|  |  |
| --- | --- |
|  | 199.jpgHANcoder CANopen getting started guide |
|  | BitsChipsRC30Target_HANtune.jpg |
| Version 0.1  11/16/2015 | Getting Started Guide |
|  | A guide on how to get started with the CANopen functionality of the HANcoder toolset. |

HANcoder CANopen getting started guide

Getting Started Guide

Contents

[Versioning 2](#_Toc435510420)

[Introduction 3](#_Toc435510421)

[1 Overview of HANcoder’s CANopen functionality 4](#_Toc435510422)

[1.1 CANopen pre-configuration 4](#_Toc435510423)

[1.2 CANopen Simulink blocks 5](#_Toc435510424)

[2 Converting a software program into a CANopen node 7](#_Toc435510425)

[3 run-time configuration of the CANopen node 11](#_Toc435510426)

[3.1 CANopen Object Browser 11](#_Toc435510427)

[3.2 Changing a parameter in the object dictionary 12](#_Toc435510428)

[3.3 Storing changed object dictionary parameters in EEPROM 13](#_Toc435510429)

[3.4 Restoring the default object dictionary parameters in EEPROM 14](#_Toc435510430)

[Appendix A: Literature References 16](#_Toc435510431)

# Versioning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr** | **Date** | **Person** | **Changes** | **Status** |
| 0.1 | 16-11-2015 | F. Voorburg | First version | Concept |

# Introduction

Users of the HANcoder toolset develop their software program in the format of a Simulink model. HANcoder specific Simulink blocks give the developer access to microcontroller specific I/O, including Controller Area Network (CAN).

When adding CAN support to a software application, some consideration needs to be made regarding how the application inputs and outputs are communicated on the CAN bus. CANopen strives to be a standardized higher-layer communication protocol that defines how this communication can be performed. By using CANopen, is it no longer necessary to roll out a proprietary communication protocol, saving valuable engineering resources.

The HANcoder toolset takes this a step further. It contains a fully functional and pre-configured CANopen communication stack, allowing a Simulink model to be converted into a CANopen compatible node, within a few minutes.

It is this document’s goal to help the reader getting started with the CANopen functionality available in HANcoder. Through easy to follow step-by-step instructions, an example is presented. The scope of the example is converting an existing Simulink model into a generic CANopen I/O device.

The first chapter introduces the CANopen functionality as it is available in the HANcoder toolset and presents the CANopen specific Simulink blocks in the HANcoder toolset. Chapter two contains an example where these CANopen Simulink blocks are used to create a generic CANopen I/O device. The last chapter explains how the CANopen configuration can be changed during run-time and how to make these changes persistent by storing them in EEPROM.

It is assumed that the reader has a basic understanding of the CANopen protocol. He or she should be familiar with the terms such as: object dictionary, electronic datasheet, and process data objects. If not, then it is recommended to first study chapter 2 in [1].

# Overview of HANcoder’s CANopen functionality

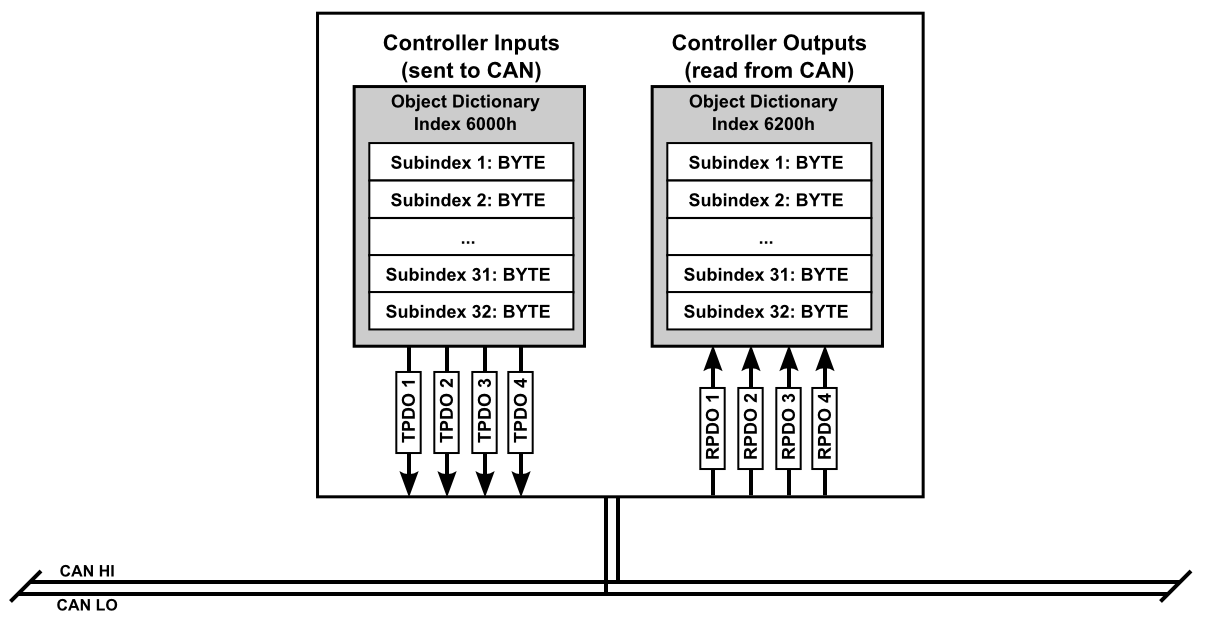
Users of the HANcoder toolset develop their software program in the format of a Simulink model. HANcoder specific Simulink blocks give the developer access to microcontroller specific I/O, including Controller Area Network (CAN).

When adding CAN support to a software application, some consideration needs to be made regarding how the application inputs and outputs are communicated on the CAN bus. CANopen strives to be a standardized higher-layer communication protocol that defines how this communication can be performed. By using CANopen, is it no longer necessary to roll out a proprietary communication protocol, saving valuable engineering resources.

The HANcoder toolset takes this a step further. It contains a fully functional and pre-configured CANopen communication stack, allowing a Simulink model to be converted into a CANopen compatible node, within a few minutes.

## CANopen pre-configuration

The CANopen pre-configuration contains 32 bytes for controller inputs at object dictionary index 6000h and 32 bytes for controller outputs at object dictionary index 6200h. These are general purpose input/output bytes for the application.



The 32 bytes at object dictionary index 6000h are mapped to the 4 transmit process data objects (TPDOs). The bytes at subindex 1 to 8 are mapped to TPDO 1, subindex 9 to 16 are mapped to TPDO 2, etc. By default, the TPDOs are transmitted on the CAN bus every 100ms.

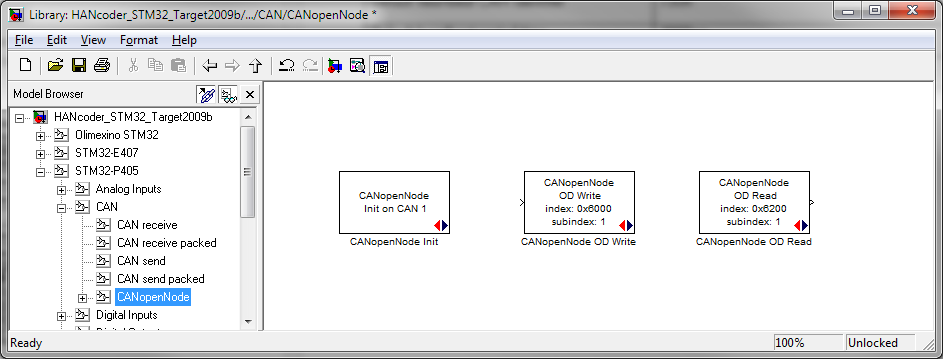
The 32 bytes at object dictionary index 6200h are mapped to the 4 receive process data objects (RPDOs). The bytes at subindex 1 to 8 are mapped to RPDO 1, subindex 9 to 16 are mapped to RPDO 2, etc. Upon reception of an RPDO, the associated values in the object dictionary are updated.

Additional pre-configuration details can be found in the node’s electronic datasheet (EDS), which is located in the same directory as this document and is named “**HANcoderNode.eds**”. The same information is present in a more human readable format in file “**HANcoderNode.html**”. The following table contains a summarized overview:

|  |  |
| --- | --- |
| Default CANopen Node Identifier | 48 (30h) |
| Default CAN communication speed | 125 kbit/sec |
| TPDO 1-4 CAN identifiers | 1B0h, 2B0h, 3B0h and 4B0h |
| RPDO 1-4 CAN identifiers | 230h, 330h, 430h, and 530h |
| Default heartbeat CAN identifier | 730h |
| Default heartbeat period time | 1000ms |
| Default node autostart setting | Autostart enabled |

## CANopen Simulink blocks

The CANopen functionality is available to the developer through the CANopen Simulink blocks:



There is a block “CANopenNode Init” for initializing the CANopen functionality, which needs to be added once to the Simulink model.

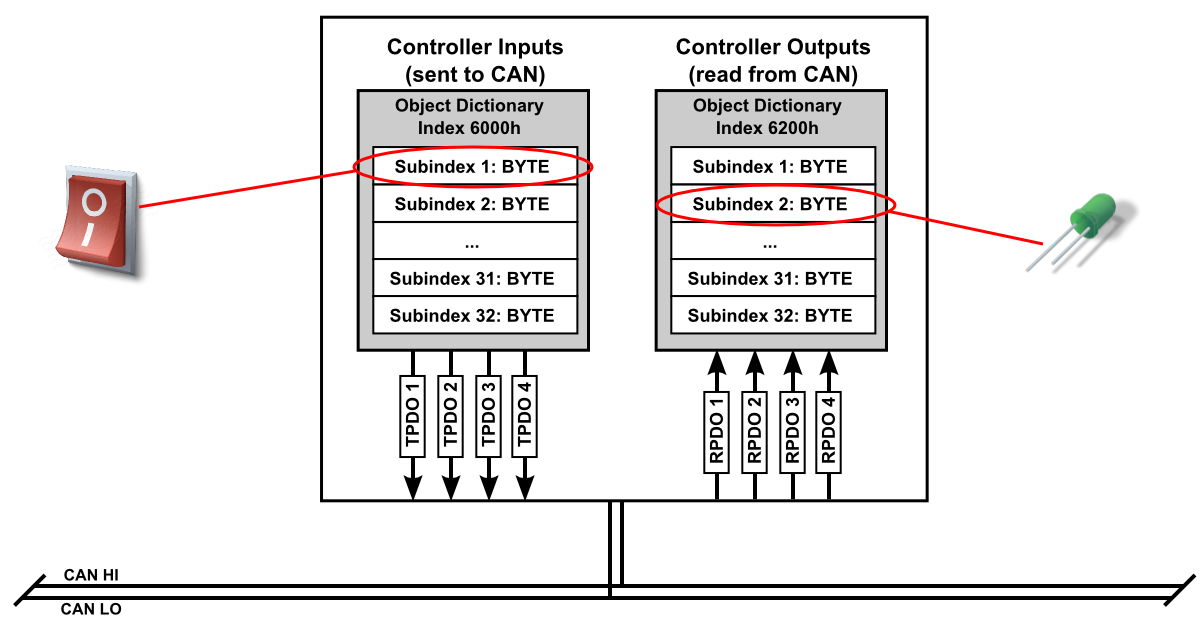
The block “CANopenNode OD Write” enables the developer to write new byte values into the object dictionary at index 6000h. The CANopen functionality then automatically takes care of placing these values into the associated TPDOs and transmitting them on the CAN bus.

The block “CANopenNode OD Read” enables the developer to read the byte values from the object dictionary at index 6200h. It contains the byte values that were received from the CAN bus in the associated RPDOs.

More details on inserting and configuring these Simulink blocks can be found in the next chapter.

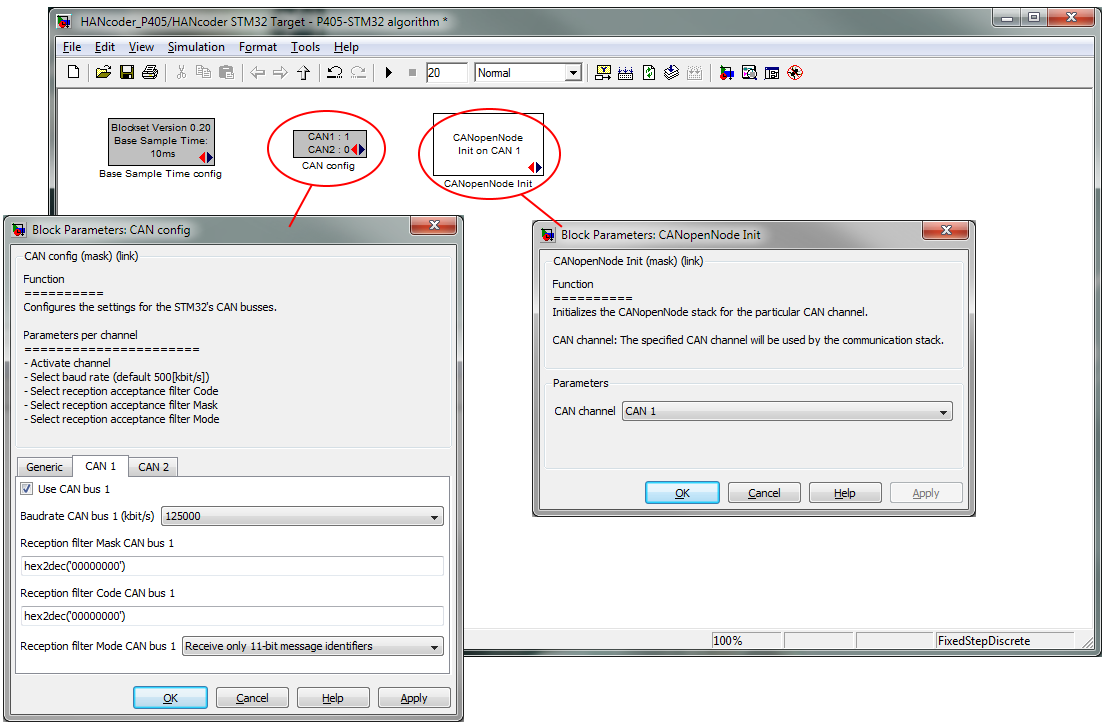
# Converting a software program into a CANopen node

Explaining the process of converting a software program into a CANopen node, is best achieved by means of an example. The example in this chapter is the creation of a CANopen I/O device:

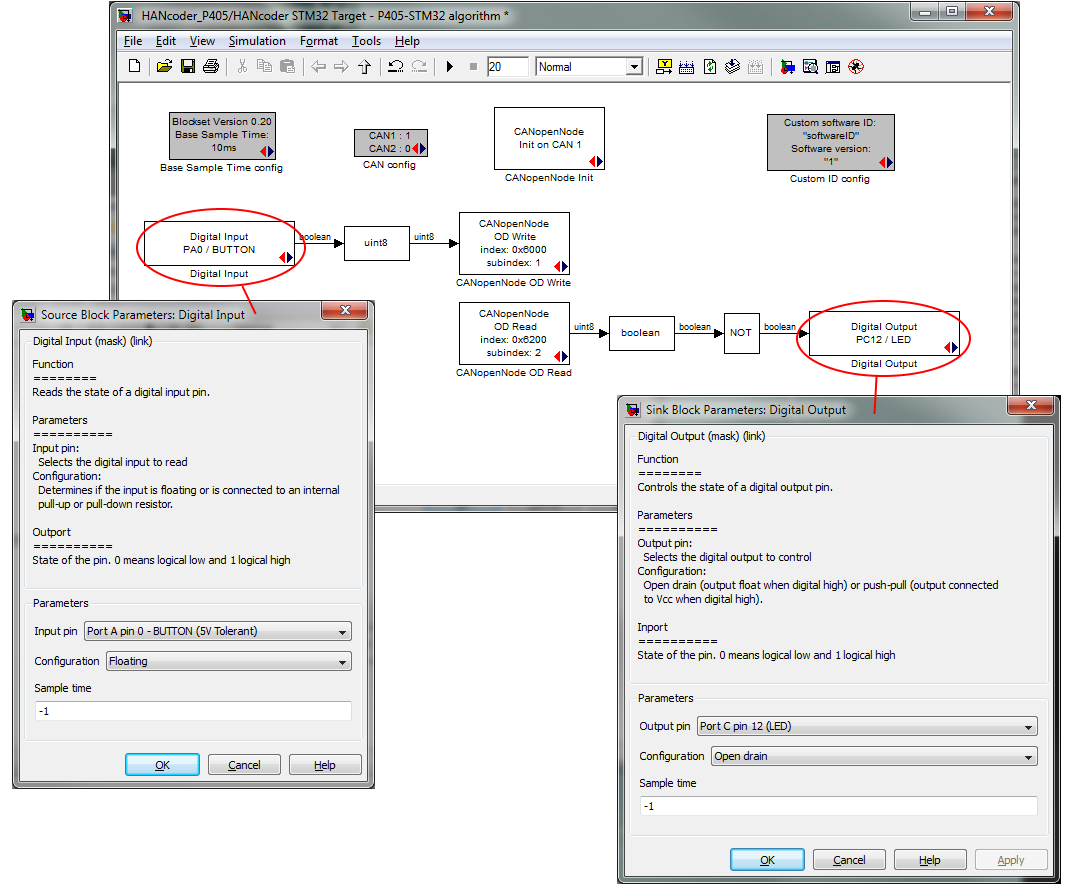


Such an I/O device could be used if another controller does not have sufficient I/O for the application and additional I/O is realized via CAN. The example CANopen I/O device has a switch (digital input) and an LED (digital output).

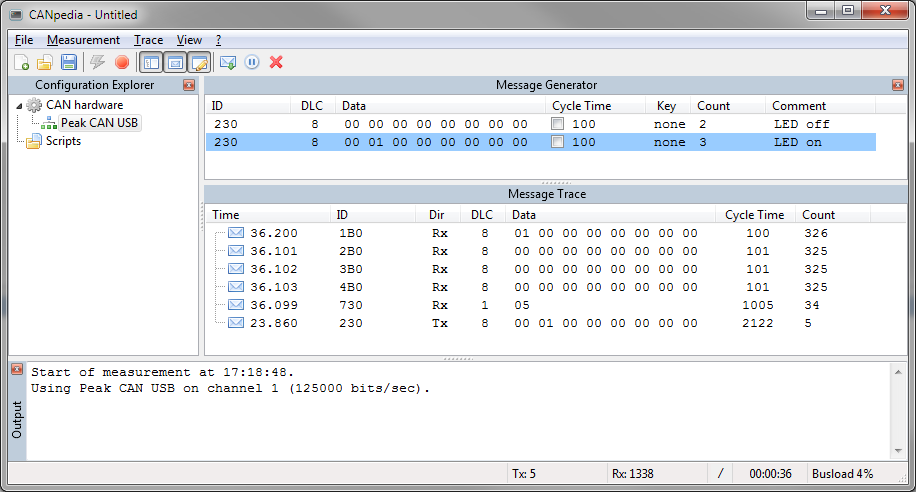
Note that the example is targeted towards an Olimex STM32-P405 board. The first step is adding a “CAN config” block and a “CANopenNode Init” block to the Simulink model. Configure the use of CAN channel 1 for 125 kbits/sec:



The second step is to read the state of the switch and store it in the object dictionary at index 6000h and subindex 1, and to update the LED based on the value stored in the object dictionary at index 6200h and subindex 2:



That’s all that needs to be done to create a CANopen compliant I/O device. If you now build the Simulink model (CTRL-B) and flash the resulting binary onto the Olimex STM32-P405, you will see the following message on the CAN bus:



Analysis of the messages on the CAN bus:

* The messages with identifier 1B0h, 2B0h, 3B0h and 4B0h are the TPDO1, TPDO2, TPDO3 and TPDO3 messages, respectively.
* The first byte in TPDO1 contains the state of the switch. It changes when the button on the Olimex STM32-P405 board is pressed/released.
* The message with identifier 730h is the node’s heartbeat message and is sent once a second. The value 05h in the first byte indicates that the node is operational.
* The message with identifier 230h is the RPDO1. Sending this message to the Olimex STM32-P405 on the CAN bus, allows another device on the CAN bus to control the LED. The state of the LED (on/off) is determine by the 2nd byte in the message.

Final note regarding the CAN communication speed: Keep in mind that the CAN communication speed value stored in the object dictionary at index 2102h, overwrites the value of the “CAN config” Simulink block.

# run-time configuration of the CANopen node

It is possible to further configure the CANopen configuration at run-time and then store the new configuration in EEPROM. Basically all writable values in the object dictionary can be changed. A few examples of commonly changed configuration parameters:

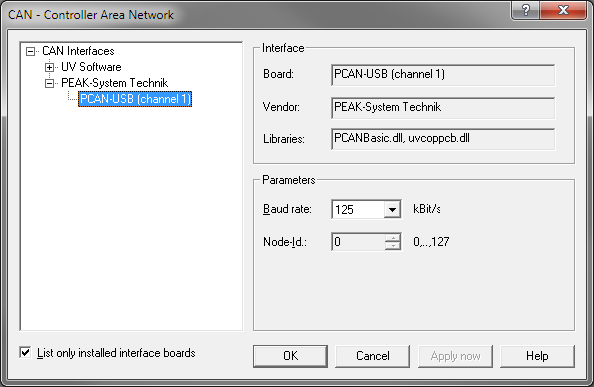
* CANopen Node ID (OD index 2101h).
* CAN communication speed (OD index 2102h).
* TPDO mapping, identifier and transmission period time (OD index 1800h-1803h and 1A00h-1A03h).
* RPDO mapping and identifier (OD index 1400h-1403h and 1600h-1603h).
* Heartbeat message transmission period time (OD index 1017h).

In this chapter an example is made for changing the CANopen Node ID from 48 (30h) to 50 (32h). The process of changing other parameters is identical to the example. It is just a matter of finding the parameter in the object dictionary. This is defined by the CANopen standard. Refer to [1] for additional details.

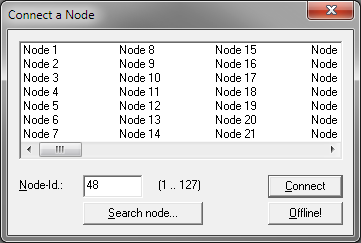
## CANopen Object Browser

The CANopen Object Browser tool (COPbrowser) is used for the run-time configuration of the CANopen node. This is free software from <http://www.uv-software.de/>. The installer was also added to the HANcoder toolset; it can be found at *\Installs\ COPbrowser\_v0.5.msi*.

When starting the COPbrowser, configure the PC’s CAN USB interface for the correct communication speed and click “OK”:



Next, select File -> Open from the menu and browse to the node’s electronic data sheet. This file is located in the same directory as this document and is called “HANcoderNode.eds”. After opening the EDS-file, the COPbrowser asks for the node’s current CANopen node ID. Enter “48” and click “Connect”:



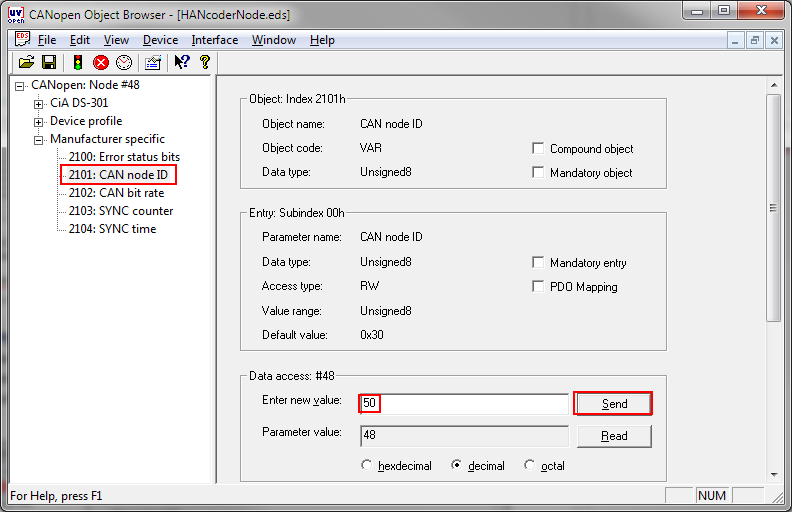
The COPbrowser is now connected to the CANopen node and has read/write access to everything in the node’s object dictionary via the CAN bus.

Note on the COPbrowser: It sometimes happens that a warning is generated after reading or writing a value from/to the object dictionary. In case this happens, simply retry the action that caused the warning.

## Changing a parameter in the object dictionary

The CANopen node ID is located in the object dictionary at index 2101h and subindex 0. The edit-box called “Parameter value” contains the current value of the parameter, which is 48.

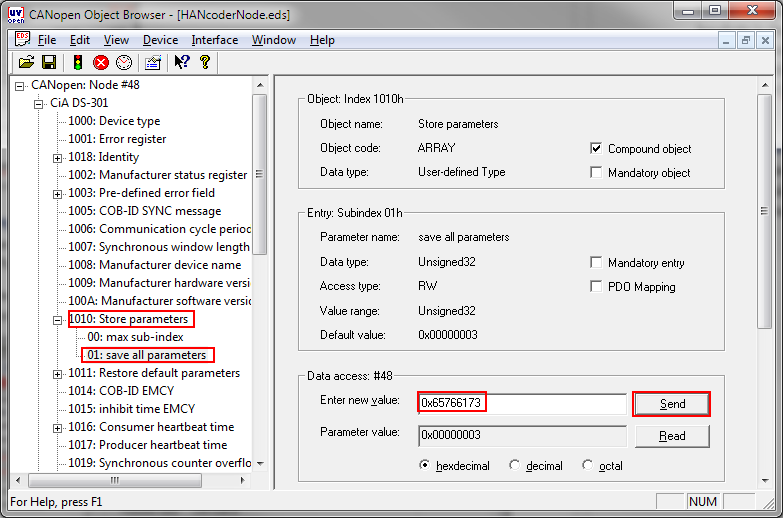
To change the CANopen node ID to 50, enter 50 in the edit-box called “Enter new value”, and click the “Send”-button:



Changing other parameters in the object dictionary follows the exact same procedure.

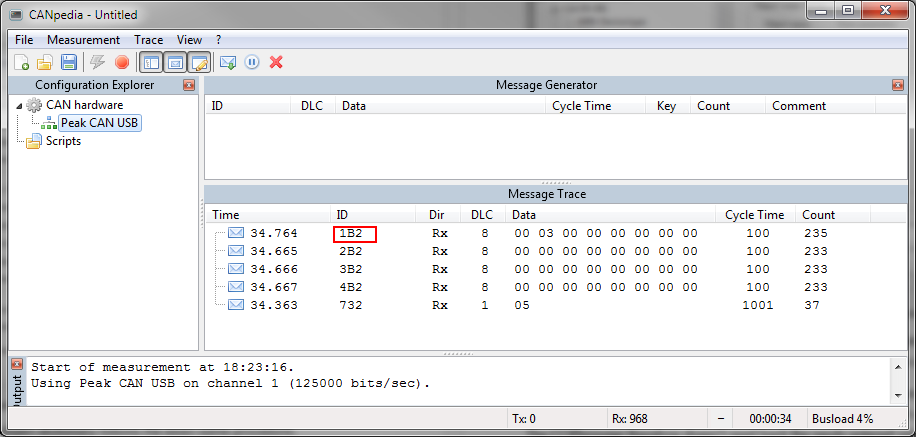
## Storing changed object dictionary parameters in EEPROM

The CANopen standard describes a method to store the current setting of the object dictionary parameters in non-volatile memory, such as EEPROM. To do this, the value 65766173h (ASCII for “save” with the “e” first) should be written to object dictionary index 1010h and subindex 1:



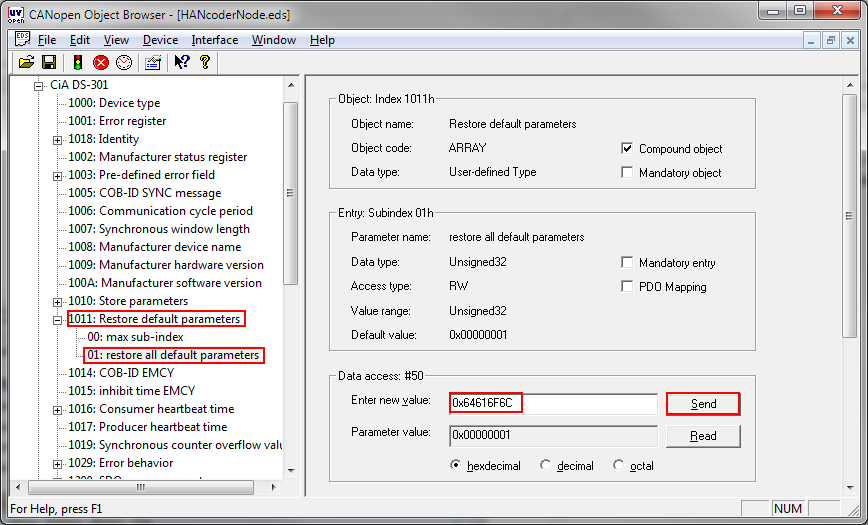
In the COPbrowser, select index 1010h and subindex 1 in the object dictionary. Next, enter the value 0x65766173 in the “Enter new value” edit-box and click the send-button.

All changes are now stored in EEPROM. After cycling the power (or using the reset button) of the Olimex STM32-P405, the new CANopen node ID is active. It can be verified by looking at the TPDO message identifier. TPDO1 is defined as 180h + Node ID. It used to be 1B0h for node ID 48 and it is now 1B2h for the new node ID of 50:



## Restoring the default object dictionary parameters in EEPROM

The CANopen standard describes a method to restore the default object dictionary parameters in non-volatile memory, such as EEPROM. By doing this, all previous made run-time parameter changes will be undone. To do this, the value 64616F6Ch (ASCII for “load” with the “d” first) should be written to object dictionary index 1011h and subindex 1:



In the COPbrowser, select index 1011h and subindex 1 in the object dictionary. Next, enter the value 0x64616F6C in the “Enter new value” edit-box and click the send-button.

After cycling the power (or using the reset button) of the Olimex STM32-P405, all default parameter values are used.

# Appendix A: Literature References

1. **Embedded Networking with CAN and CANopen**

by Olaf Pfeiffer, Andrew Ayre, and Christian Keydel

Published 2008 by Copperhill Technologies Corporation

<http://www.copperhillmedia.com>

ISBN 978-0-9765116-2-5