RO | TPDO | Uint16 | - | 0~65535 | 0 |
| 605Ah | - | Quick Shutdown Mode Selection | RW | NO | Int16 | - | 0~7 | 0 |
| 605Dh | - | Shutdown Mode Selection | RW | NO | Int16 | - | 0~7 | 0 |
| 6060h | - | Mode Selection | RW | YES | Int8 | - | 0~10 | 0 |
| 6061h | - | Mode Display | RO | TPDO | Int8 | - | 0~10 | 0 |
| 6062h | - | User Location Instruction | RO | TPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 6064h | - | Current actual position | RO | TPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 6067h | - | Position reaches threshold | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 6069h | - | Actual value of speed sensor | RO | TPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 606Bh | - | User speed command | RO | TPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 606Ch | - | Current actual rotational speed | RO | TPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 6071h | - | Target Torque | RW | RPDO | INT16 | 0.1% rated torque | -215~(215-1) | 0 |
| 6072h | - | Maximum Torque | RW | RPDO | Uint16 | 0.1% rated torque | -10000~10000 | 10000 |
| 6073h | - | Maximum Current | RW | RPDO | Uint16 | 0.1% rated torque | 0~(216-1) | 0 |
| 6074h | - | User Torque Command | RO | TPDO | INT16 | 0.1% rated torque | -215~(215-1) | 0 |
| 6076h | - | Motor rated torque | RW | YES | Uint32 | 0.1% | 0~(232-1) | 5000 |
| 6077h | - | Current actual torque | RO | TPDO | INT16 | 0.1% rated torque | -215~(215-1) | 0 |
| 607Ah | - | Target Location | RW | RPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 607Ch | - | Origin Offset | RW | YES | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 607Dh | 0h | Software Location Restriction | RO | NO | Uint8 | - | - | 2 |
| 1h | Minimum Position Limit | RW | YES | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 2h | Maximum Position Limit | RW | YES | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 607Eh | - | Instruction Polarity | RW | YES | Uint8 | - | 0-255 | 0 |
| 607Fh | - | Maximum contour speed | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 6080h | - | Maximum Motor Speed | RW | RPDO | Uint32 | Instruction Unit | 0~(232-1) | 1000 |
| 6081h | - | Contour Velocity | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 6083h | - | Contour Acceleration | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 6084h | - | Profile deceleration | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 6085h | - | Quick stop deceleration | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 6086h | - | Motor operating curve type | RW | YES | Int16 | - | -231~(231-1) | 0 |
| 6087h | - | Torque Ramp | RW | RPDO | UINT32 | 0.1%/s rated torque | 0~(232-1) | 0 |
| 608Fh | 0h | Position Encoder Resolution | RO | NO | Uint8 | - | - | 2 |
| 1h | Encoder Resolution | RW | TPDO | Uint32 | Instruction Unit | -231~(231-1) | 16384 |
| 2h | Motor speed | RW | TPDO | Uint32 | Instruction Unit | -231~(231-1) | 1 |
| 6091h | 0h | Gear ratio | RO | NO | UNIT8 | - | - | 2 |
| 1h | Motor resolution | RW | RPDO | Uint32 | - | 1~(232-1) | 1 |
| 2h | Load axis resolution | RW | RPDO | Uint32 | - | 1~(232-1) | 1 |
| 6098h | - | Zero Return Mode | RW | YES | Int8 | - | 0~35 | 0 |
| 6099h | 0h | Zero return speed | RO | NO | UNIT8 | - | - | 2 |
| 1h | Search for the speed of the deceleration point signal | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 2h | Search zero signal speed | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 609Ah | - | Zero return acceleration | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 60B0h | - | Position Offset | RW | YES | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 60B1h | - | Speed Offset | RW | YES | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 60B2h | - | Torque Offset | RW | YES | Int16 | 0.1% rated torque | -215~(215-1) | 0 |
| 60C2h | 0h | Interpolation Time | RO | NO | Uint8 | - | - | 2 |
| 1h | Interpolation time unit | RW | YES | Uint8 | - | 0~255 | 1 |
| 2h | Interpolation Time Index | RW | YES | Int8 | - | -128~63 | 0 |
| 60C5h | - | Maximum contour acceleration | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 60C6h | - | Maximum contour deceleration | RW | YES | Uint32 | Instruction Unit | 0~(232-1) | 0 |
| 60F4h | - | User location deviation | RO | TPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 60F6h | 0h | Current Loop PID Parameters | RO | NO | Uint8 | - | - | 0 |
|  | 1h | Current regulator kp | RW | RPDO | int16 | - | - | 0 |
|  | 2h | Current regulator ki | RW | RPDO | int16 | - | - | 0 |
| 60FCh | - | Motor Position Command\* | RO | TPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 60FFh | - | Target Speed | RW | RPDO | Int32 | Instruction Unit | -231~(231-1) | 0 |
| 6502h | - | Supported control modes | RO | NO | UINT32 | - | - | 0x761 |

3.12 **Correspondence Table between Object Dictionary and Motor Parameters**

For the explanation of instruction units, please refer to  [the conversion factor section of the Supplementary Manual for Guardian Beast Drive CANOpen](https://bcnyljrhe70u.feishu.cn/wiki/RkxswgglNiX2mckH9CdcVNmQnXe) .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | Subindex | Type | Unit | corresponding motor parameters |
| 6040h | - | Uint16 | - | odrv0.axis0.requested\_state = 8  The servo operating state corresponds to the closed-loop control mode of the motor sprite,  odrv0.axis0.requested\_state = 1  Other states in the servo finite-state machine correspond to the idle state of the motor sprite |
| 6041h | - | Uint16 | - | Same as above  In contour position mode, bit 10 of the status word indicates whether the target position has been reached  In homing mode, bits 10 and 12 of the status word indicate whether the set origin position has been reached |
| 6060h | - | Int8 | - | odrv0.axis0.controller.config.control\_mode  odrv0.axis0.controller.config.input\_mode  For specific correspondence, please refer to [the operation mode 6060 of the Supplementary Manual for Guardian Beast Drive CANOpen](https://bcnyljrhe70u.feishu.cn/wiki/RkxswgglNiX2mckH9CdcVNmQnXe). |
| 6061h | - | Int8 | - | Same as above |
| 6098h | - | Int8 | - | Currently, the homing method supports 17 and 35 |
| 6064h | - | int32 | Instruction Unit | odrv0.axis0.encoder.pos\_estimate \* conversion factor  Current position value, for specific usage, please refer to [the conversion factor section of the Supplementary Manual for Guardian Beast Drive CANOpen](https://bcnyljrhe70u.feishu.cn/wiki/RkxswgglNiX2mckH9CdcVNmQnXe). |
| 607Ah | - | int32 | Instruction Unit | odrv0.axis0.controller.input\_pos  Input Position |
| 60FFh | - | int32 | Instruction Unit | odrv0.axis0.controller.input\_vel  Input speed |
| 6071h | - | int16 | 0.001 Nm \* rated torque | odrv0.axis0.controller.input\_torque  Input Torque |
| 6076h | - | Uint32 | 0.1% | The rated torque of the motor is defaulted to 5NM |
| 6077h | - | int16 | Instruction Unit | odrv0.axis0.motor.current\_control.Iq\_measured\* axis.motor\_.config\_.torque\_constant \* axis.motor\_.config\_.gear\_ratio  Motor-side torque |
| 606Ch | - | int32 | Instruction Unit | odrv0.axis0.encoder.vel\_estimate\_counts  Current Rotational Speed |
| 608Fh | 1h | Uint32 | Encoder Resolution | is equivalent to the conversion factor, and for specific usage, please refer to  [the conversion factor section of the Supplementary Manual for Guardian Beast Drive CANOpen](https://bcnyljrhe70u.feishu.cn/wiki/RkxswgglNiX2mckH9CdcVNmQnXe) . |
| 6081h | - | Uint32 | Instruction Unit | odrv0.axis0.trap\_traj.config.vel\_limit  Speed limit for trapezoidal position control. |
| 6083h | - | Uint32 | Instruction Unit | odrv0.axis0.trap\_traj.config.accel\_limit  Acceleration limit for trapezoidal position control. |
| 6084h | - | Uint32 | Instruction Unit | odrv0.axis0.trap\_traj.config.decel\_limit  Deceleration limit for trapezoidal position control. |
| 607Ch | - | Int32 | Instruction Unit | odrv0.axis0.min\_endstop.config\_.offset  Origin Offset |
| 6099H | 1h | Uint32 | Instruction Unit | odrv0.axis0.controller.config.homing\_speed  Search return-to-zero signal speed |
| 609Ah | - | Uint32 | Instruction Unit | odrv0.axis0.controller.config.vel\_ramp\_rate  Search for the acceleration of the zero return signal |
| 2003h | 4h | Uint16 | - | odrv0.axis0.motor.config.current\_control\_bandwidth  Current Loop Bandwidth |
| 2001h | 1h | Float32 | - | odrv0.axis0.controller.config.vel\_gain  Speed loop kp |
| 2001h | 2h | Float32 | - | odrv0.axis0.controller.config.vel\_integrator\_gain  Speed loop ki |
| 2001h | 3h | Float32 | - | odrv0.axis0.controller.config.vel\_diff\_gain  Speed loop kd |
| 2002h | 1h | Float32 | - | odrv0.axis0.controller.config.pos\_gain  Position loop kp |
| 2002h | 2h | Float32 | - | odrv0.axis0.controller.config.pos\_integrator\_gain  Position loop ki |
| 2002h | 3h | Float32 | - | odrv0.axis0.controller.config.pos\_diff\_gain  Position loop kd |

Note: The detailed explanation of motor parameters is in the [User Manual](https://cyberbeast.feishu.cn/docx/N3SMd4QyRobzHkx3wP3cT1qXnpf)

4. **Exception code (to be updated)**

4.1 **Exception Code**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | 错误类别 | 错误码 | odrivetool 显示 | 描述 | | 系统异常 | 0x00000002 | DC\_BUS\_UNDER\_VOLTAGE | 电源电压过低 | | 0x00000004 | DC\_BUS\_OVER\_VOLTAGE | 电源电压过高 | | 0x00000008 | DC\_BUS\_OVER\_REGEN\_CURRENT | 电源反向（充电）电流过高 | | 0x00000010 | DC\_BUS\_OVER\_CURRENT | 电源正向（放电）电流过高 | | 驱动异常 | 0x00000001 | INVALID\_STATE | 驱动器状态无效 | | 0x00000040 | MOTOR\_FAILED | 电机异常 | | 0x00000100 | ENCODER\_FAILED | 编码器异常 | | 0x00000200 | CONTROLLER\_FAILED | 控制器异常 | | 0x00001000 | MIN\_ENDSTOP\_PRESSED | 低限位触发 | | 0x00002000 | MAX\_ENDSTOP\_PRESSED | 高限位触发 | | 0x00004000 | ESTOP\_REQUESTED | 紧急停止 | | 0x00020000 | HOMING\_WITHOUT\_ENDSTOP | 回零但没有限位开关 | | 0x00080000 | UNKNOWN\_POSITION | 无位置信息 | | 电机异常 | 0x00000001 | PHASE\_RESISTANCE\_OUT\_OF\_RANGE | 相间电阻超出正常范围 | | 0x00000002 | PHASE\_INDUCTANCE\_OUT\_OF\_RANGE | 相间电感超出正常范围 | | 0x00000010 | CONTROL\_DEADLINE\_MISSED | FOC 频率太高 | | 0x00000080 | MODULATION\_MAGNITUDE | SVM 调制异常 | | 0x00000400 | CURRENT\_SENSE\_SATURATION | 相电流饱和 | | 0x00001000 | CURRENT\_LIMIT\_VIOLATION | 电机电流过大 | | 0x00020000 | MOTOR\_THERMISTOR\_OVER\_TEMP | 电机温度过高 | | 0x00040000 | FET\_THERMISTOR\_OVER\_TEMP | 驱动器温度过高 | | 0x00080000 | TIMER\_UPDATE\_MISSED | FOC 处理不及时 | | 0x00100000 | CURRENT\_MEASUREMENT\_UNAVAILABLE | 相电流采样丢失 | | 0x00200000 | CONTROLLER\_FAILED | 控制异常 | | 0x00400000 | I\_BUS\_OUT\_OF\_RANGE | 母线电流超限 | | 0x00800000 | BRAKE\_RESISTOR\_DISARMED | 刹车电阻驱动异常 | | 0x01000000 | SYSTEM\_LEVEL | 系统级异常 | | 0x02000000 | BAD\_TIMING | 相电流采样不及时 | | 0x04000000 | UNKNOWN\_PHASE\_ESTIMATE | 电机位置未知 | | 0x08000000 | UNKNOWN\_PHASE\_VEL | 电机速度未知 | | 0x10000000 | UNKNOWN\_TORQUE | 力矩未知 | | 0x20000000 | UNKNOWN\_CURRENT\_COMMAND | 力矩控制未知 | | 0x40000000 | UNKNOWN\_CURRENT\_MEASUREMENT | 电流采样值未知 | | 0x80000000 | UNKNOWN\_VBUS\_VOLTAGE | 电压采样值未知 | | 0x100000000 | UNKNOWN\_VOLTAGE\_COMMAND | 电压控制未知 | | 0x200000000 | UNKNOWN\_GAINS | 电流环增益未知 | | 0x400000000 | CONTROLLER\_INITIALIZING | 控制器初始化异常 | | 0x800000000 | UNBALANCED\_PHASES | 三相不平衡 | | 控制异常 | 0x00000001 | OVERSPEED | 速度过高 | | 0x00000002 | INVALID\_INPUT\_MODE | 控制输入模式不正确 | | 0x00000004 | UNSTABLE\_GAIN | 锁相环增益不稳 | | 0x00000020 | INVALID\_ESTIMATE | 位置/速度不稳定 | | 0x00000080 | SPINOUT\_DETECTED | 机械功率和电气功率不匹配（编码器校准不正确，或磁钢不稳） | | 编码器异常 | 0x00000001 | UNSTABLE\_GAIN | 编码器带宽过高 | | 0x00000002 | CPR\_POLEPAIRE\_MISMATCH | CPR 和极对数不匹配 | | 0x00000004 | NO\_RESPONSE | 编码器无响应 | | 0x00000400 | SEC\_ENC\_COM\_FAIL | 第二编码器通信错误 |   同步自文档: <https://cyberbeast.feishu.cn/docx/BPnQd8reEotLWVxqHFNc9qYZnKh#BI4edAaE7sEYq7bAq6Bcg1qEnfg> |