User Manual PNDS3

For Plug & Drive Studio 3 Version 1.6.0 User Manual Version: 1.2.0

Contents

1 Document aim and conventions.. 3

2 For your safety..5

2.1 Warn and risk levels...5

3 Before you start.. 6

3.1 System and hardware requirements...6

3.2 Intended use and audience...6

3.3 Delivery scope and warranty...6

4 Your product.. 8

5 Installation and adapter..10

6 User interface (UI)...11

6.1 Header (1)...12

6.2 Project bar (4)..13

6.3 Work desk (3)... 17

6.4 Display wall (2).. 19

7 Project setup... 21

8 Special controls.. 24

8.1 Complex controls.. 24

8.2 Device communication..26

9 Oscilloscope... 27

10 Programming with *NanoJ*... 28

10.1 NanoJ program... 28

10.2 Mapping in the NanoJ program..31

10.3 NanoJ functions in the NanoJ program...33

10.4 Restrictions and possible problems..34

11 NanoJ App.. 35

12 Imprint, versions.. 36

1 Document aim and conventions

Beside technical data, this document explains product use and function. For possible combination with other Nanotec products, please ask your Nanotec sales partner. Before using the product, please note document font styles and conventions.

Underlined text marks a cross reference or hyperlink.

Example 1: Observe our safety notes.

Example 2: Download needed code templates from our website for EMEA / APAC or AMERICA.

Gray bold italics call out **menu paths, buttons, tab** and **file names**.

Example 1: Select **Home > Connect controller > CANopen**.

Example 2: In the **NanoJ** tab, select **NanoJ project** and open **Analog Input.cpp**.

Plain italics mark *Freehand entries* and *foreign-language* expressions. They also emphasize words of critical weight. Alternatively, bracketed exclaim marks(!) give critical weight.

Example 1: Enter *Plug & Drive Studio*. In addition to users (= *Nutzer; usuario; utente; utilisateur; utente* etc.), this document also addresses:

- Third-party users (= *Drittnutzer; tercero usuario; terceiro utente; tiers utilisateur; terzo utente* etc.).

- End users (= *Endnutzer; usuario final; utente final; utilisateur final; utente finale* etc.).

Example 2: Protect yourself, others and your equipment. Follow our *general* safety notes that are generally applicable to *all* Nanotec products. Also follow the *specific* safety notes that apply to *this* specific product.

`Courier` marks `code blocks` or `programming commands`.

Example 1: Via Bash, call `sudo make install` to copy shared objects; then call `ldconfig`.

Example 2: Use the following NanoLibAccessor function to change the logging level in NanoLib:

```
//
        ***** C++ variant *****
 void setLoggingLevel(LogLevel level);
```

The verb *to co-click*

Co-clicking means a mouse click by secondary key to open context menus etc.

Example 1: Co-click the file, select **Rename**, and rename the file.

Example 2: Co-click the file to check and select **Properties**.

Numerical values

Numbers appear in decimal. Hexadecimal notation ends in subscript *h.* Objects in the object dictionary notate in hexadecimal as `<Index>:<Subindex>`, non-notated subindices as `00`$_h$. Example: `1003`$_h$`:05`$_h$ is subindex 5 in object 1003$_h$. And `6040`$_h$ is subindex 00 in object 6040$_h$.

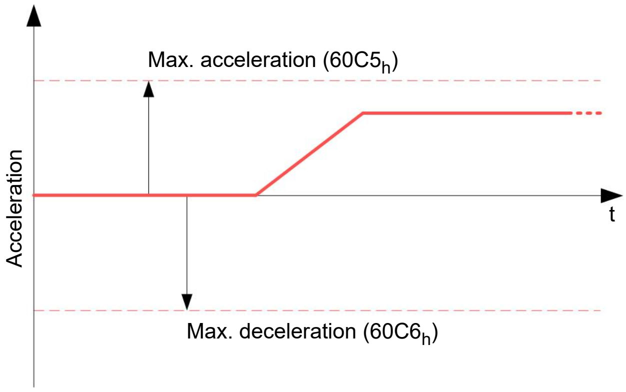
Bits

Each object bit counts up from LSB (bit number 0), such as data type *UNSIGNED8*:

	MSB							LSB	
Bit Nummer	7	6	5	4	3	2	1	0	
Bits	0	1	0	1	0	1	0	1	≙ 55_{hex} ≙ 85_{dec}

Count direction (arrows)

Illustrations always count arrow-wards; both example objects `60C5`$_h$ and `60C6`$_h$ are thus positive.



2 For your safety

Before product use, please ensure that all users read, understand and follow the instructions in this document fully.

2.1 Warn and risk levels

Please note: our hazard warnings, alert symbols and signal words mark different risk levels.

CAUTION!

CAUTION warns of *possible physical danger!* Minor / moderate injury *possible!*

►Instruction against **unhealthy** user errors.

►Alarm color #FFD100.

NOTICE

A NOTICE warns of wrong operation. Material / Ecological damage possible.

►Instruction against **destructive** user errors (= mere property risks).

►Alarm color #0070CD.

Note: Explains or simplifies a process by additional information.

3 Before you start

Before product use, you need to prepare the PC and verify product intent / limits. Via online help, you can learn how to install and set up projects and how PNDS3 runs. Observe the safety notes in the manual (www.nanotec.com).

3.1 System and hardware requirements

Plug & Drive Studio 3 (PNDS3) needs 64-bit operating systems. Nanotec recommends controller firmware *FIR-v2213* or newer. PNDS3 offers a special control for firmware update.

PNDS3

v1.6.0

64-bit OS requirements

- Windows 10
- .NET Framework 8
- Display resolution 1920x1080

Fieldbus adapters / cables

- **CANopen:**
  - □ IXXAT USB-to-CAN V2
  - □ Nanotec ZK-USB-CAN-1
- **Modbus RTU:**
  - □ Nanotec ZK-USB-RS485-1 or equivalent USB-RS485 adapter
  - □ USB cable via virtual comport (VCP)
- **Ethernet (REST) , EtherCAT, Profinet:**
  - □ suitable ethernet cable
  - □ WinPcap 4.1.3, or Npcap installation, see Installation and adapter

3.2 Intended use and audience

NOTICE

Damage: from unskilled staff!

▶Use the product only for the purpose described in this document.

▶Restrict use to expert staff only.

▶Follow valid OEM and system prescriptions for all equipment involved.

Plug & Drive Studio 3 (PNDS3) is a free software for easy Nanotec drive commissioning. The underlying operating system / hardware (PC) is **not** real-time capable. **Never** use PNDS3 for time-critical or synchronous multi-axis motion **nor** integrate it as a safety component in a product or system.

Add proper warnings and instructions for safe use / operation to each end user product with a Nanotec-produced component. Submit any Nanotec warning directly to end users. The product addresses skilled experts in industry use cases alone. Expert means:

- Training / experience in motor and controller handling
- Understanding this document plus Nanotec drive manuals
- Knowledge of valid regulations

3.3 Delivery scope and warranty

PNDS3 comes as a **.zip* folder from our download website for either EMEA / APAC or AMERICA. Duly store and unzip your download before setup. The product package contains:

- Software as an executable file
- Current firmware release

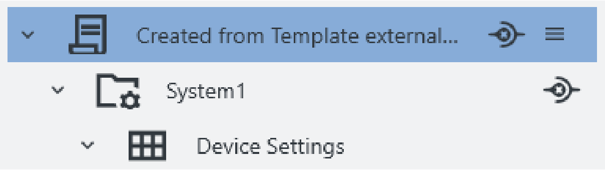
- Project templates
- Online help file

For scope of warranty, please observe our terms and conditions for either EMEA / APAC or AMERICA. **Note:** Nanotec is not liable for wrong quality, handling, installation, operation, use, and maintenance of third-party equipment! Follow valid OEM instructions.

4 Your product

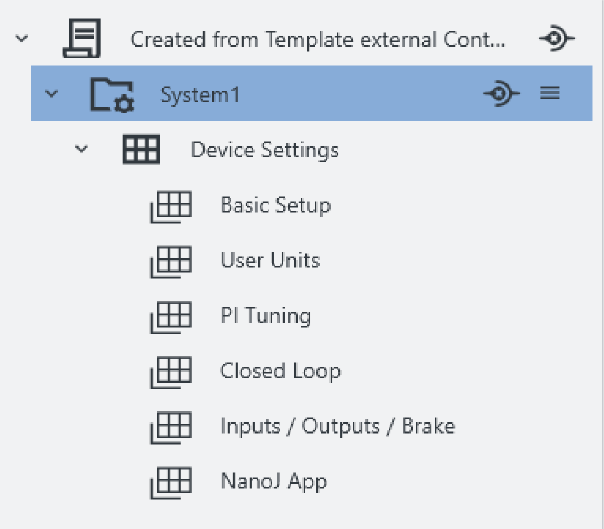
With PNDS3, you parametrize and commission Nanotec drives. Using templates for various Nanotec drives, you can add your own projects, systems and modules to the modular user interface. The software comes with a default folder structure (*Project, System, Module Group, Module,* etc.).

Project


Created from Template external...
System1
Device Settings

You manage all settings and device parameters in projects, save these as a file and im- / export them, say, as a template. Such a reusable **Project** can have multiple systems, say, the axes of a machine.

System

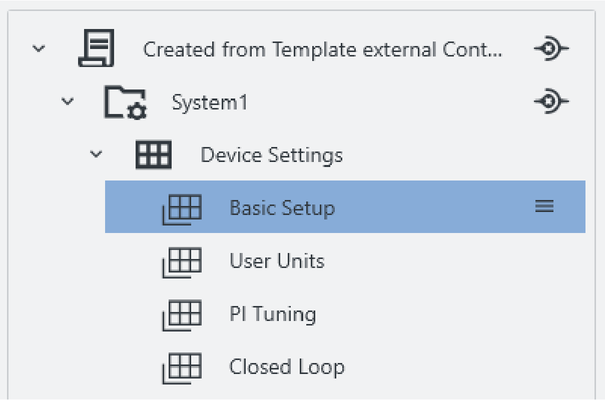

Created from Template external Cont...
System1
Device Settings
Basic Setup
User Units
PI Tuning
Closed Loop
Inputs / Outputs / Brake
NanoJ App

In a project (here: external controller), you create and store drive systems (here: X-axis). Each is im- / exportable as template.

You can extend such a reusable **System**, of at least motor and controller, by modules or module groups for encoder, gearbox, brake, settings, parameters, etc.

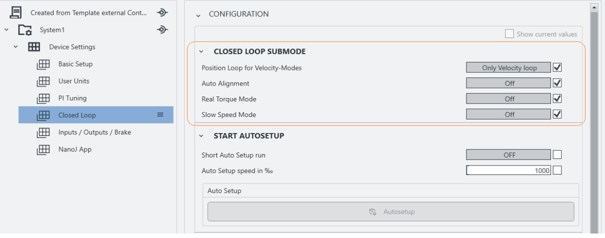
By parameters, sortable / poolable into several modules or module groups, you quickly control all system elements.

Module (Group)


Created from Template external Cont...
System1
Device Settings
Basic Setup
User Units
PI Tuning
Closed Loop

A module (group) contains parameters or controls (groups) and is im- / exportable, single or grouped, as template.

Controls Group



Created from Template external Cont...
System1
Device Settings
Basic Setup
User Units
PI Tuning
Closed Loop
Inputs / Outputs / Brake
NanoJ App

CONFIGURATION

□ Show current values

CLOSED LOOP SUBMODE

Position Loop for Velocity-Modes: Only Velocity loop ☑
Auto Alignment: Off ☑
Real Torque Mode: Off ☑
Slow Speed Mode: Off ☑

START AUTOSETUP

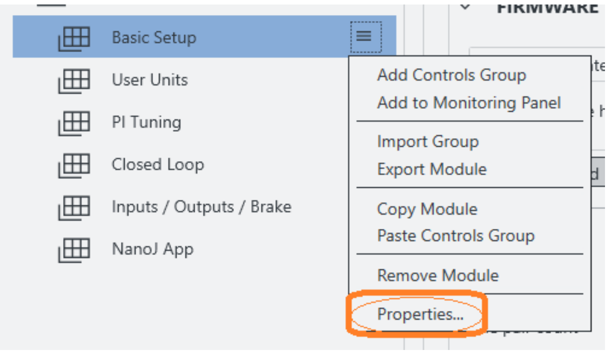
Short Auto Setup run: OFF □
Auto Setup speed in ‰: 1000 □

Auto Setup

A **Controls group** pools single device parameters (objects from the dictionary in the controller) and / or **Special controls**.

You im- / export such a control group together with set values, say, as template.

Property editing



Simply co-click an element, select **Properties**, insert a visible name, version number, and description: This way you create your individual user interface.

5 Installation and adapter

Install the software, set up the adapter – and PNDS3 is ready to go. You find PNDS3 software online as a zip download.

1. Open the website **Nanotec > Products > Software > Plug & Drive Studio 3**.
2. Download and extract the product zip file.
3. Run the executable file **PNDS3.exe** or use **setup.exe** resp. the **installer** to start the installer.
4. Only with PNDS3 installed: Prepare your fieldbus adapter (see below).

CANopen

1. Decide: **Ixxat USB-to-CAN**? Or **Nanotec ZK-USB-CAN-1**?
2. For **Ixxat USB-to-CAN:** Download the driver (www.ixxat.com/); install it by hand.
3. Connect the adapter to the computer. For **Nanotec ZK-USB-CAN-1:** Wait for self-installation.
4. Via correct cable (see product manual): Connect the installed adapter to the controller.

USB: Nanotec Virtual COM-Port (VCP)

1. Connect the voltage supply to the controller and switch it on.
2. Via correct USB cable: Connect the PC to the controller (= "mass storage device").
3. In Explorer > Controller directory: Select `cfg.txt` (= `pd4ccfg.txt`for a PD4C).
4. Open the file via text editor (Notepad etc.).
5. Add the lines `2102|=0x100000` and `4015:01=0`. Save the file.
6. Restart the controller and check if its COM port appears in the device manager.

Modbus RTU

1. For **Nanotec ZK-USB-RS485-1**: Connect the adapter to the computer and wait for self-installation.
2. For **other equivalent adapters**: Follow valid OEM instructions to install the driver.

EtherCAT

Install WinPcap 4.1.3 or Npcap and make sure the corresponding driver is activated for the designated ethernet adapter.

Profinet

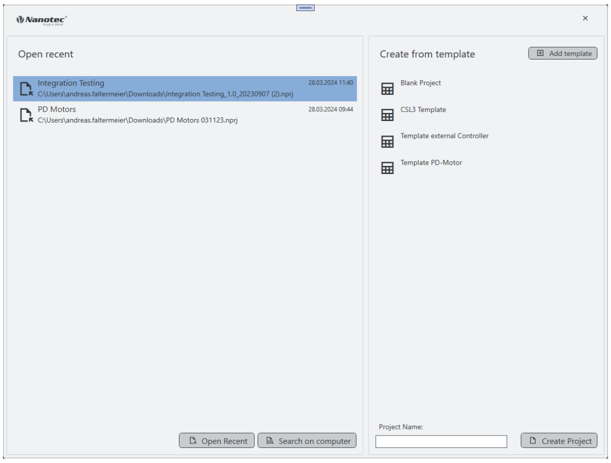
1. Install Win10cap or Npcap and make sure the corresponding driver is activated for the designated ethernet adapter.
2. Configure the IP address of the drive and the ethernet adapter accordingly, as described in the drive manual.

Ethernet (REST)

Configure the IP address of the drive and the ethernet adapter accordingly, as described in the drive manual.

6 User interface (UI)

Thanks to flexible areas and windows, fitted into the main window or usable stand-alone, you can master a wide range of tasks. Before product use, please understand the UI structure.

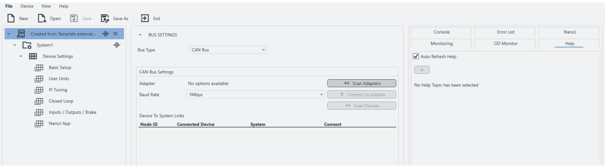


Integration Testing 28.03.2024 11:40
C:\Users\andreas.faltermeier\Downloads\Integration Testing_1.0_20230907 (2).nprj

PD Motors 28.03.2024 09:44
C:\Users\andreas.faltermeier\Downloads\PD Motors 031123.nprj

Project Name:

When PNDS3 starts for the first time, you are asked to create a new project: either a blank one or one based on a template.



Bus Type: CAN Bus

CAN Bus Settings

Adapter: No options available

Baud Rate: 1Mbps

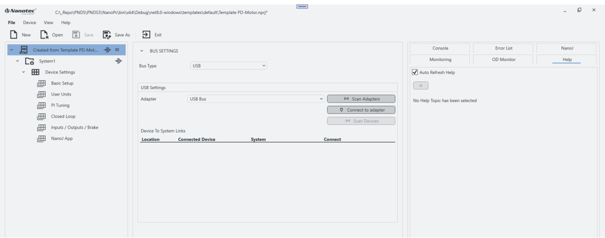
Device To System Links

Node ID	Connected Device	System	Connect

☑ Auto Refresh Help

No Help Topic has been selected

If you open a blank project, the project tree is empty at first.



C:_Repo\PNDS\PNDS3\NanoPc\bin\x64\Debug\net8.0-windows\templates\default\Template PD-Motor.nprj*

Bus Type: USB

USB Settings

Adapter: USB Bus

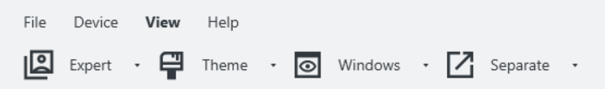
Device To System Links

Location	Connected Device	System	Connect

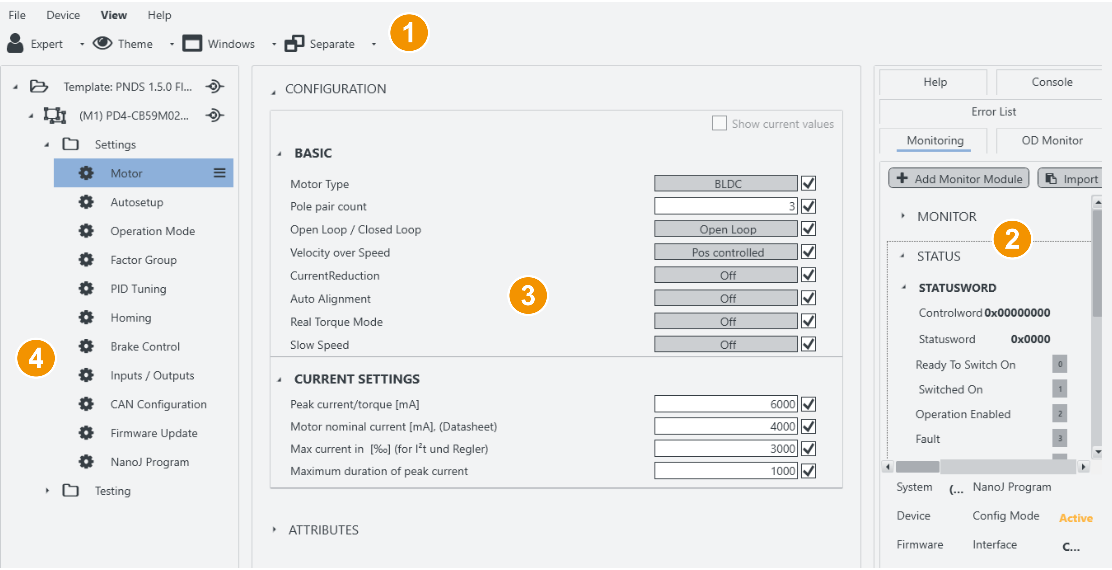
☑ Auto Refresh Help

No Help Topic has been selected

If you create a new project or load a template, the interface fills up according to your needs. This way, you design your own UI.



Using the **View** options in the main menu you can further customize the UI by changing the theme, showing/hiding features or opening them in separate windows.



BASIC

Motor Type	BLDC	☑
Pole pair count	3	☑
Open Loop / Closed Loop	Open Loop	☑
Velocity over Speed	Pos controlled	☑
CurrentReduction	Off	☑
Auto Alignment	Off	☑
Real Torque Mode	Off	☑
Slow Speed	Off	☑

CURRENT SETTINGS

Peak current/torque [mA]	6000	☑
Motor nominal current [mA], (Datasheet)	4000	☑
Max current in [‰] (for I²t und Regler)	3000	☑
Maximum duration of peak current	1000	☑

STATUSWORD

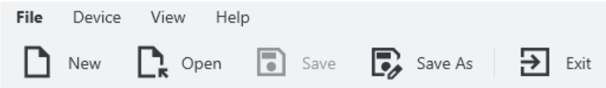
Controlword	0x00000000
Statusword	0x0000
Ready To Switch On	0
Switched On	1
Operation Enabled	2
Fault	3

- Header for main menu (1).
- Display wall (2) for monitoring, object directory, help etc.
- Work desk (3) for user controls etc.
- Project (or side) bar (4) for systems etc.

6.1 Header (1)

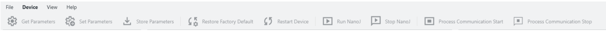
As a prominent layout bracket on top in the user interface, the UI header contains all basic functions and commands relevant to projects, devices and the UI view.

File



Leftmost above the header, you find the main menu for project files. You can load new – and save, reopen, edit existing projects.

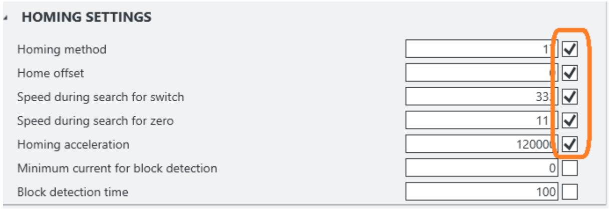
Device



Read, write, and save device parameters. Govern NanoJ programs and fieldbus network (with CANopen).

Set Parameters: Transmits *the selected* parameter values to system-connected controllers.

Note: You can select parameters to be set by ticking them.



HOMING SETTINGS

Homing method	1	☑
Home offset	0	☑
Speed during search for switch	33	☑
Speed during search for zero	11	☑
Homing acceleration	12000	☑
Minimum current for block detection	0	□
Block detection time	100	□

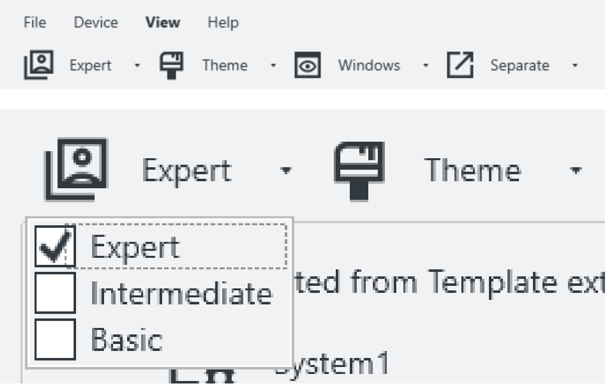
Get Parameters: *Reads* the values of system-connected controllers.

Store Parameters: *Stores* **Set**-transmitted values of system-connected controllers.

Restore Factory Default: *Restores* all parameters to their default values with the exception of the parameters related to *tuning* (motor/sensor specific) and *fieldbusses*. For further details refer to your controller manual.

Restart Device: *Restarts* the system-connected controller.

View

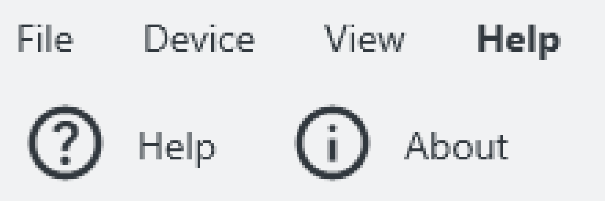


Here you can customize the UI by changing the theme, showing/hiding features or opening them in separate windows.

You can also set the **User level**, to govern user rights for the following roles:

- **Expert**: Project owner with all rights. May create and edit projects, rights, visibilities, etc. Governs via **Properties**, for each single parameter up to a complete **Controls group**, *who* may see and edit exactly *what*.
- **Intermediate**: May change device parameters, but can't edit a project.
- **Basic**: Similar to **Intermediate**, but often gets fewer editing rights from **Expert**.

Help

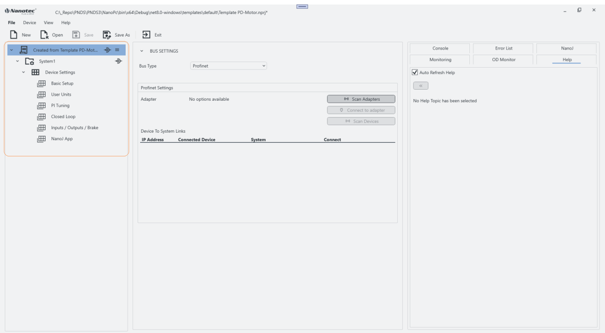


Open the online help or PNDS3 version info.

6.2 Project bar (4)

This side bar diplays your loaded project as a tree list by which you create the user interface. **Note:** Depending on assembly, you can check connections and attributes of all tree list items in the work desk (3).

Tree list

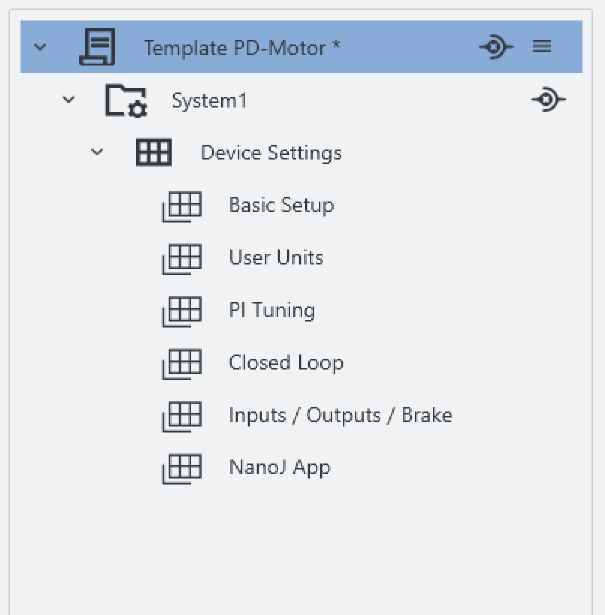

C:_Repo\PNDS\PNDS3\NanoPc\bin\x64\Debug\net8.0-windows\templates\default\Template PD-Motor.nprj*

Bus Type: Profinet

Adapter: No options available

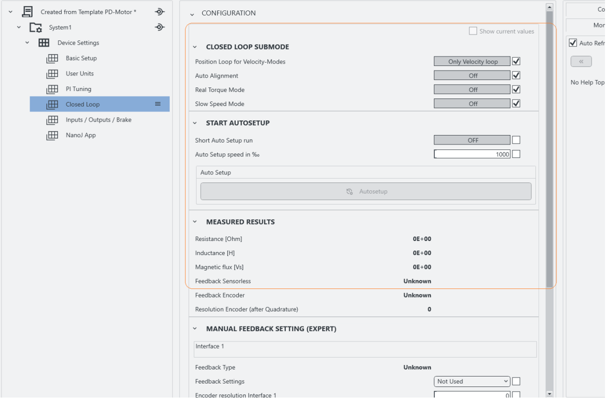
No Help Topic has been selected

You find the project bar in the very left of the user screen.



A project (here: for an *external controller*) tree-lists all systems and the items therein (see also Project setup). *One* project and *one* system are minimum; further items are optional and later on define the entire UI layout.

System 1 contains the module groups *Quick Start* with modules for the basic settings and *Application Settings* with further controls and parameter groups.



□ Show current values

CLOSED LOOP SUBMODE

Position Loop for Velocity-Modes: Only Velocity loop ☑
Auto Alignment: Off ☑
Real Torque Mode: Off ☑
Slow Speed Mode: Off ☑

START AUTOSETUP

Short Auto Setup run: OFF □
Auto Setup speed in ‰: 1000 □

Auto Setup

MEASURED RESULTS

Resistance [Ohm]: 0E+00
Inductance [H]: 0E+00
Magnetic flux [Vs]: 0E+00
Feedback Sensorless: Unknown
Feedback Encoder: Unknown
Resolution Encoder (after Quadrature): 0

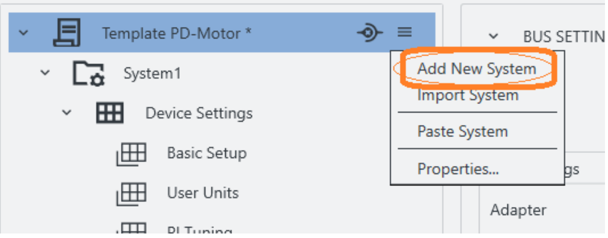
MANUAL FEEDBACK SETTING (EXPERT)

Interface 1

Feedback Type: Unknown
Feedback Settings: Not Used □
Encoder resolution Interface 1: 0 □

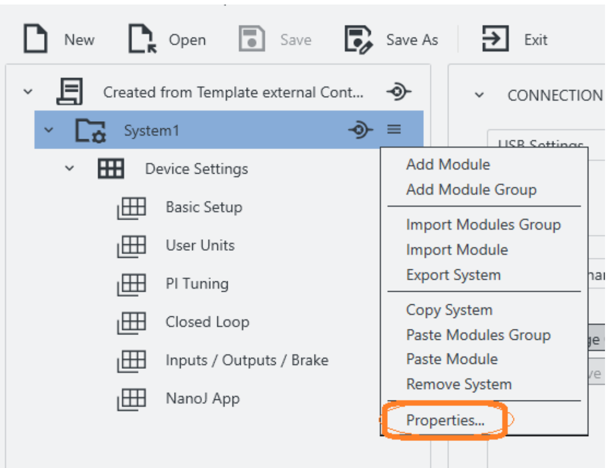
For each module, you may add one or more controls groups to the work desk (3) further to the right.

Project > System

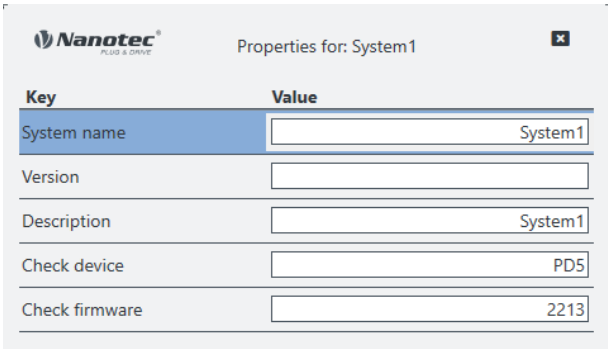


A system represents a motor with controller, that is, one per motor in a multi-axis application.

1. To set up a system: Co-click the project.
2. In the context menu: Either create a new system via **Add New System**.
3. Or fetch an existing one via **Import System**.



4. A new node (= blue) appears in the tree list.
5. To name it: Co-click the node, select **Properties**. **Note:** You can edit *any* object via **Properties**.

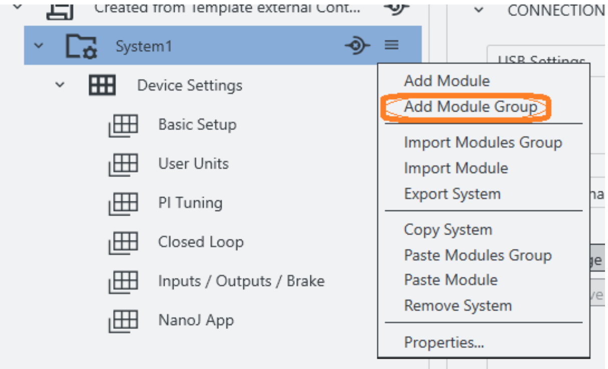


Properties for: System1

Key	Value
System name	System1
Version	
Description	System1
Check device	PD5
Check firmware	2213

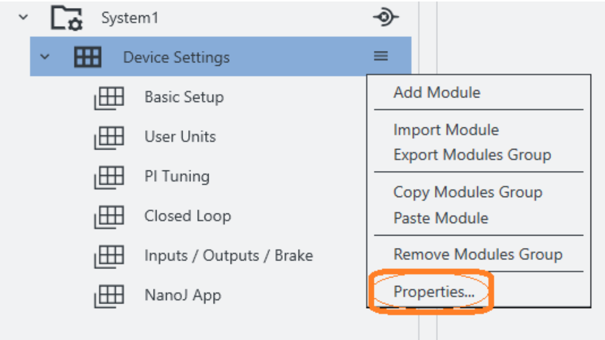
6. In the pop-up: Name the system as needed.
7. If needed: Versionize and describe the system. You can add a string for the device name and firmware version, which should be checked after connecting to a device.
8. After last entry: Set a tab stop (so that all is stored).
9. Only then: Close the pop-up.
10. Assemble the system with module groups (see below).

Project > System > Module group

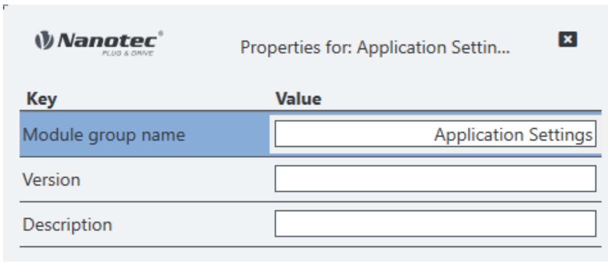


A module group bundles *several* motor functions (= modules). Depending on assembly, you can check its connections and attributes in the work desk (3).

1. To set up a module group: Co-click the system.
2. In the context menu: Either create a new module group via **Add module group**.
3. Or fetch an existing one via **Import module group**.



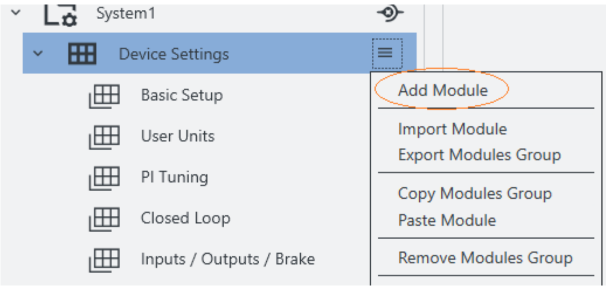
4. A new node appears in the tree list.
5. To name it: Co-click the node, select **Properties**. **Note:** You can edit *any* object via **Properties**.



Key	Value
Module group name	Application Settings
Version	
Description	

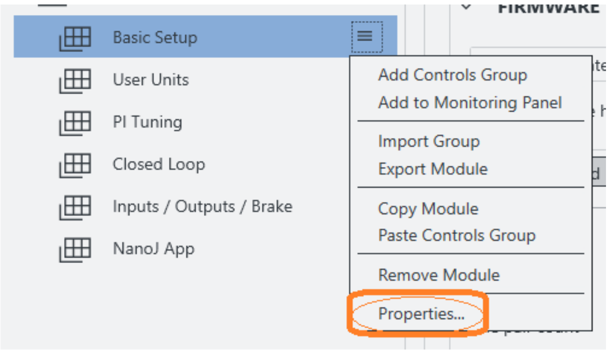
6. In the pop-up: Name the module group as needed (here: *Controller template*).
7. If needed: Versionize and describe the module group.
8. After last entry: Set a tab stop (so that all is stored).
9. Only then: Close the pop-up.
10. Assemble the module group with modules (see below).

Project > System > Module group > Module

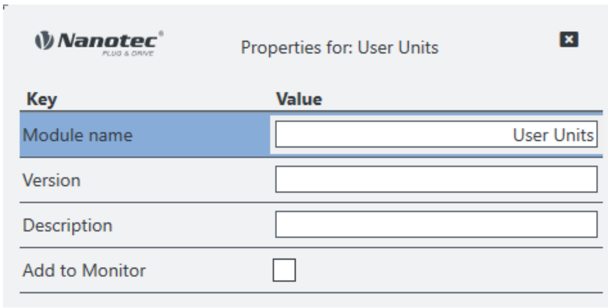


A module allows you to add a *single* motor function (= parameter set etc.). Depending on assembly, you can check its connections and attributes in the work desk (3).

1. To set up a module: Co-click the module group (here: *Controller template*).
2. In the context menu: Either create a new module via **Add Module**.
3. Or fetch an existing one via **Import Module**.



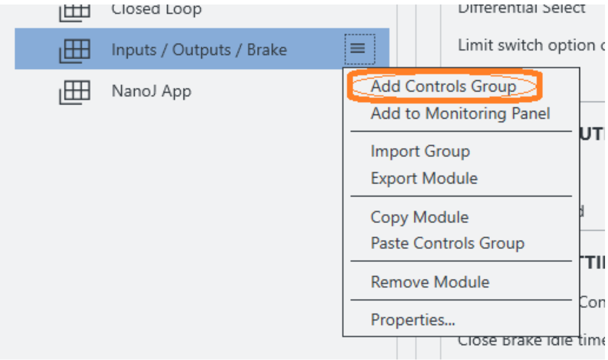
4. A new node appears in the tree list.
5. To name it: Co-click the node, select **Properties**. **Note:** You can edit *any* object via **Properties**.



Key	Value
Module name	User Units
Version	
Description	
Add to Monitor	□

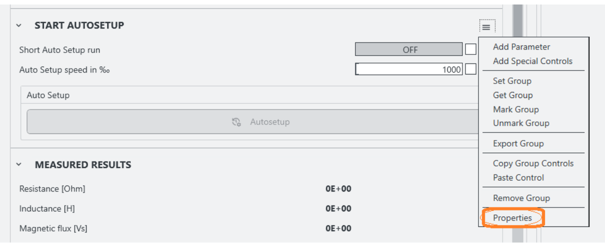
6. In the pop-up: Name the module as needed (here: *Communication settings*).
7. If needed: Versionize and describe the module, add it to a Monitor
8. After last entry: Set a tab stop (so that all is stored).
9. Only then: Close the pop-up.

Project > System > Module group > Module > Controls group



A controls group bundles individual operating elements or parameter sets.

1. To set up a controls group: Co-click the module.
2. In the context menu: Either create a new controls group via **Add Controls Group**.
3. Or fetch an existing one via **Import Group**.



Short Auto Setup run: OFF

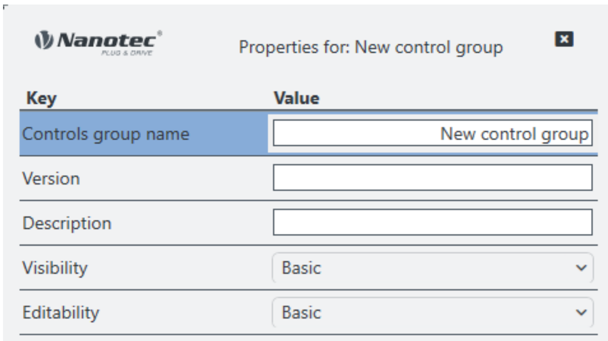
Auto Setup speed in ‰: 1000

Resistance [Ohm]: 0E+00

Inductance [H]: 0E+00

Magnetic flux [Vs]: 0E+00

4. In any case, the controls group appears in the work desk (3).
5. Right there: Co-click the group and its **Properties**. **Note:** You can edit *any* object via **Properties**.


Properties for: New control group

Key	Value
Controls group name	New control group
Version	
Description	
Visibility	Basic
Editability	Basic

6. In the pop-up: Name the controls group as needed.
7. If needed: Versionize and describe the group. **Note** the pull-downs for granted viewing and editing rights (here: both *Basic*).
8. After last entry: Set a tab stop (so that all is stored).
9. Only then: Close the pop-up.

6.3 Work desk (3)

At the work desk, in the user screen's half-left, you edit the properties / contents / controls of your project and systems. Depending on assembly, different tabs are above the worktable:

An **Attributes** area accompanies all items (also module groups); **Bus settings**, by contrast, only the project itself. The **Connection settings** tab, finally, is for systems only; and **Configuration** is only for modules. Each tab opens different aspects:

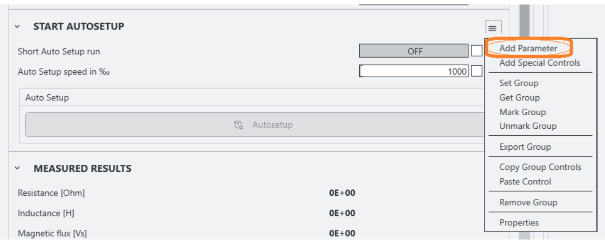
Controls groups Operator clusters

Parameters Operator values

Special controls Feature operators

Complex controls Multi-level operators

Controls group > Parameter



Short Auto Setup run: OFF

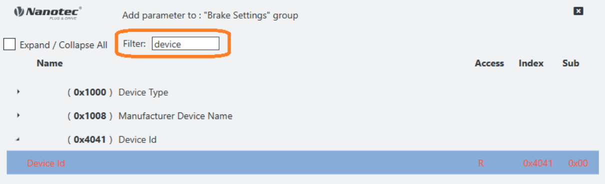
Auto Setup speed in ‰: 1000

Resistance [Ohm]: 0E+00

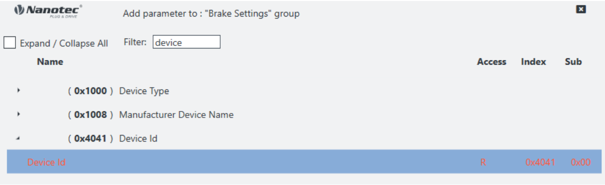
Inductance [H]: 0E+00

Magnetic flux [Vs]: 0E+00

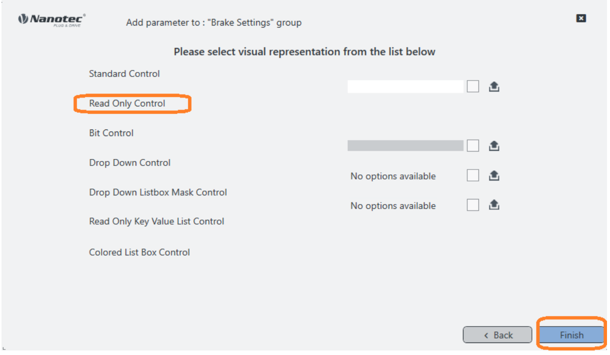
1. To add a parameter: Co-click the controls group and click **Add Parameter**.



2. In the pop-up: Enter *device* or *0x4041* to filter for the **Device Id** object.

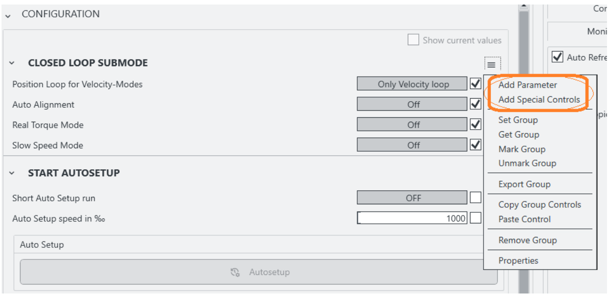


3. You may expand objects by mouse (or tick at **Expand all**).
4. Click **Device Id** and **Next** (if wrong: step **Back**).

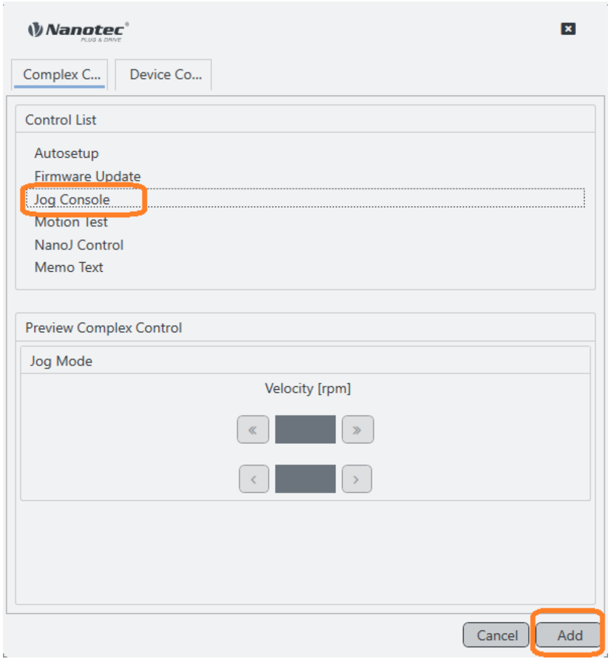

Nanotec
Add parameter to : "Brake Settings" group
Please select visual representation from the list below
Standard Control
Read Only Control
Bit Control
Drop Down Control
Drop Down Listbox Mask Control
Read Only Key Value List Control
Colored List Box Control
No options available
No options available
‹ Back
Finish

5. In the next pop-up: Select the visual reprentation and click **Finish**.

Controls group > Special controls



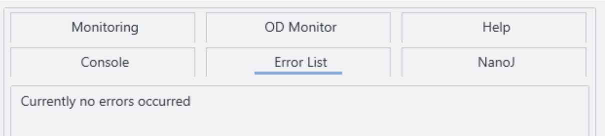
1. Co-klick a controls goup to open its context menu.
2. Select **Add Special Controls** to open the **Complex controls** list.



3. In the pop-up: Select the needed item.
4. To confirm: Click **Add**.

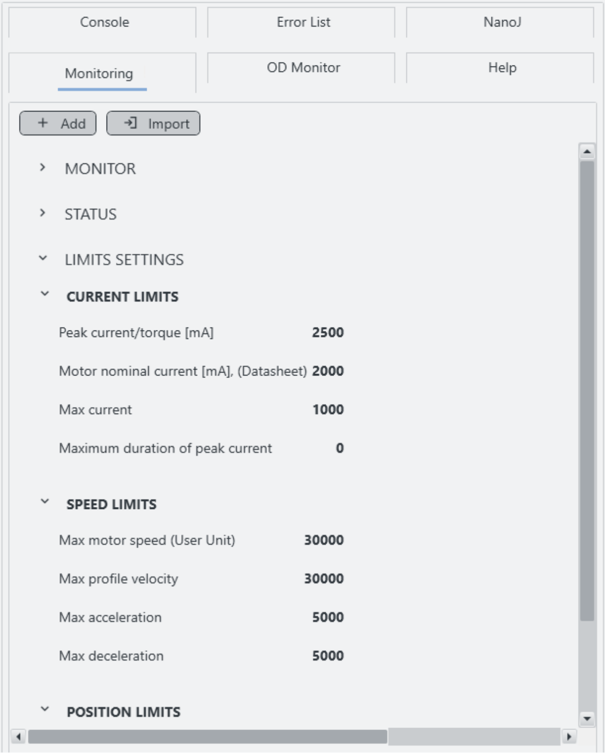
6.4 Display wall (2)

The display wall contains the monitors, error list, current OD values, help and console.



Currently no errors occurred

Several tabs facilitate navigation in the display wall, in the user screen's upper right.



MONITOR

STATUS

LIMITS SETTINGS

CURRENT LIMITS

Peak current/torque [mA] 2500

Motor nominal current [mA], (Datasheet) 2000

Max current 1000

Maximum duration of peak current 0

SPEED LIMITS

Max motor speed (User Unit) 30000

Max profile velocity 30000

Max acceleration 5000

Max deceleration 5000

POSITION LIMITS

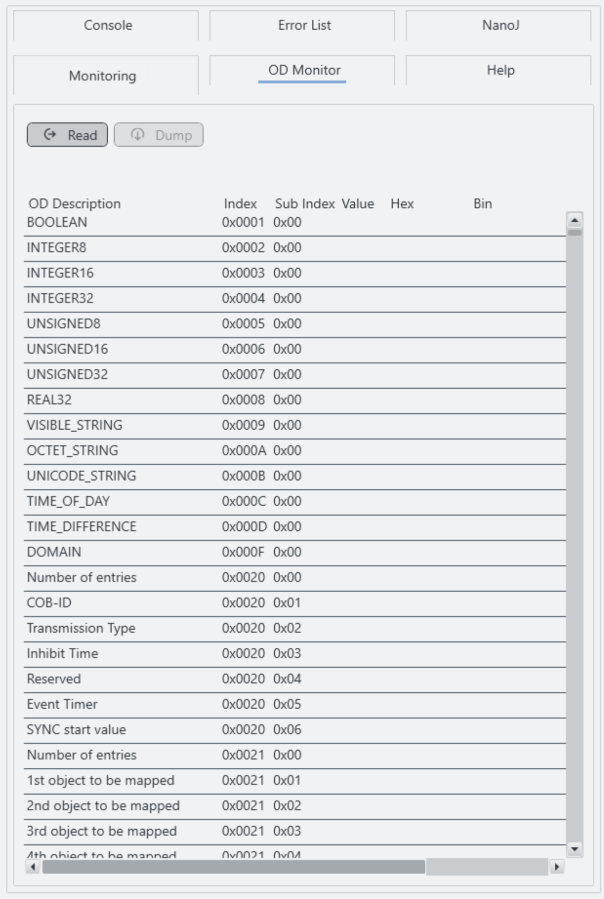
In the **Monitoring** tab, you combine either single or grouped monitors to track individual system behavior in real time.

1. To set up a monitor: Co-click the tab **Monitoring**.
2. In the context menu: Either create a new monitor via **Add Monitor Module**.
3. Or fetch an existent one via **Import Module**.

The bottom part of the monitor display contains always the system monitor which shows the current device and connection status information.

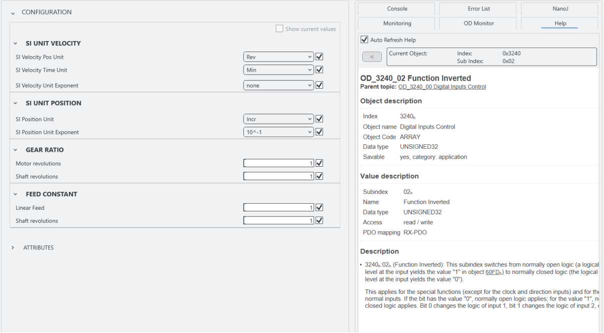
OD Monitor: Lists all objects from the controller's dictionary, together with their current values. For updates: Click **Read**.

To save the list as a text file on the hard drive: Click **Dump**. Keep the text file with current values ready in case of support enquiries.



OD Description	Index	Sub Index	Value	Hex	Bin
BOOLEAN	0x0001	0x00			
INTEGER8	0x0002	0x00			
INTEGER16	0x0003	0x00			
INTEGER32	0x0004	0x00			
UNSIGNED8	0x0005	0x00			
UNSIGNED16	0x0006	0x00			
UNSIGNED32	0x0007	0x00			
REAL32	0x0008	0x00			
VISIBLE_STRING	0x0009	0x00			
OCTET_STRING	0x000A	0x00			
UNICODE_STRING	0x000B	0x00			
TIME_OF_DAY	0x000C	0x00			
TIME_DIFFERENCE	0x000D	0x00			
DOMAIN	0x000F	0x00			
Number of entries	0x0020	0x00			
COB-ID	0x0020	0x01			
Transmission Type	0x0020	0x02			
Inhibit Time	0x0020	0x03			
Reserved	0x0020	0x04			
Event Timer	0x0020	0x05			
SYNC start value	0x0020	0x06			
Number of entries	0x0021	0x00			
1st object to be mapped	0x0021	0x01			
2nd object to be mapped	0x0021	0x02			
3rd object to be mapped	0x0021	0x03			
4th object to be mapped	0x0021	0x04			

Help: Displays the description of the currently chosen element (OD object).



SI UNIT VELOCITY

SI Velocity Pos Unit	Rev	☑
SI Velocity Time Unit	Min	☑
SI Velocity Unit Exponent	none	☑

SI UNIT POSITION

SI Position Unit	Incr	☑
SI Position Unit Exponent	10^-1	☑

GEAR RATIO

Motor revolutions	1	☑
Shaft revolutions	1	☑

FEED CONSTANT

Linear Feed	1	☑
Shaft revolutions	1	☑

☑ Auto Refresh Help

Current Object: Index: 0x3240 Sub Index: 0x02

OD_3240_02 Function Inverted
Parent topic: OD_3240_00 Digital Inputs Control

Object description

Index	3240_h
Object name	Digital Inputs Control
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application

Value description

Subindex	02_h
Name	Function Inverted
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO

Description

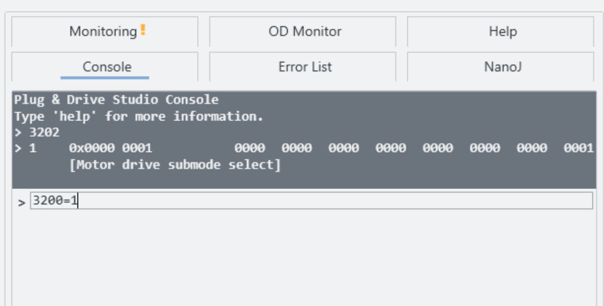
- 3240_h:02_h (Function Inverted): This subindex switches from normally open logic (a logical level at the input yields the value "1" in object $60FD_h$) to normally closed logic (the logical level at the input yields the value "0").

  This applies for the special functions (except for the clock and direction inputs) and for the normal inputs. If the bit has the value "0", normally open logic applies; for the value "1", n closed logic applies. Bit 0 changes the logic of input 1, bit 1 changes the logic of input 2,

Console: Use this to quickly read/write.from/to the device's object dictionary.

Type <od index>:<od subindex> for read.

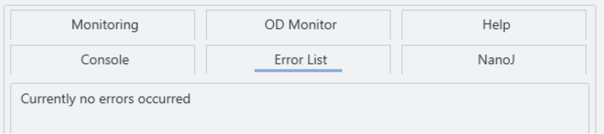
Type <od index>:<od subindex>=<value> for write.



```
Plug & Drive Studio Console
Type 'help' for more information.
> 3202
> 1     0x0000 0001          0000  0000  0000  0000  0000  0000  0000  0001
       [Motor drive submode select]
```

```
> 3200=1
```

Error List: Here you can read the actual errors.

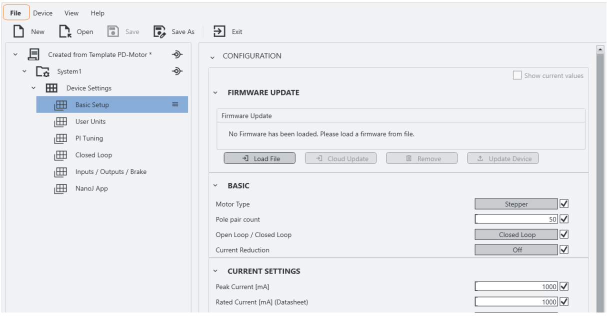


Currently no errors occurred

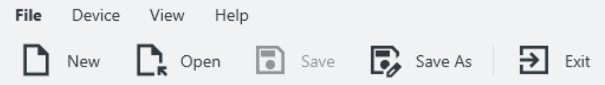
7 Project setup

In a project, you manage your devices, settings, connections, etc. **Note:** Ex works, in the software's templates folder, there is a sample project each for an external andan integrated controller. Nanotec recommends using these templates.

Load / Create a project



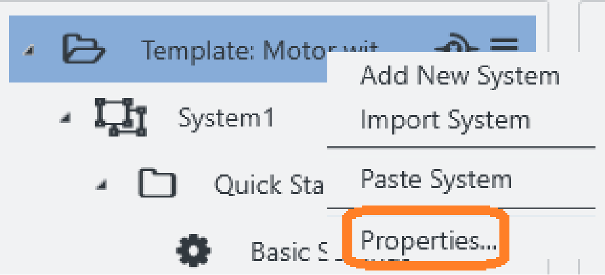
1. In the user screen: Visit the file menu (1)



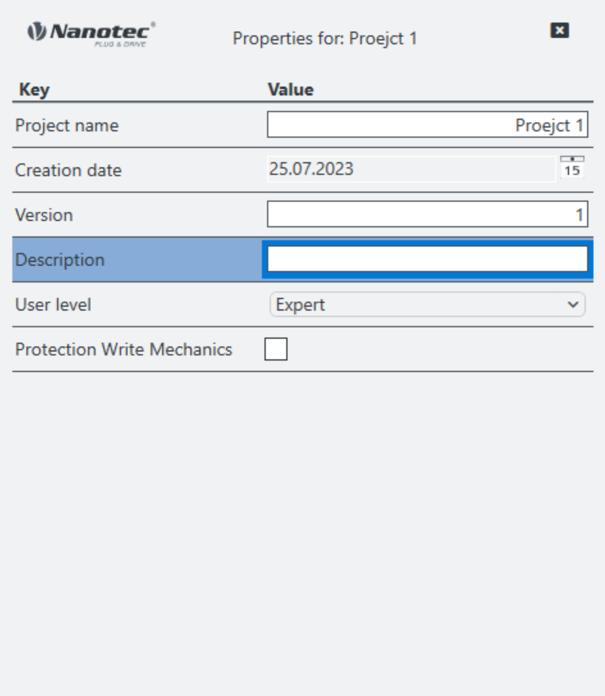
2. Preferably use **Project > Open** to select an existent sample project for template.
3. Or, for a new one instead: Select **Project > New**.

4. If a pop-up wants to store the current project: Click **Yes**.
   - **No** will close the project unstored and without backup.
   - **Cancel** will just close the pop-up.
5. The newly loaded selection appears in the project bar (4)

If needed: Name the project



1. Go to the Project bar (4).
2. Co-click the current project and **Properties**.

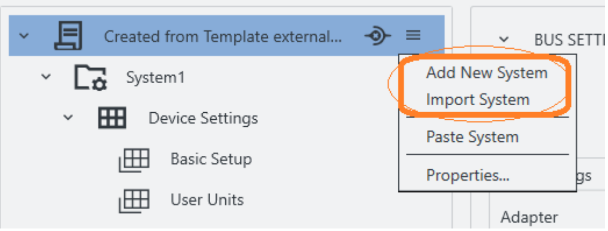


Properties for: Proejct 1

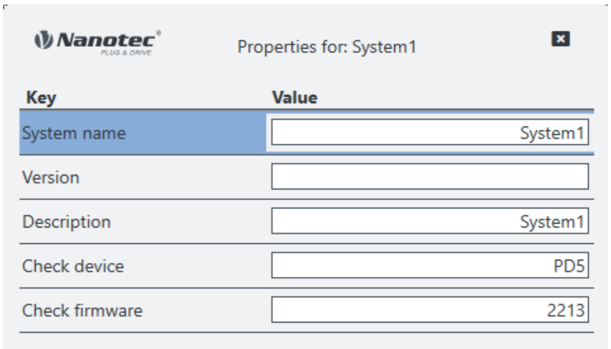
Key	Value
Project name	Proejct 1
Creation date	25.07.2023
Version	1
Description	
User level	Expert
Protection Write Mechanics	□

3. In the pop-up: Name, versionize, and describe the project. **Note** the pull-down for granted user rights (here: *Expert*).
4. After last entry: Set a tab stop (so that all is stored).
5. Only then: Close the pop-up.

Load / Create a system



1. In the project: Preferably use **Import System** to select an existent sample system for template.
2. Or, for a new one instead: Select **Add new system**.



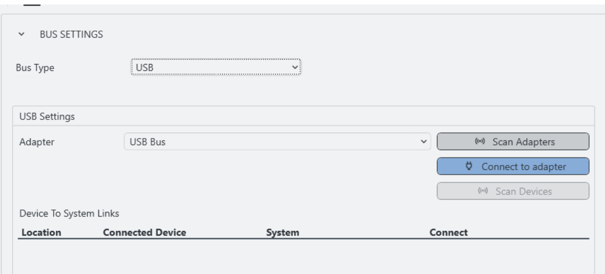
Properties for: System1

Key	Value
System name	System1
Version	
Description	System1
Check device	PD5
Check firmware	2213

3. In the pop-up: Name, versionize, describe the system as needed.
4. After last entry: Set a tab stop (so that all is stored).
5. Only then: Close the pop-up.
6. Repeat for each additional system.

Connect to adapter

1. In the project bar (4): Select your project.



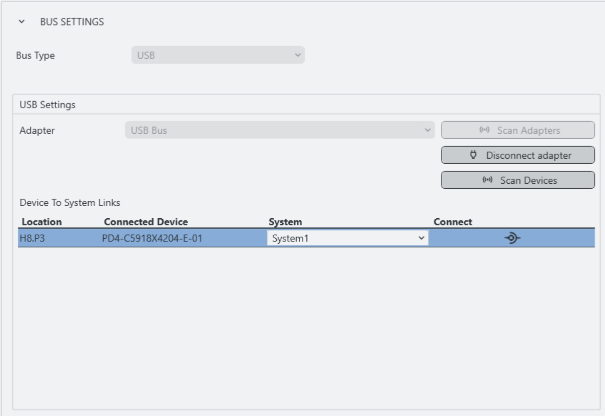
Bus Type: USB

Adapter: USB Bus

Location	Connected Device	System	Connect

2. In the work desk (3): Open the **Bus settings** tab.
3. In the **Bus settings** tab: Select the **Bus type**.
4. Check setup by **Scan adapters**. If no result: Set up an adapter and check again.
5. Select the needed adapter.

Connect to device



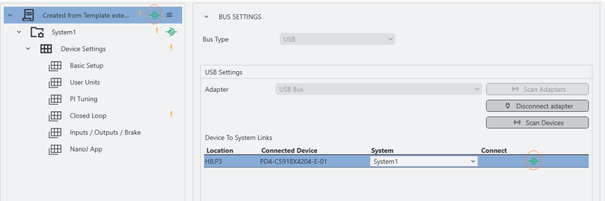
Bus Type: USB

Adapter: USB Bus

Device To System Links

Location	Connected Device	System	Connect
H8.P3	PD4-C5918X4204-E-01	System1	

1. In the **Bus settings** tab: With the adapter linked, you can see all available devices.
2. Click **Scan devices**. Check **Connected device**.
3. By Pulldown: Select a **system** to link your device to.



Bus Type: USB

Adapter: USB Bus

Device To System Links

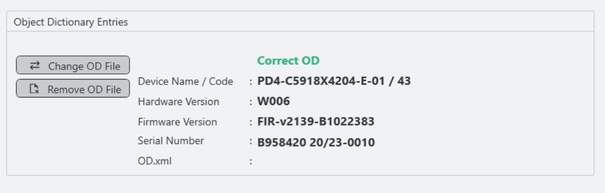
Location	Connected Device	System	Connect
H8.P3	PD4-C5918X4204-E-01	System1	

4. You can link / unlink the device via **Connect** icon (here: green).

Select the OD file

PNDS3 shows objects that match the controller firmware with correct OD file only (object dictionary). If the system is linked, a **Object Dictionary Entries** tab shows if the correct OD file is loaded. Otherwise, the generic file *Common OD* loads, by which you reach available objects of all Nanotec products.

1. Select the system.
2. Open the **Object Dictionary Entries** tab.



Correct OD

Device Name / Code : PD4-C5918X4204-E-01 / 43

Hardware Version : W006

Firmware Version : FIR-v2139-B1022383

Serial Number : B958420 20/23-0010

OD.xml :

- *Common OD*: Reloadable via **Remove OD File**
- OD file of choice: Loadable via **Change OD File**
- Firmware-correct OD files for all Nanotec controllers: In PNDS3's **Firmware** folder

Wrong OD files report an error (= red).

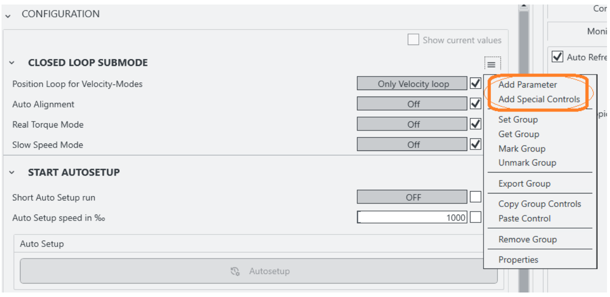
8 Special controls

Via **Special controls**, you add **Complex controls** and **Device communication settings** to the user interface. Both help you to use advanced controller functions.

Basic principle

Special controls define and monitor (as macro collections) the system behavior. Depending on assembly, you can check their connections and attributes in the work desk.

1. To set up complex controls or device communication: Co-click the controls group.
2. In the context menu: Select **Add Special Controls**.



CONFIGURATION

□ Show current values

CLOSED LOOP SUBMODE

Position Loop for Velocity-Modes: Only Velocity loop ☑

Auto Alignment: Off ☑

Real Torque Mode: Off ☑

Slow Speed Mode: Off ☑

START AUTOSETUP

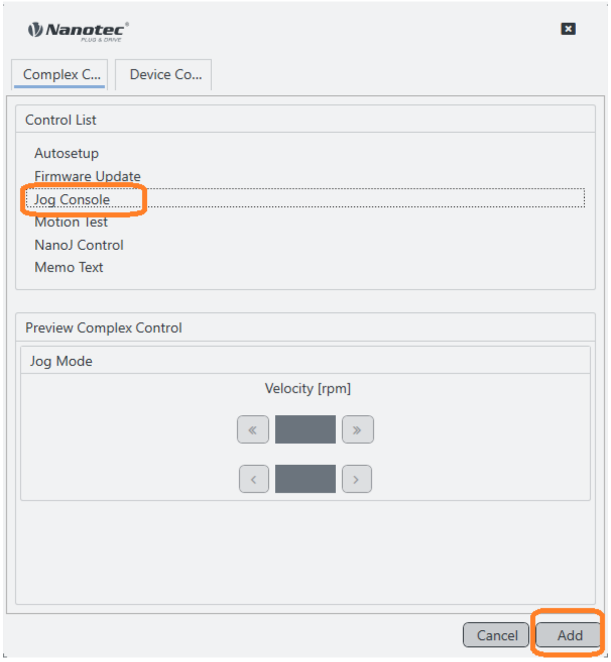
Short Auto Setup run: OFF □

Auto Setup speed in ‰: 1000 □

Auto Setup

3. **Complex controls**? **Device communication settings**? Open the tab of choice.
4. Select the needed item and **Add**.

→ The control / setting of choice appears in the work desk.



Control List

- Autosetup
- Firmware Update
- Jog Console
- Motion Test
- NanoJ Control
- Memo Text

Preview Complex Control

Jog Mode

Velocity [rpm]

8.1 Complex controls

With the **Complex controls** macro collection, you create your own controller functions. Next to **Autosetup** and **Firmware update**, these include **Jog Console, Motion Test, NanoJ Control** and **Memo Text.**

Autosetup

Autosetup detects the motor type and connected sensors (encoder / Hall sensors).

CAUTION!

Injury: from abrupt motor travel after auto-setup (= parameter loss)!

▶For motors with integrated controllers: Avert auto-setup (since it comes factory-run already).

▶Otherwise: Restart the motor after auto-setup (homing alone won't suffice).

▶Stay clear of moving motor parts.

▶Touch the motor at standstill only.

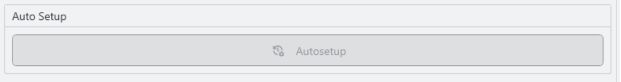
NOTICE

Motor malfunction: from auto-setup user error!

▶Close possible NanoJ programs (object 2300_h:00_h Bit 0 = "0"; cf. 2300h NanoJ Control).

▶Keep the motor load-free, and freely rotable in any direction.

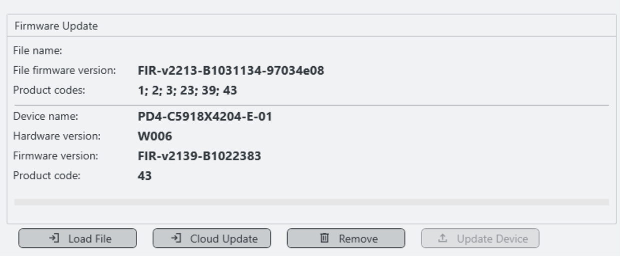
▶**Don't** touch the motor.



As long as the motor on the controller or the feedback sensors (encoder / Hall) remain the same: Run **Autosetup** only once, on initial commissioning.

Firmware update

Nanotec recommends controller firmware *FIR-v2213* or newer. Please find the current version in the **Firmware** folder on the PNDS3-website.


Firmware Update

File name:

File firmware version: FIR-v2213-B1031134-97034e08

Product codes: 1; 2; 3; 23; 39; 43

Device name: PD4-C5918X4204-E-01

Hardware version: W006

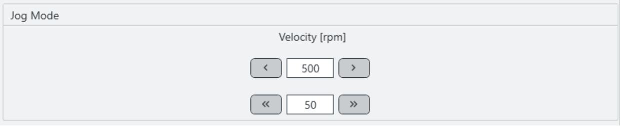
Firmware version: FIR-v2139-B1022383

Product code: 43

1. Open or add the**Firmware update** control.
2. Click **Load from file**.
3. Select a firmware file and click **Open**.
4. PNDS3 checks via product code if the chosen file fits to the product.
5. Click **Update device**.
6. Firmware updates itself.

Note: The chosen firmware file will be stored as part of the project the next time the latter is stored. If you don't want this to happen, click **Remove from project** before.

Jog Console

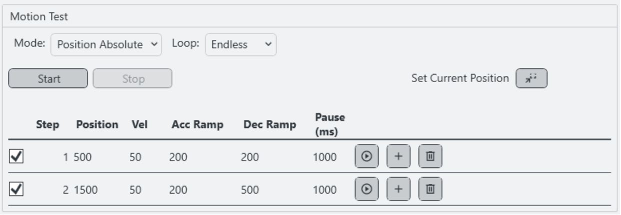

Jog Mode

Velocity [rpm]: 500

50

Via **Jog Console**, you test the motor in velocity mode. You can select two target speeds. The motor runs as long as you use the mouse to press the button for left / right rotation.

Motion Test

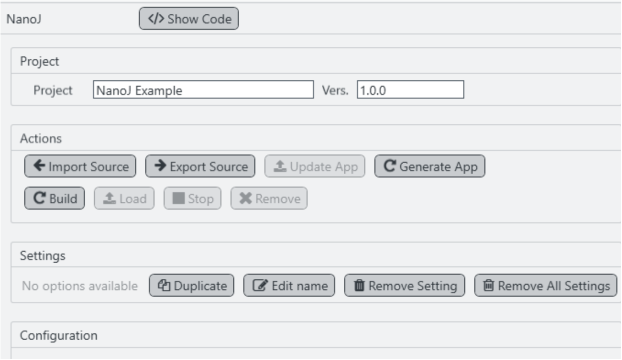

Motion Test

Mode: Position Absolute Loop: Endless

	Step	Position	Vel	Acc Ramp	Dec Ramp	Pause (ms)
☑	1	500	50	200	200	1000
☑	2	1500	50	200	500	1000

In **Motion Test**, you test the motor in position / velocity / torque mode. Your options include target values, acceleration / deceleration ramps, repetition cycles, test run duration etc.

NanoJ Control



Project: NanoJ Example Vers.: 1.0.0

No options available

In **NanoJ control**, you create a NanoJ new project (= **New**) or **Import** an existing one (code examples available in the *Knowledge Base* at nanotec.com). The button **Build** compiles the project.

The **Settings** and **Configuration** sections are reserved for the NanoJ App.

Note: The next time you store the project, the selected NanoJ file merges into the project. If you don't wish this to happen, click **Remove** before.

You can find further details in the chapter Programming with NanoJ.

Memo Text

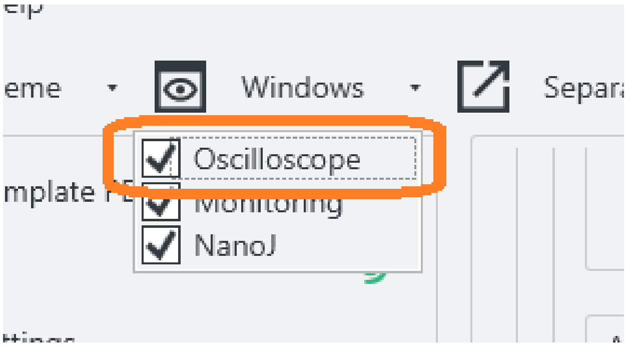
Adds a freely editable text box.

8.2 Device communication

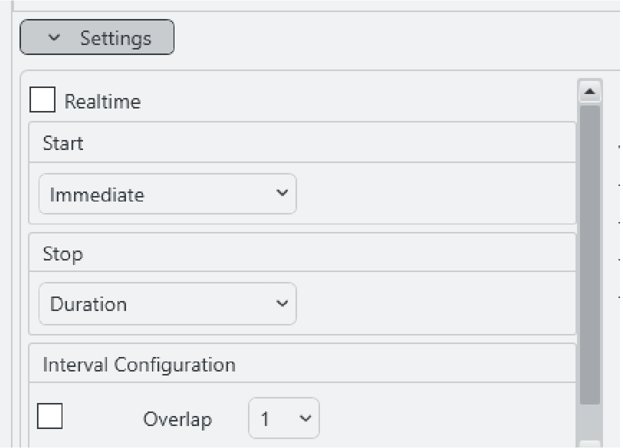
With these controls, you parametrize the device communication. **Note:** Coding switches for setting the communication parameters overwrite the software settings on some devices. For details: Follow valid OEM instructions.

9 Oscilloscope

Via **Oscilloscope**, you monitor and control in real time the current value of device parameters from the object dictionary, say, for recording.



To open the oscillosope, go to the **Main Menu > Windows** and select it.


Settings

☐ Realtime

Start

Immediate

Stop

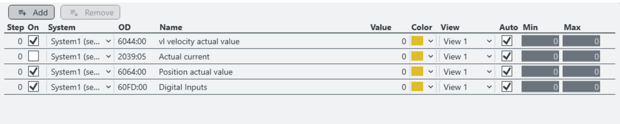
Duration

Interval Configuration

☐ Overlap 1

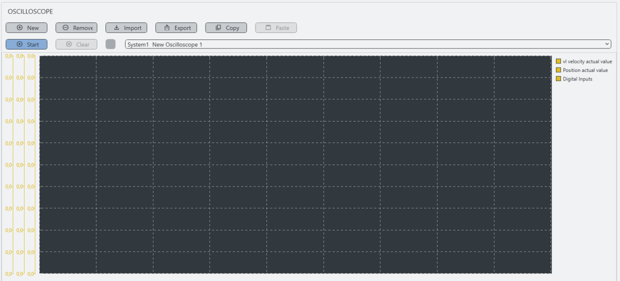
Under **Settings** you can configure the following:

- **Realtime:** If chosen, the oscilloscope starts imeediately and runs continuously, until the buffer is full. If not selected you can further define the conditions for start/stop.
- **Start: Immediate**, **Condition** (as soon as a parameter of choice changes), or **Motion test** (Motion test triggers the scope).
- **Stop: Duration** (of recording) or **Manual**.



Step	On	System	OD	Name	Value	Color	View	Auto	Min	Max
0	☑	System1 (se...	6044:00	vl velocity actual value	0	🟨	View 1	☑	0	0
0	☐	System1 (se...	2039:05	Actual current	0	🟨	View 1	☑	0	0
0	☑	System1 (se...	6064:00	Position actual value	0	🟨	View 1	☑	0	0
0	☑	System1 (se...	60FD:00	Digital Inputs	0	🟨	View 1	☑	0	0

In the right bottom corner you can add channels by selecting from the object dictionary or remove them.


OSCILLOSCOPE

System1 New Oscilloscope 1

For recording, you open a new (or import an existing) oscilloscope in the tab above the settings and click **Start**.

10 Programming with *NanoJ*

NanoJ is a programming language similar to *C* or *C++*.

10.1 NanoJ program

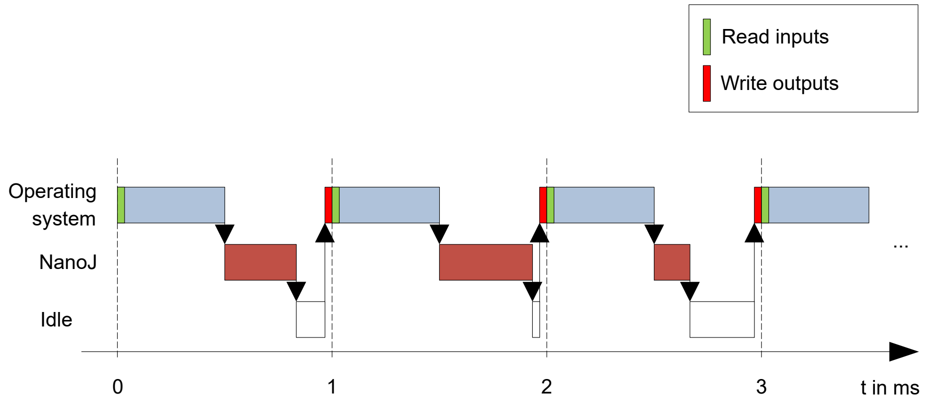
A *NanoJ program* makes a protected runtime environment available within the firmware. Here, the user can create his own processes. These can then trigger functions in the controller by, for example, reading or writing entries in the object dictionary.

Through the use of protective mechanisms, a *NanoJ program* is prevented from crashing the firmware. In the worst case, the execution is interrupted with an error code stored in the object dictionary.

If the *NanoJ program* was loaded on the controller, it is automatically executed after the controller is switched on or restarted, as long as you do not set bit 0 in object 2300$_h$ to "0".

10.1.1 Available computing time

A *NanoJ program* receives computing time cyclically in a 1 ms clock (see following figure). Because computing time is lost through interrupts and system functions of the firmware, only a portion of the computing time is available to the user program (depending on control mode and application). In this time, the user program must run through the cycle and either complete the cycle or yield the computing time by calling the `yield()` function. In the former case, the user program is restarted with the start of the next 1 ms cycle; the latter results in the program being continued on the next 1 ms cycle with the command that follows the `yield()` function.



If the *NanoJ program* needs more time than was allotted, it is ended and an error code set in the object dictionary.

TIP

When developing user programs, the runtime behavior must be carefully examined, especially for more time-intensive tasks. For example, it is therefore recommended that tables be used instead of calculating a sine value using a `sin` function.

NOTICE

If the *NanoJ program* does not yield the computing time after too long a time, it is ended by the operating system. In this case, the number `4` is entered in the statusword for object 2301$_h$; in the error register for object 2302$_h$, the number `5` (timeout) is noted, see OD_2301_00 NanoJ Status and OD_2302_00 NanoJ Error Code.

To keep the *NanoJ program* from stopping, you can activate *AutoYield* mode by writing value "5" in 2300$_h$. In *AutoYield* mode, however, the *NanoJ program* is no longer real-time capable and no longer runs every 1 ms.

10.1.2 Protected runtime environment

Using process-specific properties, a so-called *protected runtime environment* is generated. A user program in the protected runtime environment is only able to access specially allocated memory areas and system resources. For example, an attempt to directly write to a processor IO register is acknowledged with an *MPU Fault* and the user program terminated with the corresponding error code in the object dictionary.

10.1.3 NanoJ program – communication possibilities

A *NanoJ program* has a number of possibilities for communicating with the controller:

- Read and write OD values using PDO mapping
- Directly read and write OD values via NanoJ functions
- Call other NanoJ functions (e.g., write debug output)

The OD values of the user program are made available in the form of variables via *PDO mapping*. Before a user program receives the 1 ms time slot, the firmware transfers the values from the object dictionary to the variables of the user program. As soon as the user program receives computing time, it can manipulate these variables as regular C variables. At the end of the time slot, the new values are then automatically copied by the firmware back to the respective OD entries.

To optimize the performance, three types of mapping are defined: input, output, and input/output (In, Out, InOut).

- *Input mappings* can only be read; they are not transferred back to the object dictionary.
- *Output mappings* can only be written.
- *Input/output mappings*, on the other hand, can both be read and written.

The set mappings can be read and checked via the GUI for objects 2310$_h$, 2320$_h$, and 2330$_h$. Up to 16 entries are allowed for each mapping.

Whether a variable is stored in the input, output or data range is controlled in *Plug & Drive Studio* via the specification of the *linker section*.

NanoJ inputs and NanoJ outputs

To communicate with the *NanoJ program* via the respective interface, you can use the following objects:

- OD_2400_00 NanoJ Inputs: Array with thirty-two S32 values for passing values to the *NanoJ program*
- OD_2410_00 NanoJ Init Parameters: Array with thirty-two S32 values. This object can be stored, unlike 2400$_h$.
- OD_2500_00 NanoJ Outputs: Array with thirty-two S32 values, where the *NanoJ program* can store values that can be read out via the fieldbus

10.1.4 Executing a NanoJ program

When executing a cycle, the *NanoJ program* essentially consists of the following three steps with respect to the PDO mapping:

1. Read values from the object dictionary and copy them to the input and output areas
2. Execute a user program
3. Copy values from the output and input areas back to the object dictionary

The configuration of the copy processes is based on the CANopen standard.

In addition, values of the object dictionary can be accessed via NanoJ functions. This is generally slower; mappings are therefore to be preferred. The number of mappings is limited (16 entries each in In/Out/InOut).

TIP

Nanotec recommends: Map OD entries that are used and changed frequently and use NanoJ function to access OD entries that are used less frequently.

A list of available NanoJ functions can be found in chapter NanoJ functions in the NanoJ program.

TIP

Nanotec recommends accessing a given OD value either by mapping or using a NanoJ function with `od_write()`. If both are used simultaneously, the NanoJ function has no effect.

10.1.5 Structure of a NanoJ program

A user program consists of at least two instructions:

- the preprocessor instruction `#include "wrapper.h"`
- the `void user(){}` function

The code to be executed can be stored in the `void user()` function.

NOTICE

The file names of the user programs must not be longer than eight characters plus three characters in the suffix; file name `main.cpp` is permissible, file name `aLongFileName.cpp` is not permissible.

NOTICE

In *NanoJ programs*, global variables may only be initialized within functions. It then follows:

- No `new` operator
- No constructors
- No initialization of global variables outside of functions

Examples:

The global variable is to be initialized within the `void user()` function:

```
unsigned int i;
void user(){
 i = 1;
 i += 1;
}
```

The following assignment results in an error during compilation:

```
unsigned int i = 1;
 void user() {
 i += 1;
}
```

10.1.6 NanoJ program example

The example shows the programming of a square wave signal in object 2500_h:01_h.

```
// file main.cpp
map S32 outputReg1 as inout 0x2500:1
#include "wrapper.h"

// user program
void user()
{
  U16 counter = 0;
  while( 1 )
  {
    ++counter;

    if( counter < 100 )
     InOut.outputReg1 = 0;
    else if( counter < 200 )
      InOut.outputReg1 = 1;
    else
      counter = 0;

    // yield() 5 times (delay 5ms)
    for(U08 i = 0; i < 5; ++i )
      yield();
  }
}// eof
```

You can find other examples at us.nanotec.com.

10.2 Mapping in the NanoJ program

With this method, a variable in the *NanoJ program* is linked directly with an entry in the object dictionary. The creation of the mapping must be located at the start of the file here, even before the `#include "wrapper.h"` instruction.

TIP

Nanotec recommends:

- Use mapping if you need to access an object in the object dictionary frequently, e. g., *controlword* 6040_h or *statusword* 6041_h.
- The `od_write()` and `od_read()` functions are better suited for accessing objects a single time, see Accessing the object dictionary.

10.2.1 Declaration of the mapping

The declaration of the mapping is structured as follows:

```
map <TYPE> <NAME> as <input|output|inout> <INDEX>:<SUBINDEX>
```

Where:

-
  ```
  <TYPE>
  ```

  The data type of the variable; U32, U16, U08, S32, S16 or S08.
- `<NAME>`
  The name of the variable as it is used in the user program.

-
  ```
  <input|output|inout>
  ```

The read and write permission of a variable: a variable can be declared as an `input`, `output` or `inout`. This defines whether a variable is readable (`input`), writable (`output`) or both (`inout`) and the structure by means of which it must be addressed in the program.

-
  ```
  <INDEX>:<SUBINDEX>
  ```

  Index and subindex of the object to be mapped in the object dictionary.

Each declared variable is addressed in the user program via one of the three structures: *In*, *Out* or *InOut* depending on the defined write and read direction.

NOTICE

A comment is only permitted above the respective mapping declaration in the code, not on the same line.

10.2.2 Example of mapping

Example of a mapping and the corresponding variable accesses:

```
// 6040h:00h is UNSIGNED16
map U16 controlWord as output 0x6040:00
// 6041h:00h is UNSIGNED16
map U16 statusWord as input 0x6041:00

// 6060h:00h is SIGNED08 (INTEGER8)
map S08 modeOfOperation as inout 0x6060:00

#include "wrapper.h"

void user()
{
  [...]
  Out.controlWord = 1;
  U16 tmpVar = In.statusword;
  InOut.modeOfOperation = tmpVar;
  [...]
}
```

10.2.3 Possible error at `od_write()`

A possible source of errors is a write access with the `od_write()` function (see NanoJ functions in the NanoJ program) of an object in the object dictionary that was simultaneously created as mapping. The code listed in the following is incorrect:

```
map U16 controlWord as output 0x6040:00
#include " wrapper.h"
void user()
{
 [...]
  Out.controlWord = 1;
  [...]
  od_write(0x6040, 0x00, 5 ); // der Wert wird durch das Mapping überschrieben
  [...]
}
```

The line with the `od_write(0x6040, 0x00, 5 );` command has no effect. As described in the introduction, all mappings are copied to the object dictionary at the end of each millisecond.

This results in the following sequence:

1. The `od_write` function writes the value `5` in object 6040$_h$:00$_h$.

2. At the end of the 1 ms cycle, the mapping is written that also specifies object 6040_h:00_h, however, with the value `1`.
3. From the perspective of the user, the `od_write` command thus serves no purpose.

10.3 NanoJ functions in the NanoJ program

With NanoJ functions, it is possible to call up functions integrated in the firmware directly from a user program. Code can only be directly executed in the protected area of the protected execution environment and is realized via so-called *Cortex Supervisor Calls* (Svc Calls). Here, an interrupt is triggered when the function is called, thereby giving the firmware the possibility to temporarily permit code execution outside of the protected execution environment. Developers of user programs do not need to worry about this mechanism – for them, the NanoJ functions can be called up like normal C functions. Only the *wrapper.h* file needs to be integrated as usual.

10.3.1 Accessing the object dictionary

void **od_write** (U32 index, U32 subindex, U32 value)

This function writes the transferred value to the specified location in the object dictionary.

index	Index of the object to be written in the object dictionary
subindex	Subindex of the object to be written in the object dictionary
value	Value to be written

NOTICE

It is highly recommended that the processor time be passed on with `yield()` after calling a `od_write()`. The value is immediately written to the OD. For the firmware to be able to trigger actions that are dependent on this, however, it must receive computing time. This, in turn, means that the user program must either be ended or interrupted with `yield()`.

U32 **od_read** (U32 index, U32 subindex)

This function reads the value at the specified location in the object dictionary and returns it.

index	Index of the object to be read in the object dictionary
subindex	Subindex of the object to be read in the object dictionary
Output value	Content of the OD entry

NOTICE

Active waiting for a value in the object dictionary should always be associated with a `yield()`.

Example

```
while (od_read(2400,2) != 0) // wait until 2400:2 is set
{ yield(); }
```

10.3.2 Process control

```
void yield()
```

This function returns the processor time to the operating system. In the next time slot, the program continues at the location after the call.

```
void sleep (U32 ms)
```

This function returns the processor time to the operating system for the specified number of milliseconds. The user program is then continued at the location after the call.

ms	Time to be waited in milliseconds

10.4 Restrictions and possible problems

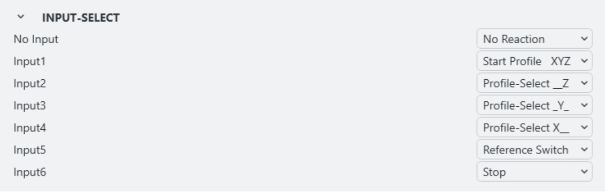
Restrictions and possible problems when working with NanoJ are listed below:

Restriction/problem	Measure
If an object is mapped, e. g., 0x6040, the object is reset to its previous value every 1 ms. This makes it impossible to control this object via the fieldbus or the *Plug & Drive Studio*.	Instead use `od_read` / `od_write` to access the object.
If an object was mapped as output and the value of the object was never defined before starting the *NanoJ program*, the value of this object may be random.	Initialize the values of the mapped objects in your NanoJ program to ensure that it behaves deterministically.
The array initialization must not be used with more than 16 entries.	Use `constant array` instead.
Too many local variables and arrays within functions may result in a stack overflow.	Declare the variables globally. Memory requirements are monitored already during compilation; errors do not occur at runtime.
Functions that are too deeply nested may result in a stack overflow.	Observe a maximum nesting depth of 2.
`float` must not be used with comparison operators.	Use `int` instead.
`double` must not be used.	
If a NanoJ program restarts the controller (either directly with an explicit restart or indirectly, e. g., through the use of the Reset function), the controller may fall into a restart loop that can be exited only with difficulty if at all.	
`math` or `cmath` cannot be included.	

11 NanoJ App

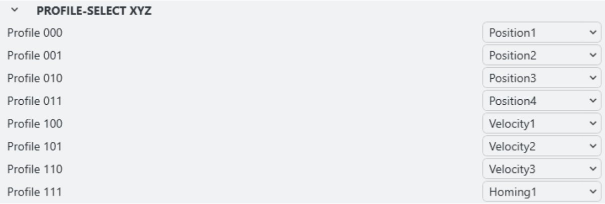
With the *NanoJ App* you can quickly have a NanoJ program created for controlling via digital/analog inputs for standalone operation.

You find the module *NanoJ App* in every Project template (with the exception of CSL3, because this product does nto support NanoJ). With the supplied default *Settings* and the *Configuration* you assign to an input combination.



INPUT-SELECT

No Input	No Reaction
Input1	Start Profile XYZ
Input2	Profile-Select _Z
Input3	Profile-Select _Y_
Input4	Profile-Select X__
Input5	Reference Switch
Input6	Stop



PROFILE-SELECT XYZ

Profile 000	Position1
Profile 001	Position2
Profile 010	Position3
Profile 011	Position4
Profile 100	Velocity1
Profile 101	Velocity2
Profile 110	Velocity3
Profile 111	Homing1

1. Under **INPUT SELECT** choose a function for each of your product's digital inputs: Start/Stop a motion profile, use the input as switch or release/enable or for profile selection (bitwise with one or more inputs, depending on the number of profiles).
2. Under **PROFILE SELECT** define the operation mode for each motion profile.
3. In the following configuration tabs enter the settings (speed, target etc.) for the specific operation mode (homing, position, velocity).
4. Save and click **Generate App**.

12 Imprint, versions

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

Document	Changes	PNDS3
1.0.0 (06/2022)	Edition	V1.3.0
1.0.1 (11/2022)	New software version V1.4.0, new firmware FIR-v2213	V1.4.0
1.1.0 (11/2023)	New software version V1.5.2	V1.5.2
1.1.1 (12/2023)	Minor corrections	V1.5.3
1.2.0 (04/2024)	New software version V1.6.0 including installer. ■ New software version V1.6.0. requires .NET 8 Runtime. ■ New chapters Programming with NanoJ and NanoJ App.	V1.6.0