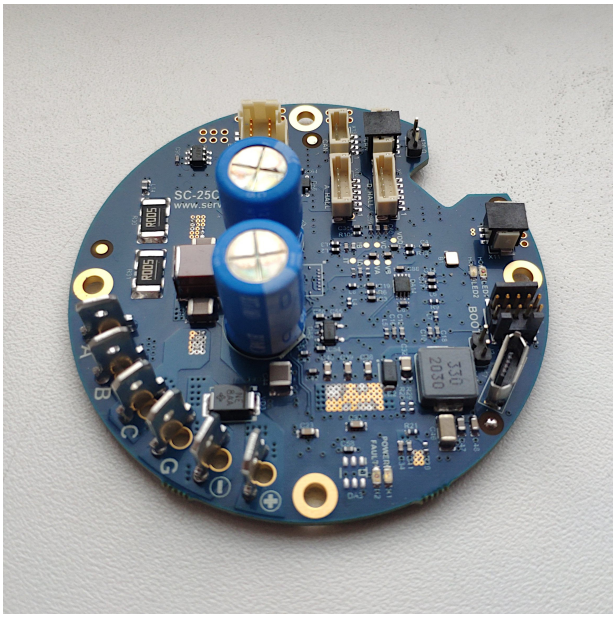
SERVOSILA SC-25 Brushless Motor Controllers

Programming Guide

Revision C



www.servosila.com/en/motion-control

Table of Contents

Introduction

Servosila SC-25 Brushless Motor Controllers provide an a CANopen control interface, and an open USB 2.0 interface, a virtual COM port, for receiving commands from a control computer/PLC as well as for sending telemetry back.

No proprietary drivers or SDKs are needed to connect the Servosila controllers to a Linux or a Windows 10/8 computer. Both Windows 10/8 and Linux come with prepackaged drivers and programming APIs sufficient to interface to Servosila controllers; this includes the popular Linux distributions of Debian and Ubuntu. Drivers for Windows 7 are available on request.

The controllers feature a built-in USB-to-CAN routing function, a “built-in USB2CAN adapter/dongle”, that enables PCs/PLCs to access an entire CAN network through connection to a single Servosila controller via its USB port. In other words, by connecting to a USB port of a single Servosila controller, a control computer/PLC gains access to the entire CANbus network to which the Servosila controller is connected to. Up to 16 controllers chained via their CANbus ports can be controlled by the same PC/PLC using the built-in USB-to-CANbus routing function. The limit is mainly due to throughput of a USB port of the SERVOSILA controller that acts as a USB2CAN dongle in addition to driving a motor; otherwise, all devices on a CANbus network are equally accessible to a control computer/PLC via a connection to a single Servosila controller. This includes non-Servosila devices such as GPS receivers or IMUs.

Up to 126 Servosila controllers can be connected to the same CANbus instance if a control computer/PLC has a hardware CANbus interface or a dedicated USB2CAN adapter. Linux SocketCAN API can be used to develop software that sends commands to Servosila controllers or receives telemetry back from the controllers.

Whenever a USB 2.0 interface (vs. a CANbus interface) is used to connect Servosila controllers to a Linux or a Windows 10/8 computer, the controllers appear as Virtual COM ports to both operating systems. A semi-standard text protocol called SLCAN is then used to send commands to the controllers through the virtual COM port as well as for receiving telemetry back.

In addition, most Linux distributions come with a special SLCANd daemon which makes Servosila controllers attached via USB Virtual COM ports appear as CANbus devices or as USB2CAN dongles. This daemon makes it possible to use the same Linux SocketCAN API whenever the Servosila controllers are connected to Linux via USB 2.0 or a CANbus interface (does not matter which one, the API is still SocketCAN API).

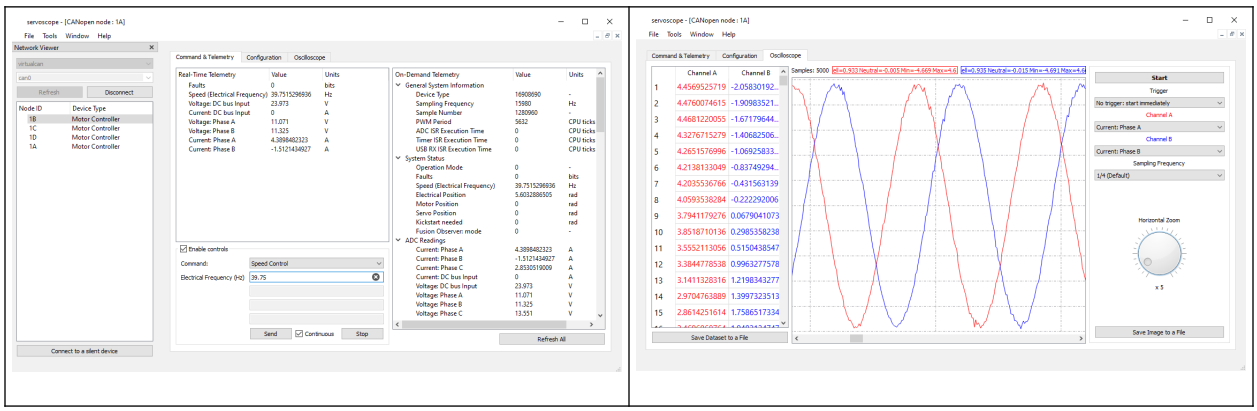
As Windows 10/8 systems do not have such a unified CANbus API, the ways for a control system’s software running on Windows to interact with Servosila controllers include:

- either through a virtual COM-port (USB 2.0) by writing to and reading from the port using a standard text protocol called SLCAN which is based on exchange of text strings formatted according to fairly simple open specification,
- or using a third-party USB2CAN adapter/dongle, and whatever programming API the adapter/dongle provides for Windows.

In general, user software for a control computer/PLC can be developed in any language that supports either Linux SocketCAN API or read/write operations for virtual COM ports on Windows 10/8 or Linux. This includes C/C++, Qt, Java, Python, MATLAB, LabView, and many other languages and packages.

The brushless motor controllers come with a graphical software tool called “Servoscope”. The tool provides means of configuring the controllers, displaying real-time telemetry coming via CANbus or USB, and sending commands down to the controllers.

The tool allows visualizing the telemetry data by plotting graphical charts. This is helpful when tracking down transient issues with electric drives.



The application runs on Windows 10/8 or Linux including the popular distributions of Debian and Linux. No drivers need to be installed on either of the platforms to run the tool. The tool connects to Servosila controllers via CANbus or USB interfaces.

The *Servoscope* application comes with a built-in software simulator of electric drives. In a real time way, the simulator numerically solves differential equations that define the dynamics of brushless motors, and provides a simulated CANbus/CANopen interface via Linux SocketCAN API for user control software being tested to connect to. The interface of the software simulator exactly matches the interface of a real Servosila controller coupled with a real electric drive.

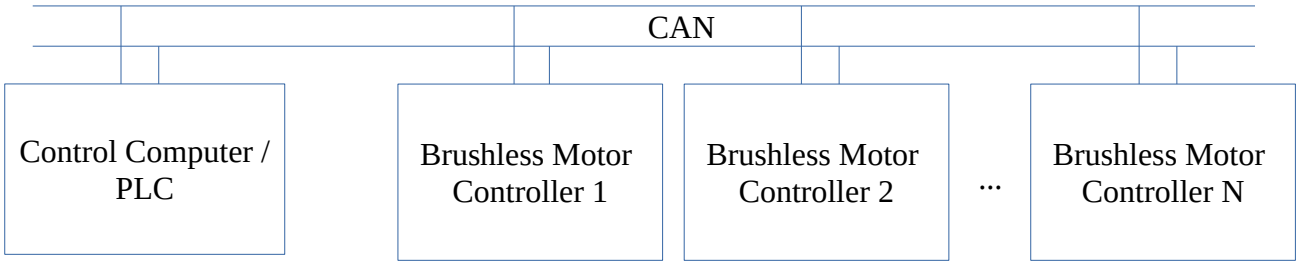
The simulator allows debugging of user software via CANbus/CANopen protocol prior to procuring a real hardware. Even if the hardware is available, it is still wise to first test your code against a simulator to avoid risking to burn a motor, break a gearbox, or cause an injury.

The simulator allows modeling the behavior of brushless motors or control systems under various conditions including dangerous or critical ones. The simulator is one of the software tools packaged into *Servoscope* application.

Network Architecture

CAN Network

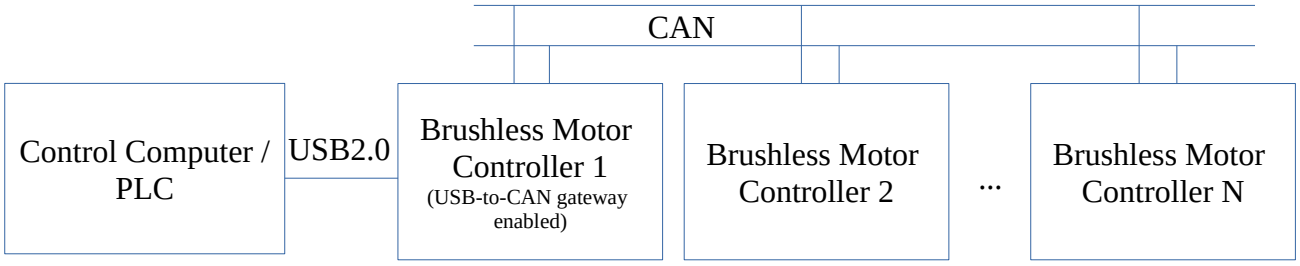
Servosila SC25 Brushless Motor Controllers are designed to connect to a control computer (PLC or Autopilot) via a CAN bus network. The control computer needs to have a physical CAN bus interface or some sort of an interface adapter to connect to the controllers this way.



Up to 126 Servosila brushless motor controllers can be connected to the same CAN bus instance if a control computer/PLC has a hardware CANbus interface. Linux SocketCAN API can be used to develop software that sends commands to Servosila controllers or receives telemetry back from the controllers.

USB-to-CAN Gateway + CAN Network

The other option is to use one of the Servosila SC25 Brushless Motor Controllers as a USB-to-CAN gateway (adapter). In such a network architecture, the control computer uses its USB 2.0 interface to connect to one of the brushless motor controllers acting as a “USB2CAN adapter”, while the controllers themselves interconnect via a CAN network.



In this network architecture, it is allowed to connect third-party CAN devices to the network of brushless motor controllers. In such a case, the control computer communicates to the third-party devices via the USB-to-CAN gateway function in the same way as it communicates to the brushless motor controllers themselves.

Up to 16 controllers chained via their CAN bus ports can be controlled by the same PC/PLC using the built-in USB-to-CAN routing function. The limit is mainly due to throughput of a USB port of the Servosila controller that acts as a USB2CAN dongle in addition to driving a motor.

Unique Node ID

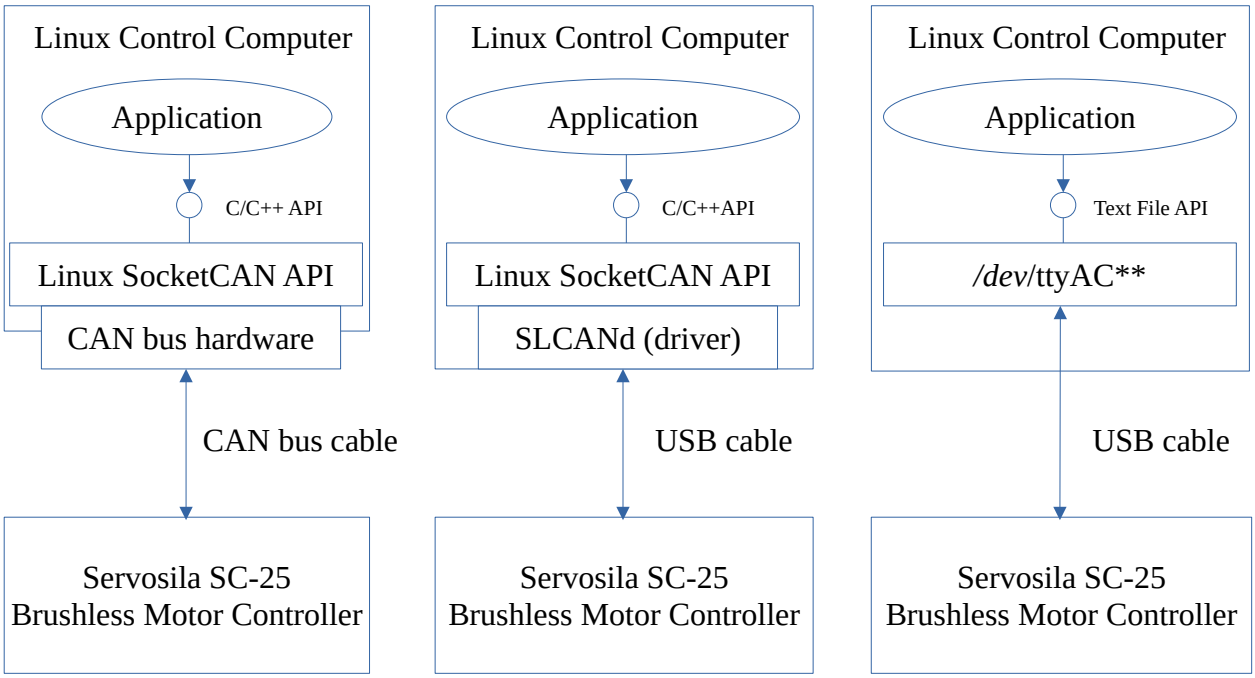
Each brushless motor controller is assigned a unique Node ID, an integer between 1 and 126. The Node IDs are used as an addressing scheme when sending messages to a particular device on a CANbus network[1], including when using the USB-to-CAN gateway function of the controllers.

Servosila SC-25 Brushless Motor Controllers come with factory-assigned sequential Node IDs. Since there are just 125 of possible Node IDs, the factory-assigned Node IDs of your set of controllers might not be unique. If that turns out to be the case, change the Node IDs of the controllers using the "Servoscope" configuration tool. Make sure you check uniqueness of Node IDs before connecting devices a CAN network.

1 Node IDs are defined by CANopen standard

Application Programming Interfaces

Linux APIs

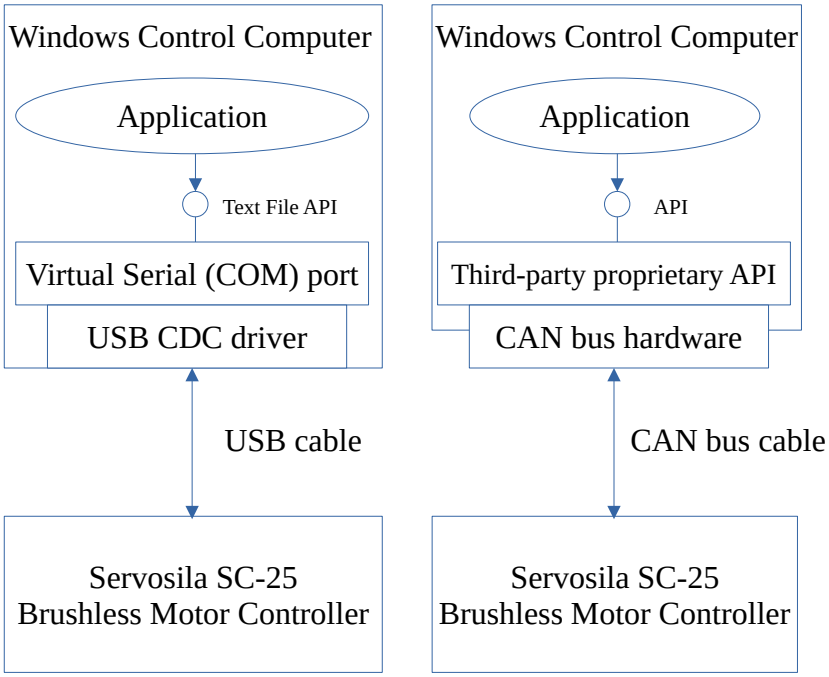


Linux SocketCAN API can be used to develop software that sends commands to SERVOSILA controllers or receives telemetry back from the controllers.

If the path taken is to develop the control software from scratch, there are samples of open source programs available from Servosila or can be found on the Internet that make use of Linux SocketCAN API. Otherwise, many off-the-shelf software products such as LabView support SocketCAN API out of the box, and come with an integrated CANopen stack.

Whenever a USB 2.0 interface (vs. a CANbus interface) is used to connect Servosila controllers to a Linux control computer, the controllers appear as virtual serial ports. A text protocol called SLCAN is then used to send commands to the controllers through the virtual COM port as well as for receiving telemetry back.

Windows APIs

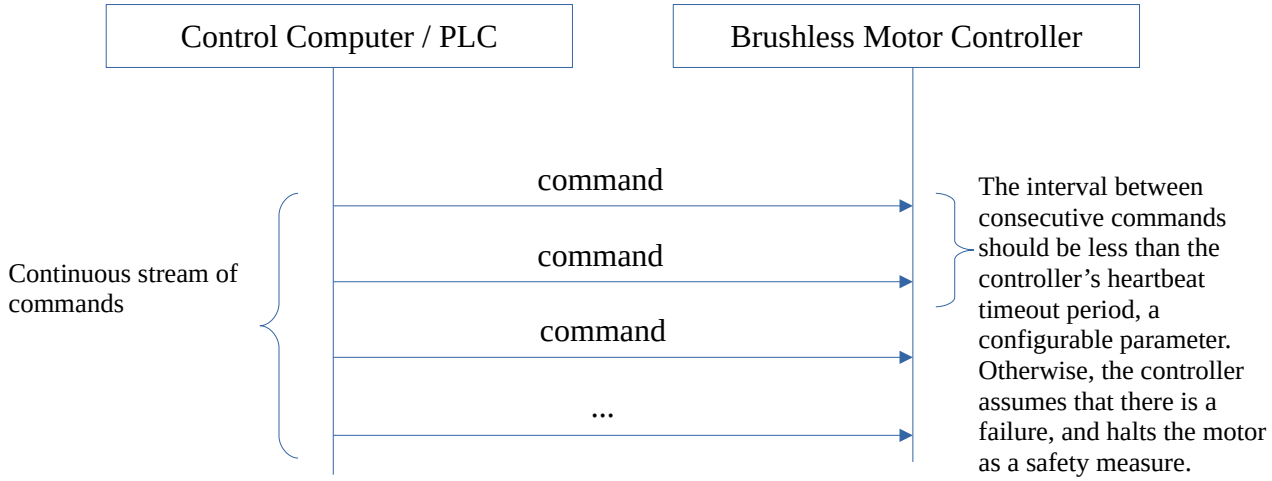


As Windows 10/8/7 systems do not have such a unified CANbus API, the ways for a control system's software running on Windows to interact with Servosila brushless motor controllers include:

- either through a virtual COM-port (USB 2.0) by writing to and reading from the port using a standard text protocol called SLCAN which is based on exchange of text strings formatted according to fairly simple open specification,
- or using a third-party USB2CAN adapter/dongle, and whatever programming API the adapter/dongle provides for Windows.

Message Flows

Sending COMMANDS to devices



The control computer is expected to *continuously* send commands[2] to each of the Servosila SC25 Brushless Motor Controllers on the network. If such a continuous stream of commands suddenly stops coming to a motor controller, the controller assumes that there is a network fault or a failure in the parent control system. In such a case, the controller stops its motor as a safety measure, and might enable a brake function depending on configuration settings.

How often the commands need to be sent depends on "heartbeat timeout" configuration parameter of each of the controllers. The parameter is set using the "Servoscope" software. The controller expects to receive at least one message from a control computer within the configured time interval. Otherwise, the controller times out, and halts the motor as a safety measure.

Typically, a control computer would continuously send a "Servo" command with a target position or an "Electronic Speed Control" command with a target speed, - or any other command understood by the controller. If a target position or a target speed has not changed since the previous time the command was sent, the control computer just re-transmits the previously sent command with the same target value.

Note that CAN bus does not guarantee delivery of messages (frames), and the controllers do not send back any acknowledgments confirming that a command has been received or executed. As such, the control computer shall rely solely on the method of continuous streaming of commands to ensure guaranteed delivery of the commands, assuming that most of the commands get through to the controllers.

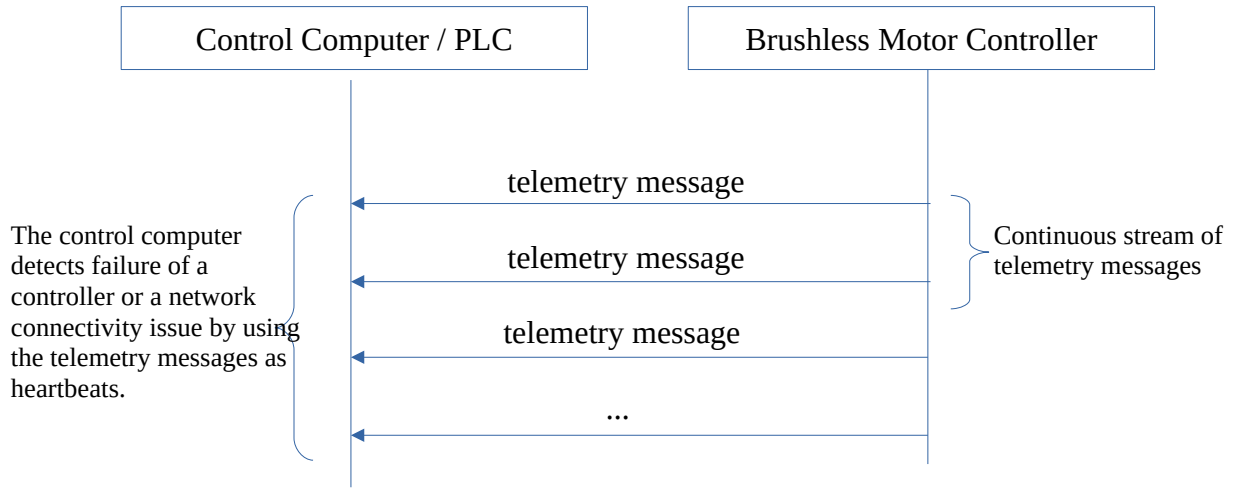
2 The commands are delivered as CANopen RPDO messages.

Sending the commands too often might flood the network with unwanted messages, thus there is a configuration parameter that controls the trade-off.

The flow of commands described in this section is applicable to both CAN and USB interfaces.

The format of the command messages is described later in this document. The format follows CANopen standard (RPDO).

Receiving TELEMETRY from devices



Servosila SC25 Brushless Motor Controllers continuously (periodically) send telemetry messages[3] back to a parent control system. The interval between telemetry messages is configured via the “Servoscope” software tool. The telemetry messages are sent asynchronously, and are not synchronized to commands or requests coming from a control computer.

The control computer should detect situations when telemetry messages stop coming from a particular device. This event typically indicates either a network fault or a failure of the device. The control computer generally should detect and handle such abnormal situations.

Note that CAN bus network does not guarantee delivery of telemetry messages. A running motor controller is not aware of whether or not the parent control system is actually receiving its telemetry. The control computer is not required to acknowledge receiving the telemetry messages.

The flow of telemetry messages described in this section is applicable to both CAN and USB interfaces.

The format of the telemetry messages is defined later in this document. The format follows CANopen standard (TPDO).

3 The telemetry data is delivered as CANopen TPDO messages.

Fault Acknowledgment

Whenever an internal fault is detected by a Servosila SC25 Brushless Motor Controller (e.g. an encoder error), the controller automatically powers off the motor as a safety measure, raises one or more "Fault Bits" flags in telemetry, and starts waiting for a "Reset" command to come from a parent control system. Until a "Reset" command comes, the controller ignores all other commands received from the parent control system. All configuration management functions[4] keep working as usual.

The parent control system is expected to continuously monitor the "Fault Bits" parameter delivered to it via a telemetry message. If the "Fault Bits" parameter is 0 (all bits are clear), then nothing needs to be done in response to such a telemetry message.

However, if one or more bits of the "Fault Bits" parameter indicate a fault, the parent control system is expected to send a "Reset" command back to the controller since the controller does not know on its own how to react to such a failure. The "Reset" command needs to be sent once the issue is rectified, or the control system decides that the fault can be safely ignored, and the electric drive is ready to re-start operation, may be in a different mode of operation.

Until the controller receives such a “Reset” command, it will keep the electric motor de-energized as a safety measure.

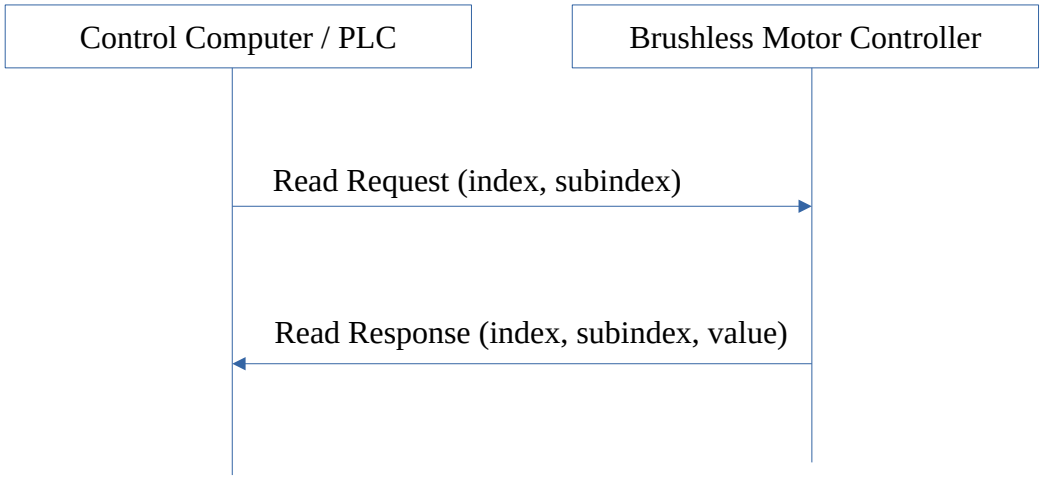
The fault acknowledgment flow described in this section is applicable to both CAN and USB interfaces.

Reading a Telemetry or a Configuration Parameter

The control computer has a way to read out[5] an arbitrary telemetry or configuration parameter stored in the device. To do this, the control computer sends a message with a request to a device. Upon receiving such a request, the device immediately replies back with a message that carries the most up-to-date value of the requested parameter.

4 CANopen SDO functions

5 CANopen SDO protocol



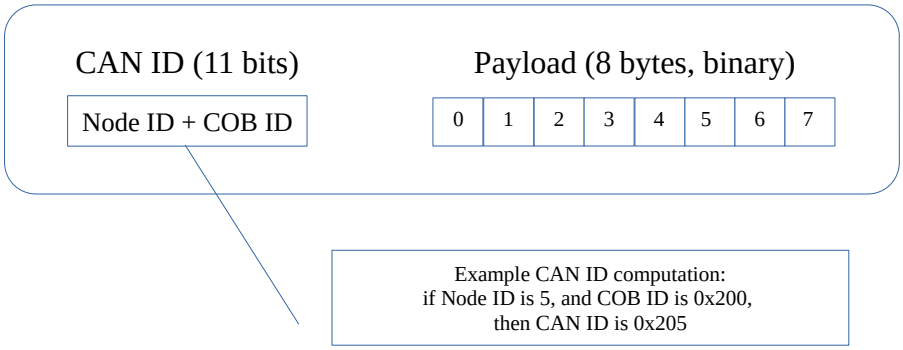
To identify a parameter of interest, the control computer uses an [Index:Sub-Index] pair of numbers. A list of all parameters, their indices, sub-indices and data types is given in a Servosila Device Reference document.

If the device does not find the requested parameter (e.g. an invalid index or sub-index), the device replies with an error message.

The formats of the request and response messages are defined later in this document. The formats follow CANopen SDO standard.

Encoding and Decoding

General message structure



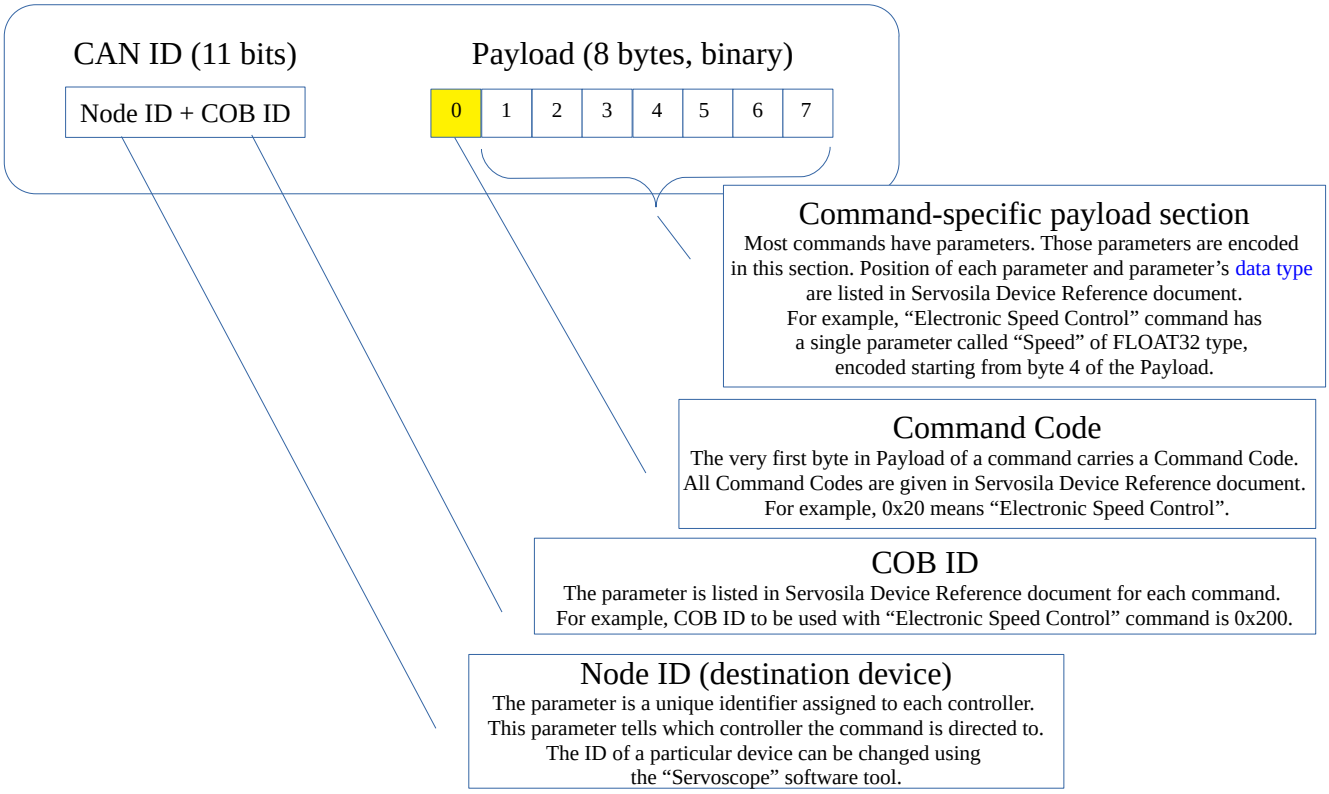
All types of messages consist of just two components, a **CAN ID** and a binary **Payload**.

- CAN ID is an addressing mechanism in a CAN bus network. CAN ID is defined as a sum of Node ID and a COB ID. To create a CAN ID, just take your Servosila device's unique Node ID and add a COB ID listed in Servosila Device Reference document for the given type of message.
- Node ID is a unique identifier assigned to every Servosila SC-25 Brushless Motor Controller, or to any other device in a CANopen network. The identifier assigned to a Servosila controller can be changed using the "Servoscope" software tool.
- COB ID is an integer number that defines what kind of message this is (a command, telemetry, or a configuration management message). The COB IDs are given in Servosila Device Reference document for each command or telemetry message, and are standardized by CANopen. There are just 8 possible COB IDs predefined in CANopen standard for commands and telemetry messages, and a few more COB IDs are defined for configuration management or network management.
- Payload is always an 8 byte array (binary). The contents of the payload depend on the kind of message being sent or received.

Supported APIs accept those two components (CAN ID and Payload) one way or another when sending or receiving messages (see sample C++ projects). Linux `cansend` command uses a '#' symbol to separate CAN ID and Payload components in a command line API call. The payload is encoded as a string of hexadecimal symbols, two hex symbols per each byte of Payload:

```
user@debian:~$ cansend can0 201#ABCD000000000000
```

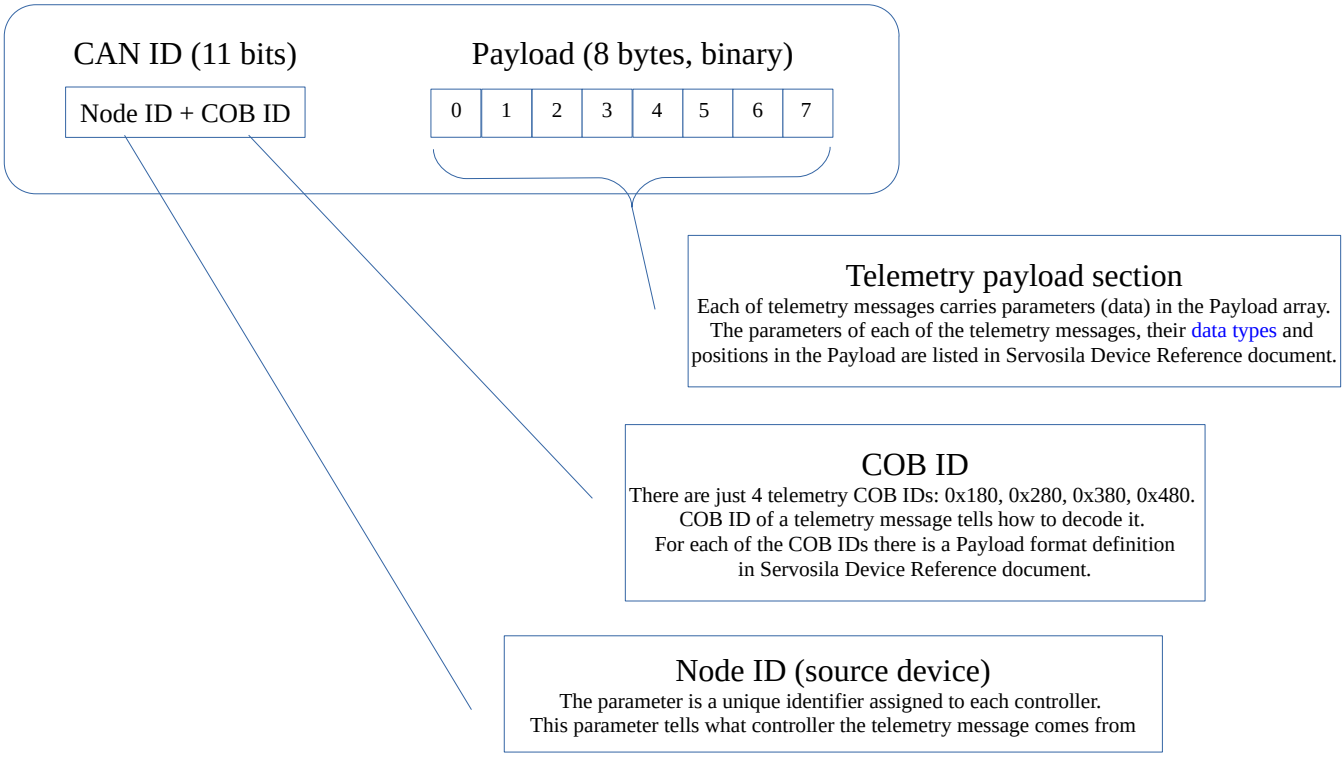
Encoding COMMANDS



See sample C++ projects `slcan-esc-command` and `canbus-esc-command` for details on how to encode and send out a command to a Servosila SC-25 Brushless Motor Controller.

Hint: press a "View" button in command section of the "Servoscope" software to view properly rendered commands.

Decoding TELEMETRY

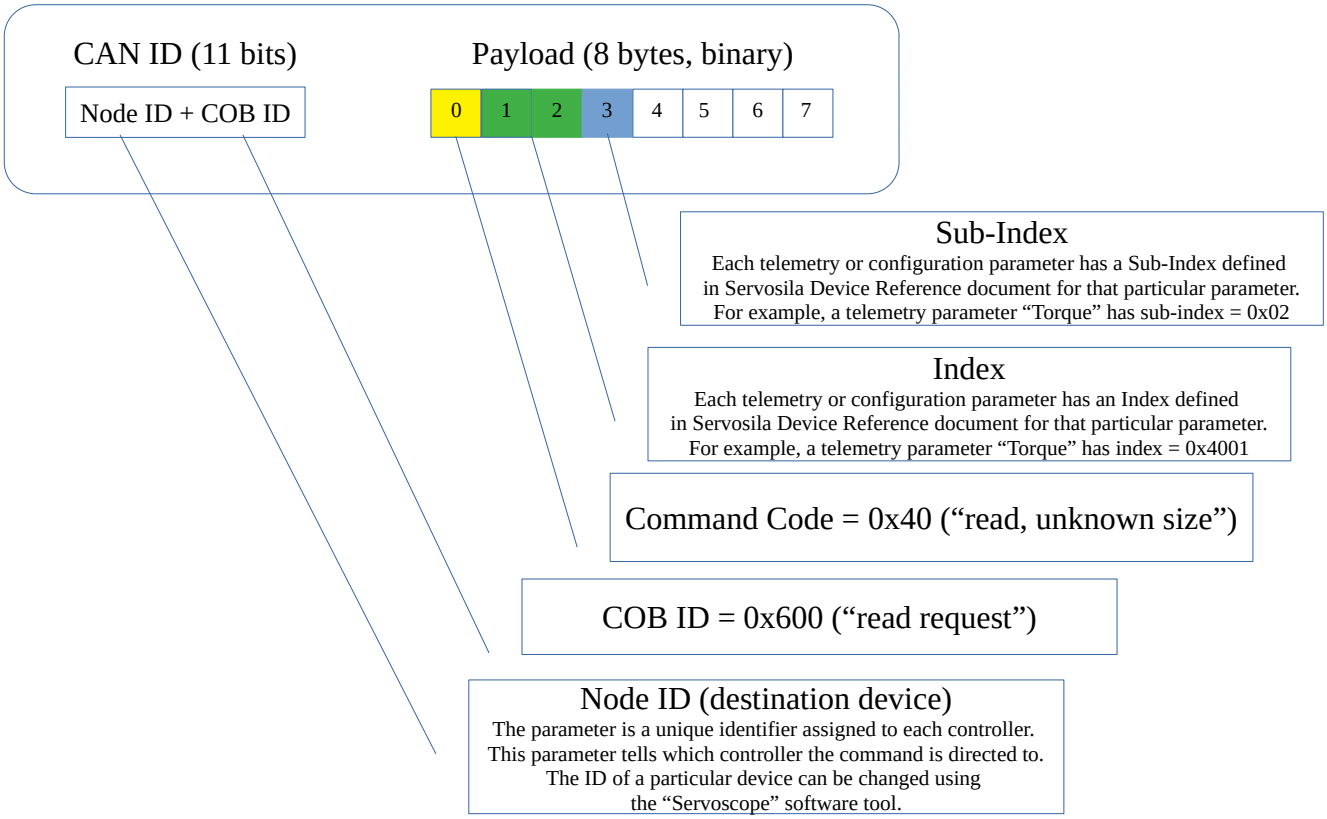


See sample C++ projects `slcan-telemetry` and `canbus-telemetry` for an example of how to receive and decode a telemetry message coming from a Servosila SC-25 Brushless Motor Controllers.

CAN ID Analysis & Decoding Example (C++)

```
//Extract Node ID from CAN ID
uint32_t extract_node_id_from_can_id(uint32_t can_id)
{
    //assert(can_id<=2047); //11bit only
    const uint32_t node_id = can_id & 127; //0000-1111111
    return node_id;
}

//Extract COB ID from CAN ID
uint32_t extract_cob_id_from_can_id(uint32_t can_id)
{
    //assert(can_id<=2047); //11bit only
    const uint32_t cob_id = can_id & 0x780; //1111-0000000
    return cob_id;
}
```

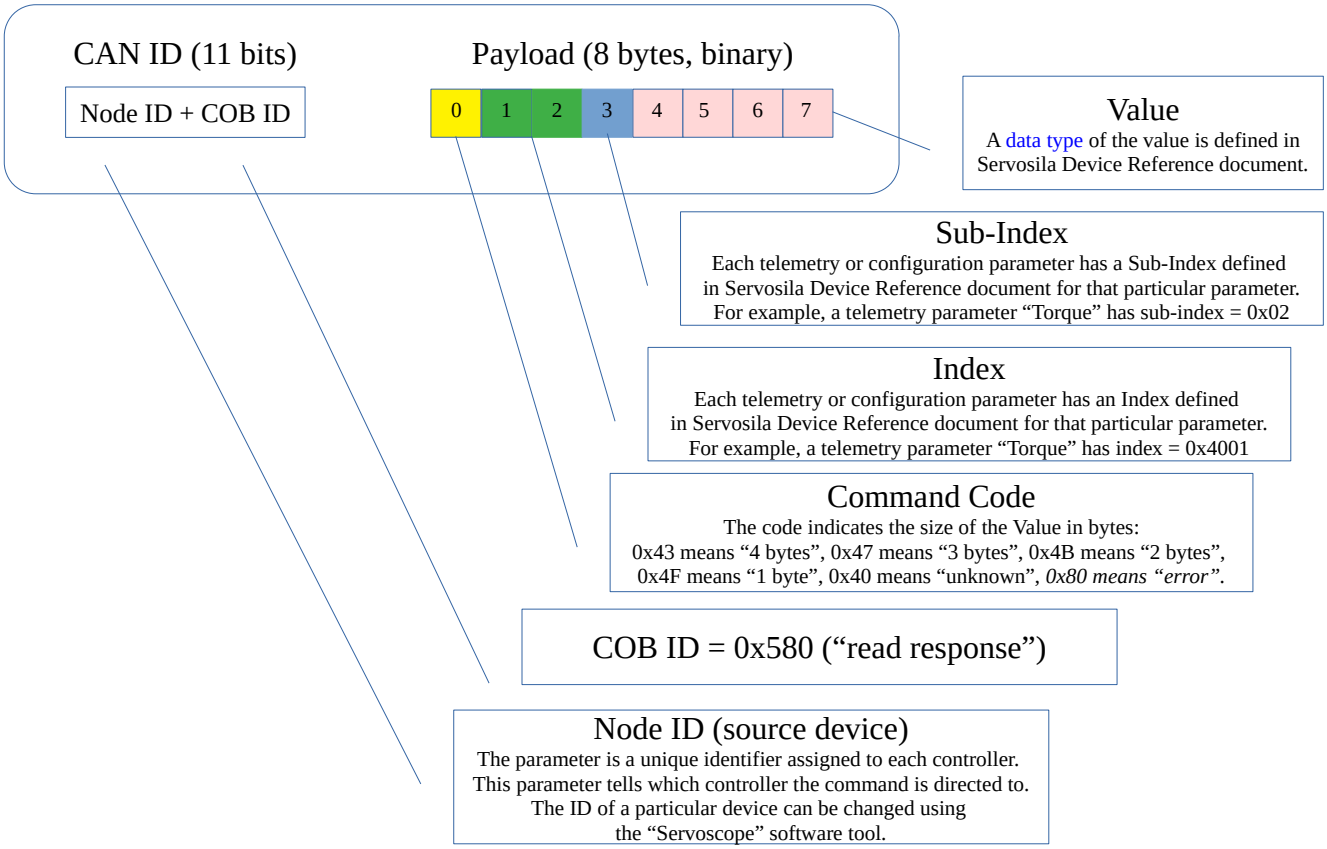
Encoding a READ REQUEST



Notes:

- Payload bytes 4-7 are not used in this message. The bytes are usually set to all zeros.
- The Index is encoded in the “little-endian format” (UINT16).
- Command Code = 0x20 means “write request”. This code can be used to programmatically change configuration parameters. The payload bytes 4-7 are then used to deliver a new value to be written into a configuration parameter. The type of the value must match a data type specified in Servosila Device Reference for the configuration parameter that needs to be programmatically updated.

Decoding a READ RESPONSE



Notes:

- Command Code = 0x80 means an error response, typically, caused by an invalid index or sub-index in a read request.
- The Index is encoded in the “little-endian format” (UINT16).

Data Types

Data Type	Size in bytes	Comments
BOOL	1	Same as UINT8.
UINT8	1	Byte.
INT8	1	Signed byte.
UINT16	2	Little-endian format.
INT16	2	Little-endian format.
UINT32	4	Little-endian format.
INT32	4	Little-endian format.
FLOAT32	4	IEEE 32-bit floating point value. Little-endian format.
FLOAT16	2	This is a proprietary floating number format defined by Servosila. See a description below.

FLOAT16

The FLOAT16 type is transmitted the same way as INT16, a signed integer. However, upon receiving, the signed integer value needs to be linearly scaled to range [-128, +128] to extract an encoded floating point value. This is a special data type that allows to fit this particular range of floating point numbers into 2 bytes instead of 4 bytes. This is just a data compression technique that allows to transmit more data in the limited 8byte-long CAN payload.

A C++ example of decoding or encoding a FLOAT16 number:

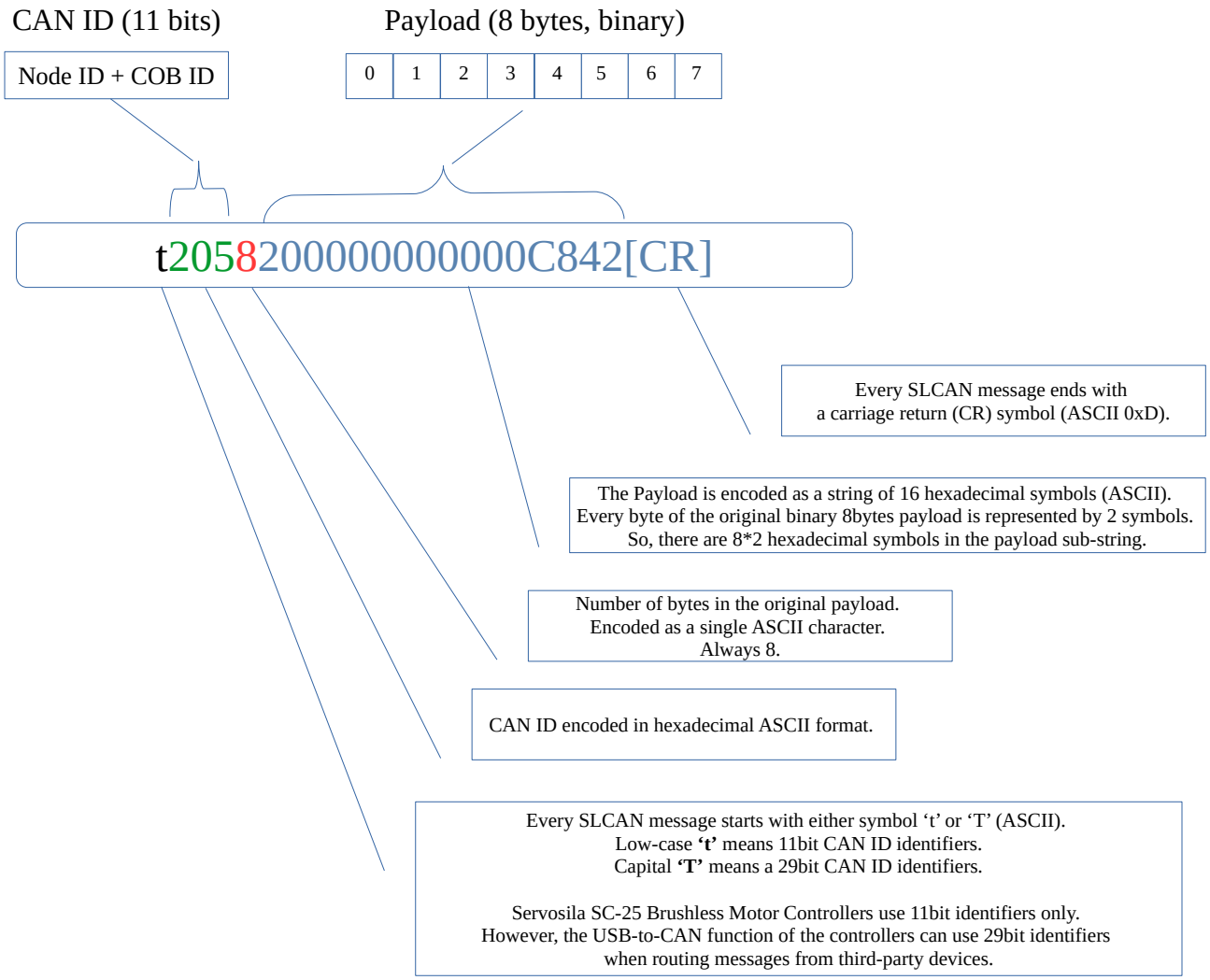
```
#define MAX_FLOAT16     ( 128.0f)
#define MIN_FLOAT16     (-MAX_FLOAT16)

float decode_float16 (int16_t bits)
{
        //de-scaling
        const float f32 = float(bits) * (MAX_FLOAT16 / 32767.0f);           //LINEAR SCALING FORMULA
        return f32;
}

int16_t encode_float16 (float f32)
{
        //clipping the value
        if(f32 > MAX_FLOAT16) f32 = MAX_FLOAT16;
        if(f32 < MIN_FLOAT16) f32 = MIN_FLOAT16;
        //scaling
        const int16_t bits = lroundf(f32 * (32767.0f / MAX_FLOAT16));       //LINEAR SCALING FORMULA
        return bits;
}
```

List of applicable COB IDs

COB ID (hex)	Used to transmit	Direction
0x200	commands	Control Computer → Device
0x300	commands	Control Computer → Device
0x400	commands	Control Computer → Device
0x500	commands	Control Computer → Device
0x180	telemetry	Device → Control Computer
0x280	telemetry	Device → Control Computer
0x380	telemetry	Device → Control Computer
0x480	telemetry	Device → Control Computer
0x580	read responses	Device → Control Computer
0x600	read requests	Control Computer → Device

SLCAN Text Protocol

Format of an SLCAN text message




```
user@debian:~$ echo -e "t2018AABB000000000000\r" >> /dev/ttyACM0
```

Serial Port Settings

Since the USB serial port is a virtual one, ***it does not matter*** what serial port settings are used.

If in doubt, use the following settings:

```
Baud rate: 115200, Data bits: 8, Stop bits: 1, Parity: None, Flow control: None.
```

Starting SLCANd daemon under Linux

Linux's pre-packaged SLCANd daemon allows using Linux SocketCAN API with USB interface of Servosila SC-25 Brushless Motor Controllers. Otherwise, the daemon is not required.

Use the following commands to start SLCANd daemon under Linux:

```
root@debian# slcand /dev/ttyACM0 can0
```

```
root@debian# ip link set up can0
```

Sample Projects (C++)

Project Name	OS	Interface	Language	Type of application	Dependencies	Description
`canbus-esc-command`	Linux	CAN	C++	command line	none	The sample shows how to send an Electronic Speed Control command via CAN bus (Linux SocketCAN API).
`canbus-telemetry`	Linux	CAN	C++	command line	none	The sample demonstrates how to receive and decode telemetry messages coming via CAN bus.
`slcan-esc-command`	Linux	USB	C++	command line	none	The sample shows how to send an Electronic Speed Control command via USB (virtual serial port) using SLCAN text protocol.
`slcan-telemetry`	Linux	USB	C++	command line	none	The sample demonstrates how to receive and decode telemetry messages coming via USB (virtual serial port) using SLCAN text protocol.
`MotorControlGUI`	Windows, Linux	USB	C++	GUI	Qt library	The sample is an end-to-end demonstration of a graphical user interface (GUI) for controlling an electrical drive. The sample displays telemetry coming from the controller and periodically sends commands to the controller.

The sample projects come with reusable helper C++ functions and classes that help encode or decode CAN, SLCAN and CANopen messages. Use a C++ compiler (GCC, MinGW, Visual Studio) and an IDE (Qt Creator, Code Blocks, Visual Studio) to compile and launch the samples.

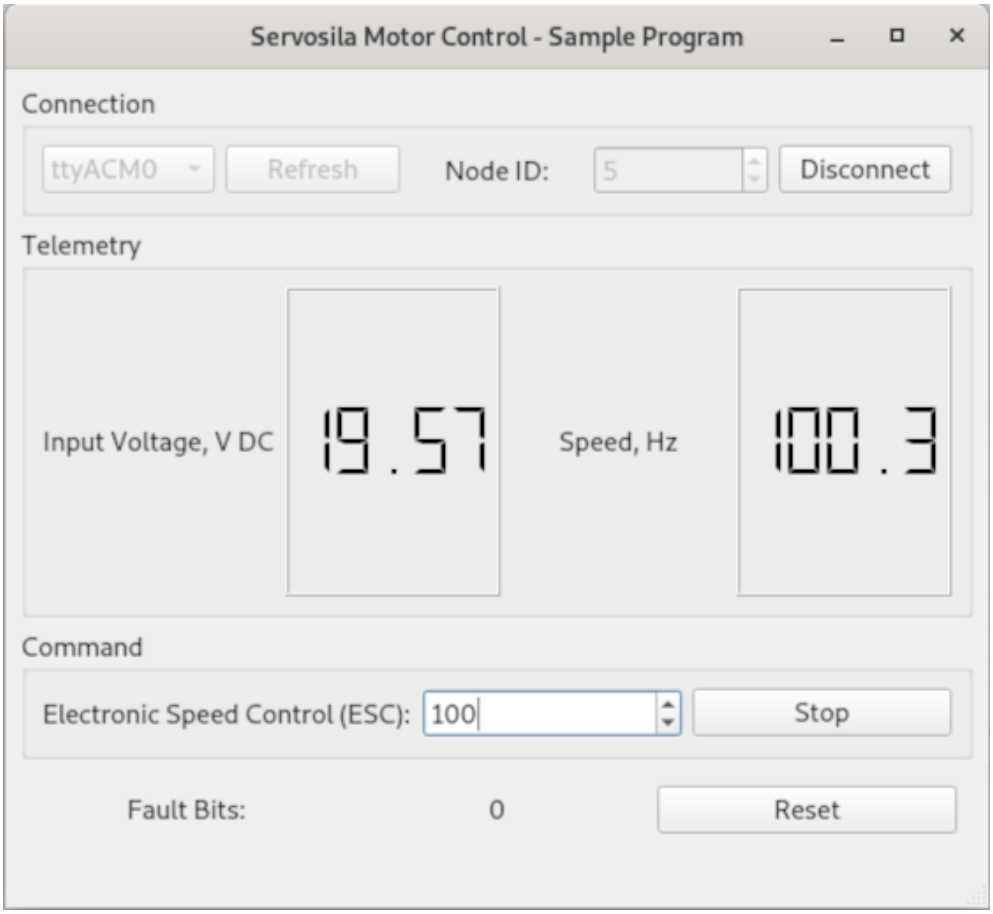


Figure 1: Screenshot of MotorControlGUI, an example project written in C++ with Qt library. The sample program runs on both Windows and Linux. The source code of the application is freely available for modification, reuse or distribution.

Servo drives designed around SERVOSILA SC-25C brushless motor controllers

YouTube: http://www.youtube.com/user/servosila

www.servosila.com/en/motion-control