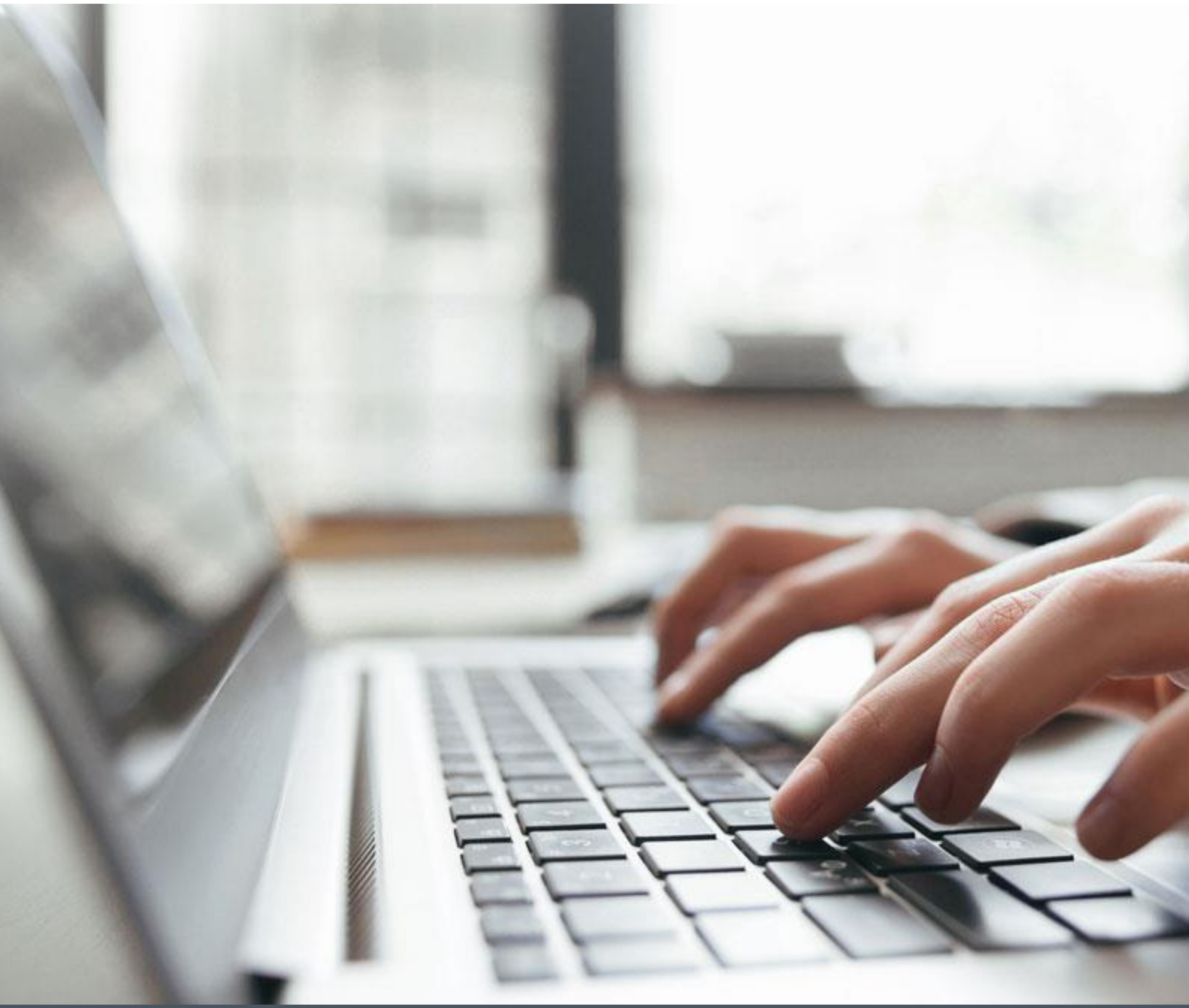


Application Note

How to use *Digital Inputs* in *NanoJ*

Version 1.0.1

www.nanotec.com

Contents

1 Intended use and audience

This application note shows you how to use the digital inputs of a Nanotec motor controller in a NanoJ program. You can find the corresponding NanoJ code template in the download folder.

Digital Inputs offers a NanoJ code template for assigning certain functions to electronic Nanotec motor controller input signals. To open and edit the template requires Plug & Drive Studio software. Both NanoJ and Plug & Drive Studio are for use with Nanotec products only, by trained specialists only.

2 Prerequisites

NOTICE

Malfunction from incompatibility! Plug & Drive Studio comes in various software versions. Find out and, if necessary, install the correct version for your Nanotec motor controller in advance.

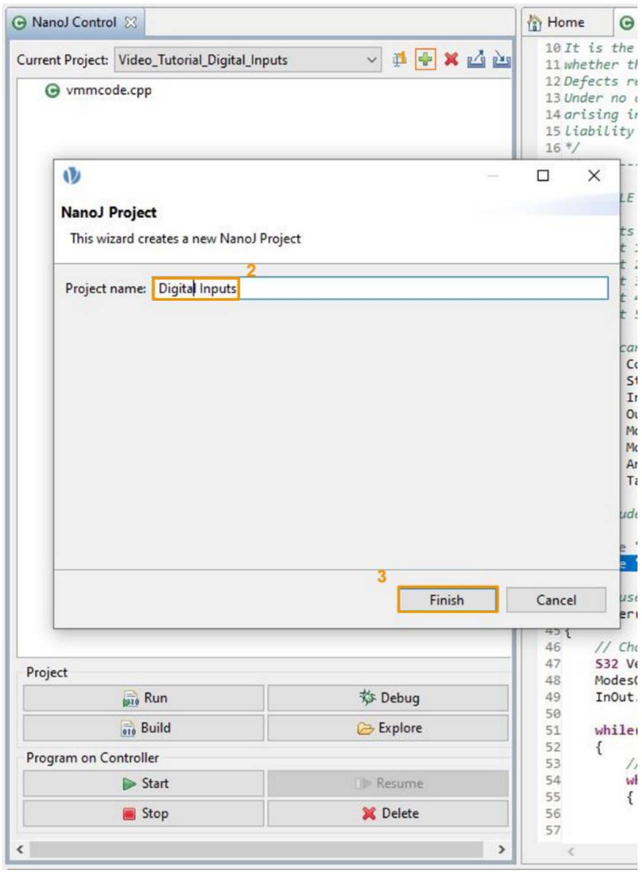
You must have the correct Plug & Drive Studio version installed on your computer:

1. Open the Nanotec software webpage.
2. Click on the *Plug & Drive Studio* buttons.
3. Browse *Compatible Products* to find out which version is compatible with your motor controller.
4. Download and install the latest compatible Plug & Drive Studio version on your computer.
5. If not done so yet: Also download the latest NanoJ V2 Library (`nanotec.h`).

3 Creating a new project in Plug & Drive Studio

Open the *NanoJ Control* tab and click on the "**+**" icon (1). A *NanoJ Project* tab pops up:

1. Assign a new project name (2).
2. Click on *Finish* (3) to close the tab.
3. Your new project is now created.



Current Project: Video_Tutorial_Digital_Inputs

NanoJ Project

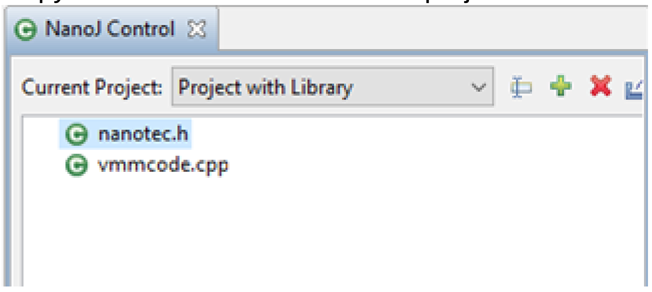
This wizard creates a new NanoJ Project

Project name: Digital Inputs

4 Including the nanotec.h library into your NanoJ project

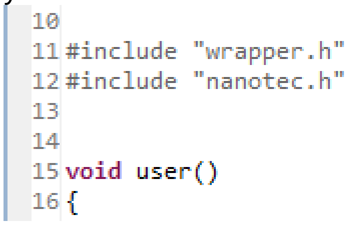
The Plug & Drive Studio installation folder does include `wrapper.h`. But you must download the NanoJ V2 library (`nanotec.h`) from our [knowledge base] and copy it into NanoJ:

1. Generate a new NanoJ project or open an existing one.
2. Copy the `nanotec.h` file into the project tree via drag & drop:



Current Project: Project with Library

3. To implement the NanoJ V2 library, add `#include wrapper.h` and `#include nanotec.h` to your code:



```

#include "wrapper.h"
#include "nanotec.h"


void user()
{
```

5 Using the code template for digital inputs in NanoJ

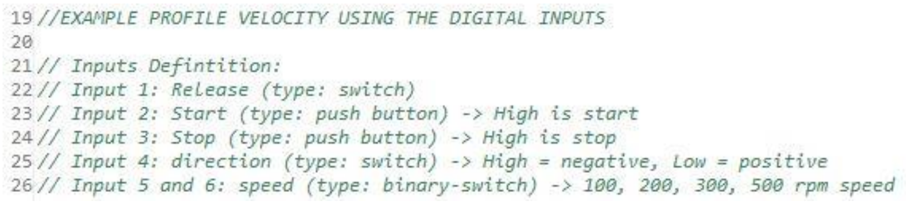
NOTICE

Variable signal level! Some Nanotec controllers provide digital input pins switchable between 5V and 24V. For correct digital inputs setup: Refer to the corresponding manual of your motor controller.

Digital Input code allows a routing in a NanoJ program depending on the input signals. After setting up the correct input signal level, you can assign the individual digital inputs.

5.1 Assigning digital inputs

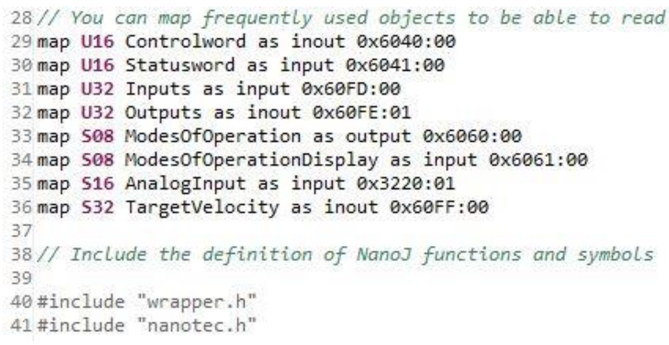
With the NanoJ code template, you can implement a release function (Input 1), a start and a stop push button (Input 2 and Input 3), as well as a direction switch (Input 4) to toggle between positive and negative direction. Also, you can use the Inputs 5 and 6 for selecting four speed levels (see the scope for our program in lines 21 to 26):



```
19 //EXAMPLE PROFILE VELOCITY USING THE DIGITAL INPUTS
20
21 // Inputs Defintition:
22 // Input 1: Release (type: switch)
23 // Input 2: Start (type: push button) -> High is start
24 // Input 3: Stop (type: push button) -> High is stop
25 // Input 4: direction (type: switch) -> High = negative, Low = positive
26 // Input 5 and 6: speed (type: binary-switch) -> 100, 200, 300, 500 rpm speed
```

5.2 Including libraries, mappings

For our case, we use the Nanotec NanoJ V2 library `nanotec.h` in our code template to provide basic functions to control our motor. To include the `nanotec.h` library, we must at least add the object mappings in lines 29 to 36 to our code. We also include the libraries `wrapper.h` and `nanotec.h`:

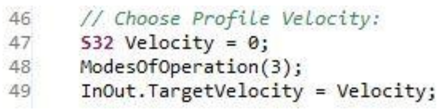


```
28 // You can map frequently used objects to be able to read or write them
29 map U16 Controlword as inout 0x6040:00
30 map U16 Statusword as input 0x6041:00
31 map U32 Inputs as input 0x60FD:00
32 map U32 Outputs as inout 0x60FE:01
33 map S08 ModesOfOperation as output 0x6060:00
34 map S08 ModesOfOperationDisplay as input 0x6061:00
35 map S16 AnalogInput as input 0x3220:01
36 map S32 TargetVelocity as inout 0x60FF:00
37
38 // Include the definition of NanoJ functions and symbols
39
40 #include "wrapper.h"
41 #include "nanotec.h"
```

5.3 Main program loop: void user()

5.3.1 Selecting a profile velocity, defining local variables

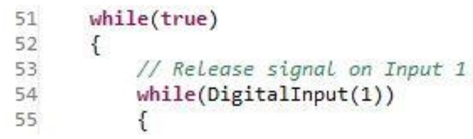
First, we select the *Profile Velocity* operation mode via mapped object 0x6060 (0x6060=3; line 48) `modesOfOperation(3)`. And we also define a variable to assign the velocity value to (`S32 Velocity = 0`). By assigning the velocity variable to the mapped object 0x60FF, we set the actual `TargetVelocity` for the motor as mapped in line 36:



```
46    // Choose Profile Velocity:
47    S32 Velocity = 0;
48    ModesOfOperation(3);
49    InOut.TargetVelocity = Velocity;
```

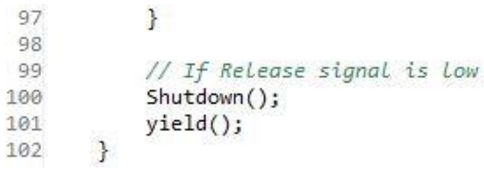
5.3.2 Implementing a release function (Input 1)

For Input 1 signals, we implement a release function. A high release signal **powers**, a low signal **un-powers**, the motor. The release function thus ensures the motor to run on a high release signal only.



```
51    while(true)
52    {
53        // Release signal on Input 1
54        while(DigitalInput(1))
55        {
```

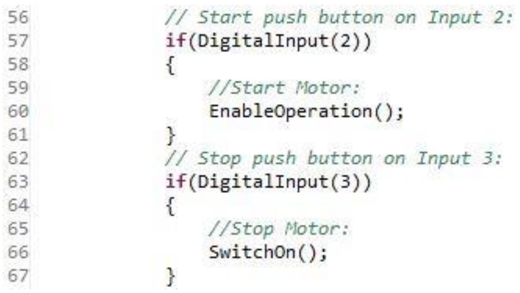
With a release signal set to low, we shut down the power state machine via `Shutdown()` function:



```
97            }
98
99            // If Release signal is low
100           Shutdown();
101           yield();
102   }
```

5.3.3 Implementing the start / stop push buttons (Input 2 & 3)

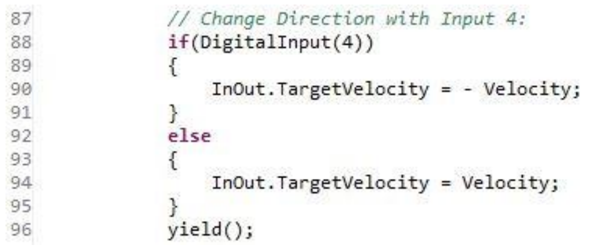
On Input 2, we implement a start push button. The motor starts to turn when Input 2 is high. We use the `nanotec.h` library function `EnableOperation()` to switch the power state machine to *operation enabled*. After that, we implement a stop signal on Input 3 that will stop the motor when Input 3 is set to high. To stop the motor, we use `SwitchOn()` to switch the power state from *machine* to *ready to switch on*:



```
56            // Start push button on Input 2:
57            if(DigitalInput(2))
58            {
59                //Start Motor:
60                EnableOperation();
61            }
62            // Stop push button on Input 3:
63            if(DigitalInput(3))
64            {
65                //Stop Motor:
66                SwitchOn();
67            }
```

5.3.4 Implementing a direction signal (Input 4)

With Input 4, we select the direction of the motor's motion. A high signal on Input 4 will turn the motor in the negative direction. A low signal on Input 4 entails a motion in the positive direction:

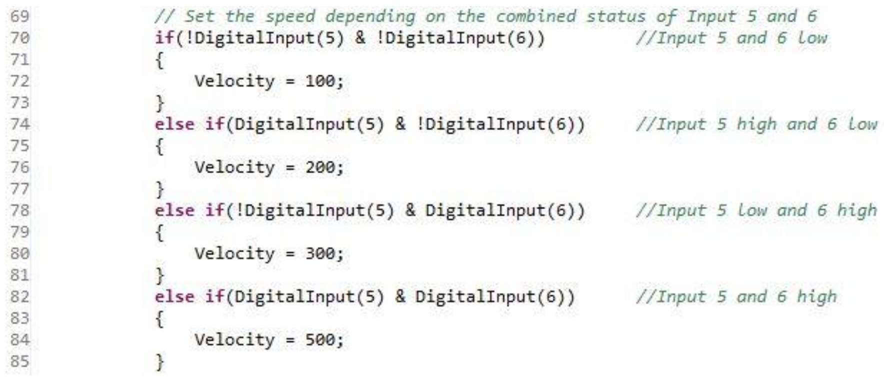


```
            // Change Direction with Input 4:
            if(DigitalInput(4))
            {
                InOut.TargetVelocity = - Velocity;
            }
            else
            {
                InOut.TargetVelocity = Velocity;
            }
            yield();
```

5.3.5 Selecting a binary-coded speed (Input 5 & 6)

With the Inputs 5 and 6, we will implement a binary coded selection between four velocity speeds. With two input signals, we can generate the four Bit code words `00, 01, 10, 11` (line 69 to 85):

- If both Inputs 5 and 6 are low, we will select a speed of 100 rpm (Bit code `00`).
- If Input 5 is high and input 6 is low, we will select a speed of 200 rpm (Bit code `10`).
- If Input 5 is low and input 6 is high, we will select a speed of 300 rpm (Bit code `01`).
- If both Inputs 5 and 6 are high, we will select a speed of 500 rpm (Bit code `11`):



```
69          // Set the speed depending on the combined status of Input 5 and 6
70          if(!DigitalInput(5) & !DigitalInput(6))        //Input 5 and 6 low
71          {
72              Velocity = 100;
73          }
74          else if(DigitalInput(5) & !DigitalInput(6))    //Input 5 high and 6 low
75          {
76              Velocity = 200;
77          }
78          else if(!DigitalInput(5) & DigitalInput(6))    //Input 5 low and 6 high
79          {
80              Velocity = 300;
81          }
82          else if(DigitalInput(5) & DigitalInput(6))     //Input 5 and 6 high
83          {
84              Velocity = 500;
85          }
```

Your code is finally implemented.

6 Liability

This Application Note is based on our experience with typical user requirements in a wide range of industrial applications. The information in this Application Note is provided without guarantee regarding correctness and completeness and is subject to change by Nanotec without notice.

It serves as general guidance and should not be construed as a commitment of Nanotec to guarantee its applicability to all customer applications without additional tests under the specific conditions and – if and when necessary – modifications by the customer.

The provided information does not replace datasheets and other product documents. For the latest version of our datasheets and documentations please visit our website at www.nanotec.com.

The responsibility for the applicability and use of the Application Note in a particular customer application lies solely within the authority of the customer. It is the customer's responsibility to evaluate, investigate and decide, whether the Application Note is valid and suitable for the respective customer application, or not.

Defects resulting from the improper handling of devices and modules are excluded from the warranty. Under no circumstances will Nanotec be liable for any direct, indirect, incidental or consequential damages arising in connection with the information provided.

In addition, the regulations regarding the liability from our Terms and Conditions of Sale and Delivery shall apply.

7 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