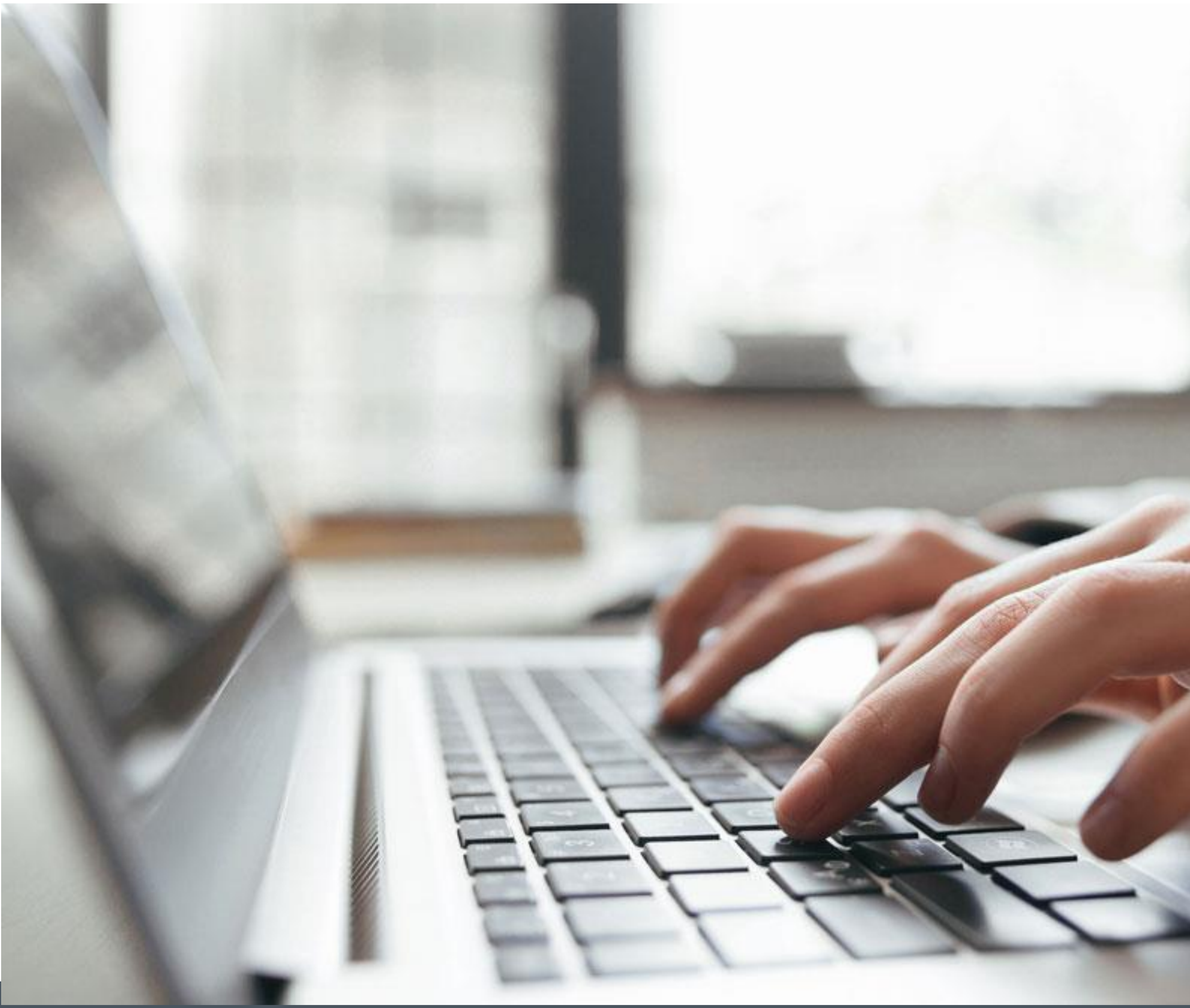

Application Note

How to use Modbus registers in Nanotec controllers

Version 1.0.2

www.nanotec.com

Contents

1 Intended use and audience

Nanotec controllers use CANopen protocol and thus need correct object mapping for Modbus devices. By default mapping and example code, this application note shows you how to access your Modbus messages in their registers and how to align the object mappings of Modbus and CANopen. It applies to Nanotec products of the following data transfer standards:

- Modbus RTU
- Modbus TCP

All products required in this document are for use by trained experts only. Before product use, please ensure that all users read, understand and follow the instructions in this document fully.

2 Requirements

NOTICE

Malfunction from incompatibility! Products used here come in various versions. You must ensure proper operation of motor and slave drive before you can use our example projects.

► Use compatible equipment only.
► Configure the slave drive (= Nanotec controller) in advance.
► Keep slave / controller free from stand-alone program runs (= NanoJ etc.).
► Follow valid OEM instructions.

3 Mapping

Modbus works with a Mailbox system. By Modbus message, you can access a defined register range (read = *5000+;* write = *6000+*), with no direct reference to the drive's CANopen-based object dictionary.

In so-called mapping, you must assign our objects to registers 5000+ / 6000+. You can do so for object 0x3502 and 0x3602, with our Plug & Drive Studio software. By standard Modbus function, you can access **mapped** objects only. For **direct** object access, in contrast, use the encapsulated mode [function code 43 (0x2B)].

The Plug & Drive interface (PDI) for Nanotec-specific drive control is an alternative to the device profile as defined by CANopen Standard CiA 402. By PDI, you can trigger drive commands instantly without state machine run. **Note**: PDI comes only in firmware version FIR-v18xx and higher. Older firmware needs standard PDO mapping.

3.1 How to align Modbus registers to CANopen

Modbus registers have always 16 bits, whereas CANopen objects have 8, 16, or 32. We must consider this in mapping and duly align Modbus registers / CANopen objects.

For 32-bit CANopen objects, we use *two* Modbus registers (= *one* per 16). Whereas for 8-bit objects, we may fill the map with functionless dummy-object entries (= index, data type):

0002h Signed integer (8 bit)	0003h Signed integer (16 bit)	0004h Signed integer (32 bit)
0005h Unsigned integer (8 bit)	0006h Unsigned integer (16 bit)	0007h Unsigned integer (32 bit)

3.2 Default mapping

The Nanotec drive has a preconfigured default mapping for both standard PDO objects and PDI:

Mapping	**Nanotec CANopen object**	**Modbus address**
Standard Tx PDO 0x3602:xx		
0x3602:0x01	0x6041 *Statusword*	5000
0x3602:0x02	0x0005 *Dummy object*	5001 high byte
0x3602:0x03	0x6061 *Modes-of-operation display*	5001 low byte
0x3602:0x04	0x6064 *Position actual value*	5002, 5003
0x3602:0x05	0x6044 *Vl velocity actual value*	5004
0x3602:0x06	0x60FD *Digital inputs*	5005, 5006
Tx PDO for PDI	0x2292:01 *PDI status*	4996
	0x603F *Error code*	4997
	0x2292:02 *PDI return value*	4998, 4999
Standard Rx PDO 0x3502:xx		
0x3502:0x01	0x6040 *Controlword*	6000
0x3502:0x02	0x0005 *Dummy object*	6001 high byte
0x3502:0x03	0x6060 *Modes of operation*	6001 low byte
0x3502:0x04	0x607A *Target position*	6002, 6003
0x3502:0x05	0x6081 *Profile velocity*	6004, 6005
0x3502:0x06	0x6042 *Vl target velocity*	6006
0x3502:0x07	0x60FE:01 *Digital outputs*	6007, 6008
Rx PDO for PDI	0x2291:01 *PDI set value 1*	5996, 5997
	0x2291:02 *PDI set value 2*	5998
	0x2291:03 *PDI set value 3*	5999 high byte
	0x2291:04 *PDI command*	5999 low byte

Note: Older firmware versions (FIR-v2039 and lower) have a slightly different default Rx PDO mapping. There, the object 0x3202 is mapped in 0x3502:04 so that it occupied the registers 6002 and 6003. All subsequent entries used to be located both one subindex and two registers higher.

3.3 How to change the mapping

You **can't** change PDI mappings. For all other, you must adjust objects 0x3502:xx / 0x3602:xx:

1. Disable mapping; set subindex 0x00 to value *0*.
2. Adjust the mapped objects by syntax *0xIIIISSLL*
   I = index; *S* = subindex; *L* = length of object to be mapped.
3. Enable mapping; set subindex 0x00 to the number of mapped subindices.
4. Store all for a retain after restart.

Example: We want to add object 0x2039:03 to the Tx-PDO map (phase A for current; phase U for BLDC). Thus, via *0x3602:0x00 = 0x00,* we deactivate mapping. Objects already ex works are:

0x3602:0x01	0x604100***10*** (object 6041h:00h) ***16*** mapped bits long
0x3602:0x02	0x000500***08*** (dummy object 0005h:00h) ***8*** mapped bits long
0x3602:0x03	0x606100***08*** (object 6061h:00h) ***8*** mapped bits long
0x3602:0x04	0x606400***20*** (object 6064h:00h) ***32*** mapped bits long
0x3602:0x05	0x604400***10*** (object 6044h:00h) ***16*** mapped bits long
0x3602:0x06	0x60FD00***20*** (object 60FDh:00h) ***32*** mapped bits long

1 For new entry: Add 0x3602:0x07 = 0x20390120 (object 2039h:01h); 32 (20h) mapped bits long.
2 Set the total of mapped objects to **seven**: 0x3602:0x00 = 0x**07**.
3 You now reach the new entry by Modbus registers 5007 and 5008.

4 Examples for motor start-up

To start the motor, **do** address the objects for one of the following modes:

4.1 Velocity mode

Object	0x6060 = 2	0x6040 = 6	0x6040 = 7	0x6040 = 15
Modbus register address	6001 = 2	6000 = 6	6000 = 7	6000 = 15

The motor now runs at the speed written to object 0x6042. You can adjust it on the fly, say, to 100 (default unit rpm). In that case, object **0x6042 = 100** needs Modbus address **6006 = 100**.

4.2 Profile position

Object	0x6060 = 1	0x6040 = 6	0x6040 = 7	0x6040 = 15
Modbus register address	6001 = 1	6000 = 6	6000 = 7	6000 = 15

The motor, now powered, should hold its position. Adjust the motion parameters as needed, say, to position *10000* (default unit depends on firmware version; *2000 incr./rev* or *1/10°*) and speed *500* (default: rpm).

0x6081 = 500	6004 = 0, and 6005 = 500
0x607A = 10000	6002 = 0, and 6003 = 10000

To start absolute positioning motion, set a rising edge to controlword bit 4:

0x6040 = 31	6000 = 31

For a new motion, reset – and again set – controlword bit 4 (for relative positioning, also set bit 6):

0x6040 = 95	6000 = 95

5 Liability

This document builds on our experience with typical requirements in a wide range of industrial applications. Still, we assume no liability for correctness and completeness.

Document contents are subject to change without notice and serve for general guidance only. None is to be construed as a guarantee for applicability to all scenarios without additional client-side tests and, if necessary, modifications.

Neither does this document replace datasheets and other product documentation. For their latest versions please visit our website at www.nanotec.com.

The customer alone must self-responsibly evaluate, investigate and decide for which application this document is valid / suitable / usable, or not.

No warranty exists for defects due to improper device / module handling. In no case is Nanotec liable for direct, indirect, incidental or consequential damages in connection with this document.

In addition, the liability rules of our Terms and Conditions of Sale and Delivery shall apply.

6 Imprint

Nanotec Electronic GmbH & Co. KG │ Kapellenstraße 6 │ 85622 Feldkirchen │ Germany
Tel. +49 (0)89 900 686-0 │ Fax +49 (0)89 900 686-50 │ info@nanotec.de │ www.nanotec.com