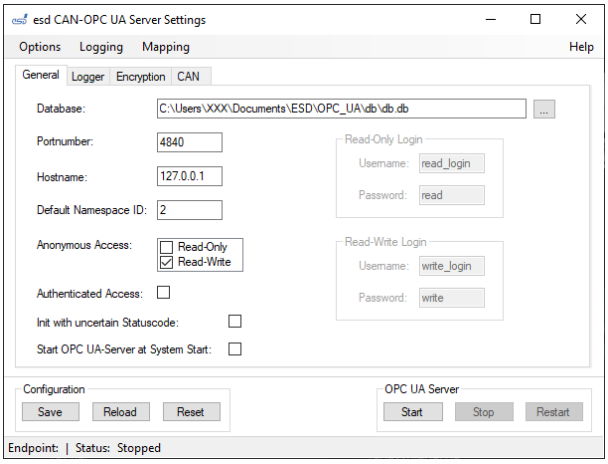
CAN-OPC UA Server

OPC UA Server for the Controller Area Network with Flexible Data Rate



Software Manual

to Product C.1103.31

esd electronics gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
http://www.esd.eu
Phone: +49 (0) 511 3 72 98-0 • Fax: +49 (0) 511 3 72 98-68

Notes

The information in this document has been carefully checked and is believed to be entirely reliable. esd electronics makes no warranty of any kind with regard to the material in this document and assumes no responsibility for any errors that may appear in this document. In particular, the descriptions and technical data specified in this document may not be constituted to be guaranteed product features in any legal sense.

esd electronics reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance, or design.

esd electronics gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

Tel.: +49-511-37298-0
Fax: +49-511-37298-68
E-Mail: info@esd.eu
Internet: www.esd.eu

This manual contains important information and instructions on safe and efficient handling of the CAN-OPC UA Server. Carefully read this manual before commencing any work and follow the instructions.
The manual is a product component, please retain it for future use.

Document Information

Document file:	I:\Texte\Doku\MANUALS\PROGRAM\CAN\C.1103.31_CAN-OPC_UA_Server\CAN-OPC_UA_Server_Software-Manual_en_14.docx
Date of print:	2023-04-19
Document-type number:	DOC0800

Software version:	from Rev. 1.0.3 or newer

Document History

The changes in the document listed below affect changes in the hardware as well as changes in the description of the facts, only.

Rev.	Chapter	Changes versus previous version	Date
1.0	-	First English software manual for CAN-OPC UA Server.	2021-05-04
1.1	3 / 5	Added an additional parameter description and solved minor errors. Improve *Quick Start* guide.	2021-09-02
1.2	4 / 6	Added an additional parameter to describe the endianness. Added an information box to describe the fifo-mode that is being used for CAN Frames in *Extended Frame Format* for CAN driver versions before 3.x.	2021-09-08
1.3	5	Added comment to the baudrate section.	2021-09-17
1.4	-	Modified Software version.	2021-10-06
	-	Editorial change only: Version number in footer corrected to 1.4	2023-04-19

Technical details are subject to change without further notice.

Classification of Warning Messages and Safety Instructions

This manual contains noticeable descriptions for a safe use of the CAN-OPC UA Server and important or useful information.

NOTICE

Notice statements are used to notify people on hazards that could result in things other than personal injury, like property damage.

NOTICE

This NOTICE statement contains the general mandatory sign and gives information that must be heeded and complied with for a safe use.

INFORMATION

INFORMATION

Notes to point out something important or useful.

Data Safety

This software can be used to establish a connection to data networks. This may allow attackers to compromise normal function, get illegal access or cause damage.
esd does not take responsibility for any damage caused by the device if operated at any networks. It is the responsibility of the device's user to take care that necessary safety precautions for the device's network interface are in place.

Number Representation

All numbers in this document are base 10 unless designated otherwise.

Table of Contents

List of Tables

List of Figures

1 Introduction

This document describes the configuration options and functionality of the CAN-OPC UA Server. The OPC UA *Server* offers an easy and efficient way to connect an esd CAN / CAN FD interface to an OPC UA *Server*. It provides a secure and platform-independent data exchange platform for industrial communication. The *Server* itself comes as a *Windows Service* and is easy to setup and control. It is able to link OPC UA *Nodes* and CAN messages defined by CAN identifier, bit position and length, together with a predefined datatype. Moreover, the *Server* supports multiple security policies and extensive access control. Within this document the CAN-OPC UA Server is referred to as *Server*.

1.1 Overview

Chapter 1 contains a general overview on the structure of this manual.
Chapter 2 provides information about the installation of the product.
Chapter 3 comprise a short guide how to start the *Server* out of the box.
Chapter 4 describes the mapping of the *Server Objects* and the linking process to a CAN interface.
Chapter 5 provides information about the *Server* control and describes the features within this product.
Chapter 6 describes the functionality of the *Server* and how it processes CAN frames.
Chapter 7 comprise some additional information about the product.
Chapter 8 contains order information.

1.2 Glossary

Abbreviation	Term
API	**A**pplication **P**rogramming **I**nterface
CAN	**C**ontroller **A**rea **N**etwork
CAN FD	**C**ontroller **A**rea **N**etwork with **F**lexible **D**ata Rate
CiA	**C**AN **i**n **A**utomation
OPC UA	**O**pen **P**latform **C**ommunications **U**nited **A**rchitecture
CSV	**C**omma-**S**eparated **V**alues
DB	**D**ata **B**ase
XLSX	Spreadsheet file format
TXT	Text Format
DER	**D**istinguished **E**ncoding **R**ules
SHA	**S**ecure **H**ash **A**lgorithm
RSA	**R**ivest–**S**hamir–**A**dleman
OOP	**O**bject-**O**riented **P**rogramming

1.3 Terminology

Within this manual you encounter the following terms:

CAN	**C**ontroller **A**rea **N**etwork is a serial bus system (also known as CAN bus) that was originally designed for use in vehicles but is now also used in automation technology. With the standardization of CAN FD, the original CAN is also referred as 'Classical CAN' by the *CAN in Automation* (CiA) to distinguish it from the enhanced standard.
CAN FD	**C**ontroller **A**rea **N**etwork with **F**lexible **D**ata Rate is an enhancement of the Classical CAN protocol. The main differences to standard CAN are the extended payload from 8 up to 64 bytes and the ability to send this payload with a higher data rate.
OPC UA	**O**pen **P**latform **C**ommunications **U**nited **A**rchitecture is a service-oriented and platform-independent data exchange standard for industrial communication developed by the *OPC Foundation*.
CAN Handle	Logical link between the application and a physical CAN port. An application can open several CAN handles to the same or to different CAN ports.
CAN ID	Identifier of a CAN message either in the Standard Frame Format (11-bit) or the Extended Frame *Format* (29-bit).
CAN Interface	A CAN interface is a dedicated *esd* hardware which is either connected to a local bus (PCI, USB, PC/104, etc.) of a CPU or remotely connected (Ethernet, Wireless, etc.) to a host system.
CAN Message	Logical unit which consists of a CAN ID and a payload either as data frame or as remote request frame.
CAN Port	The physical connector to a CAN bus which is handled by a CAN controller. Each port is assigned to an individual logical net number, also referred to as *Net ID*.
Data Frame	Frame which contains up to 8 bytes (64 bytes on CAN FD) of data either in Standard or Extended Frame Format.
RTR Frame	A frame transmitted to request the transmission of a data frame.
NTCAN-API	API of the cross-platform communication interface for *esd* CAN and CAN FD hardware.
NTCAN Event	Logical unit, which consists of an Event ID and a payload to describe the reason of the event (errors, warnings, state changes, ...).
DER	**D**istinguished **E**ncoding **R**ules is a subset of basic encoding rules. They are written in *Abstract Syntax Notation One (*ASN.1) and define the data as a stream of bits.
SHA	**S**ecure **H**ash **A**lgorithms are a group of cryptographic hash functions published by the *National Institute of Standards and Technology* (NIST). Normally, SHA is followed by a number which describes the digest lengths in bits.

RSA	**R**ivest–**S**hamir–**A**dleman is an asymmetric cryptosystem, which is widely used for secure data transmission.
Basic256	256-bit encryption. It supports various Sha hash algorithms for certificates.
Basic128Rsa15	128-bit encryption that uses RSA15 as a Key-Wrap. It supports Sha1 or stronger hash algorithms for certificates.
Basic256Sha256	256-bit encryption. It supports Sha256 or stronger hash algorithms for certificates.

2 Installation

2.1 Requirements

OS	*Windows 7* or later
CPU	x86/x64-based CPU with min. 1GHz
RAM	min. 7 MB of storage (configuration depended)
ROM	approx. 25 to 100 MB (installation depended)
CAN	*esd* CAN interface board supported by *Windows*

2.2 Setup

Setup is done via executing the setup file. This is usually named *CAN-OPC_UA_SERVER_V***.exe*. It offers an easy way to select the function and features you need.

The following options are available:

Server	This option contains the actual OPC UA Server as a *Windows Service*.
Control	This option provides the control tool for the *Server*, which can be used to change settings or start / stop the *Server*.
Parser	This option contains the parser, which is needed to parse between CSV files, which are used to map CAN messages and OPC *Nodes*, and DB files, which the *Server* uses as database. Moreover, this option provides a XLSM template as guidance.
Client	This option installs a small standalone OPC UA *Client* with a user interface, which can be used to connect to the *Server* and perform basic operations.
GenCert	This option provides a generator for unsigned certificates. It is required to install *OpenSSL* externally and add it to the system path in order to use it.
Help	This option contains the manual.
Sample	This option provides a sample configuration which can be used to get used to the *Server*. Moreover, it contains some unsigned sample certifications for test purposes.
Port	This option opens the TCP-Port 4840, which is the standardized port for the OPC UA protocol.

The most compact installation contains just the *Server*, the control tool, and the parser. These tools provide a user-friendly configuration and the possibility to parse between CSV and DB files, which the *Server* needs for configuration.

INFORMATION
Note that administrator rights are needed to execute this process successfully.

2.3 Removal

CAN-OPC UA Server can be removed by *Windows* ‘Software’ page (Start → Settings → System) or the ‘Uninstall’ link in the Start menu (default: Start → Programs → esd).

3 Quick Start

This chapter describes a fast and simple way to get started with the *Server*. However, to fully use the capabilities of the *Server*, it is recommended to read the whole manual before using the product in a real-world environment. This tutorial uses the sample configuration, which can be installed during the installation.

Starting the Server
After installing the *Server* successfully, the control tool is launched as a tray icon.

Figure 1: *Quick Start: Starting the Server (1)*

By double-left-clicking the icon, the *Settings Menu* is opened.
By default, the *Server* is configured on the standardized port 4840 and the IP address 127.0.0.1.

Start the *Server* by pressing the *Start* button in the lower right corner. When the status changes to *Running*, the *Server* has been started successfully.

Any OPC UA *Client* is able to discover and connect to the *Server* under the endpoint, which is displayed in the status bar on the bottom of the menu. However, by default there are no CAN interfaces and no user-defined OPC UA *Nodes* configured.
Press the *Stop* button to go on.

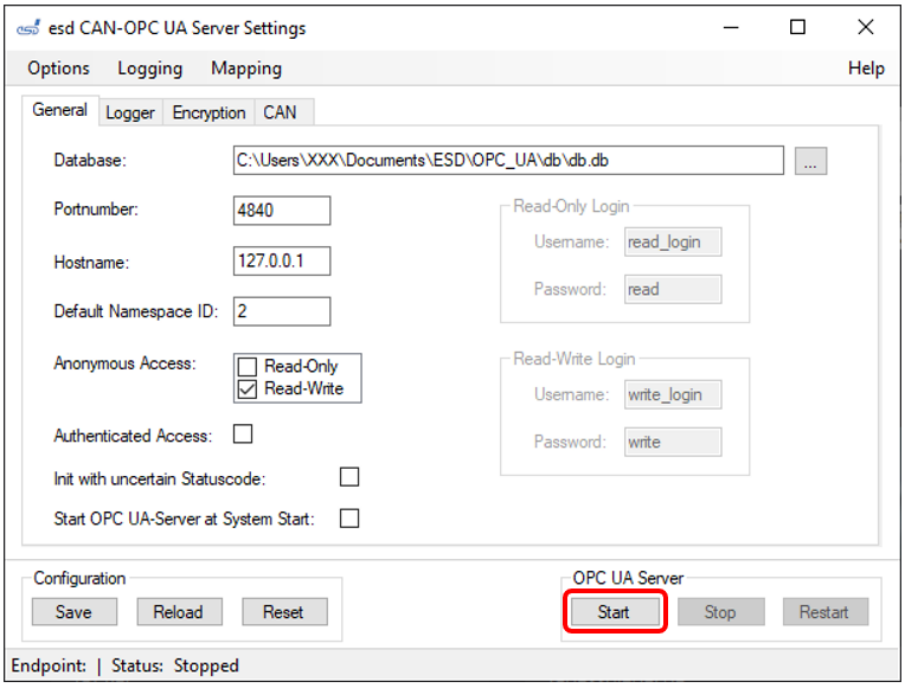


Figure 2: *Quick Start: Starting the Server (2)*

Configuration of CAN Interfaces
To configure a CAN interface, go to the Tab *CAN* and select one of the available CAN interfaces. Press the >> button to shift it to the used CAN interfaces.
After that, press the *Save* button in the lower left corner and confirm the dialog.
Now start the *Server* again.

When browsing the *Server*, there is a *Node* named after the net number of the CAN interface, which provides information about baud rate, bus state and statistic. However, there are still no user-defined OPC UA *Nodes* configured.
Press the *Stop* Button to go on.

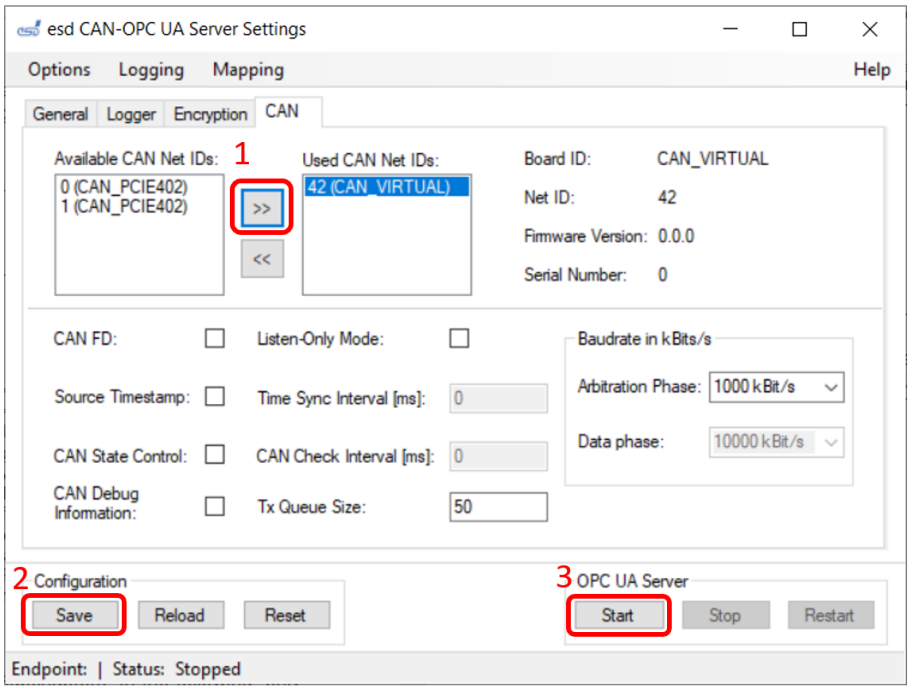


Figure 3: *Quick Start: Configuration of CAN Interfaces*

Add OPC UA Nodes
To add some OPC UA *Nodes* to the *Server*, it is needed to add a *Server* database. The *Server* comes with a sample configuration in form of a CSV and DB file. However, to configure it for the CAN interface that is being used, there are some changes to do. If the configured CAN interface is on net number '42', the following step can be skipped.

Open the *Mappings Menu* and press tab *Options → Update CAN net.*
Enter '42' as the old CAN net number and the net number of the newly configured CAN interface as new CAN net number.
After that, press the button *Choose DB* and select the database *sample_db.db*, which is located in the documents directory under *ESD\OPC_UA\sample\mapping*.
When the textbox in the *Mappings Menu* displays *FINISH: success*, the database is updated successfully.

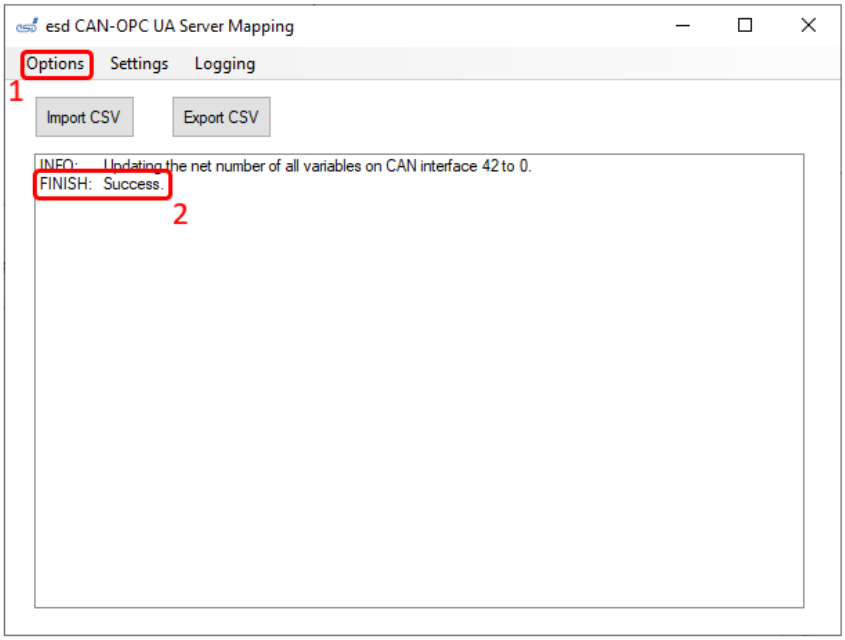


Figure 4: *Quick Start: Add OPC UA Nodes (1)*

Open the *General* tab in the *Settings Menu* and select the database *sample_db.db*, which is located in the documents directory under *ESD\OPC_UA\sample\mapping*.
Save the configuration and start the *Server*. When browsing the *Server*, there are some OPC UA *Nodes*, which can be used to discover the gateway functionality.

The sample provides a possible implementation of a control over two pumps, which are implemented as *Objects* and contain several *Variables* to inform about the actual sensor data and the pump state. The parameter of each *Node* is shown in **Table 1**. However, it is also possible to enable the option *CAN Debug Information* in the *CAN* tab of the *Settings Menu*, which adds an additional *Node* to each *Variable* with further information.

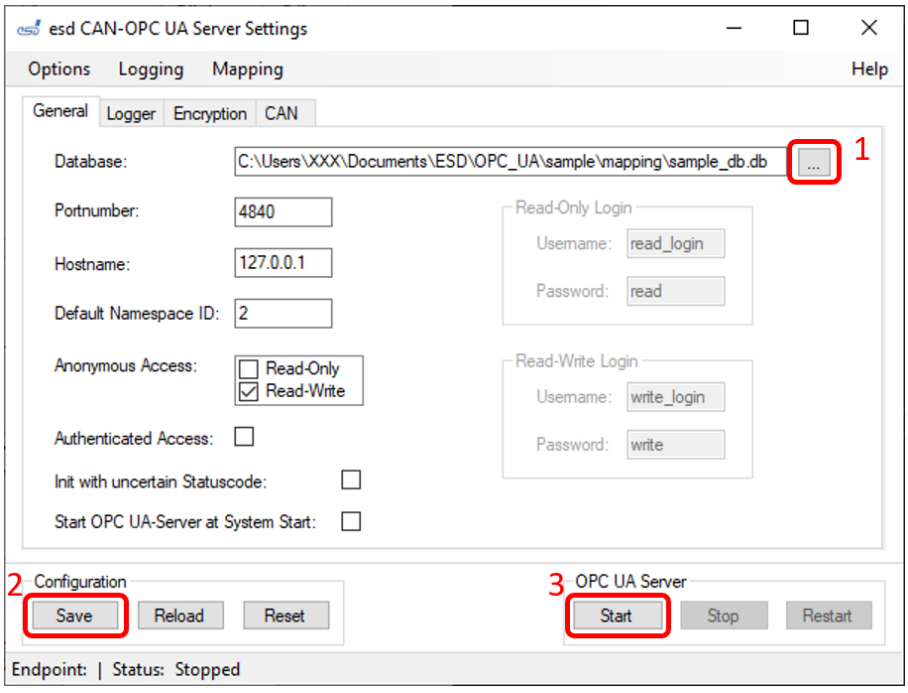


Figure 5: *Quick Start: Add OPC UA Nodes (2)*

Objects	Variables	CAN ID	Startbit	Datatype
Pump 1				
	Enable Pump	30_{dez}	0	Boolean
	Motor Speed	10_{dez}	0	UInt32
	Motor Torque	20_{dez}	0	UInt32
	Pump State	0_{dez}	0	Enumeration
	Temperature	20_{dez}	32	Float
Pump 2				
	Enable Pump	130_{dez}	0	Boolean
	Motor Speed	110_{dez}	0	UInt32
	Motor Torque	120_{dez}	0	UInt32
	Pump State	100_{dez}	0	Enumeration
	Temperature	120_{dez}	32	Float

Table 1: Sample configuration with two pumps

For an in-depth description of all adjustments and features take a closer look at the following chapter.

4 Mapping

This chapter describes the mapping between the OPC UA *Nodes* and the CAN interface.
The *Server* has several configurable *Node* classes to add. However, the most important ones, which are mapped onto the CAN frame, are the *Variables*. Each *Variable* contains multiple options, but the basic schema is shown in Figure 6. The bigger cells show the bytes while the smaller cells show the individual bits.
There is a *Variable* configured, that is declared as an unsigned 16-bit integer data type. It is also configured as a Classical CAN frame with the identifier 12_{dez} and the start bit position 21. Because only full bytes can be sent, this leads to a frame with the length of five bytes. However, the variable is only linked to parts of the 3rd, 4th and 5th Byte. The other parts of the Frame are free to use and do not affect this *Variable.*
Whenever the frame is sent, all *Variables*, which are configured on the frame, are transmitted. The length of the frame is always as short as possible, which is defined by the linked *Variable* with the highest startbit. It is also possible to define the bit length of a *Variable* independently of the data type. This can be useful if the *Variable*'s value is limited to a smaller range of numbers than the data type. The *Server* is able to process *Variables* bit exact regardless of the byte boundaries. However, to increase performance, it is recommended to choose numbers which start and end on the byte boundaries.

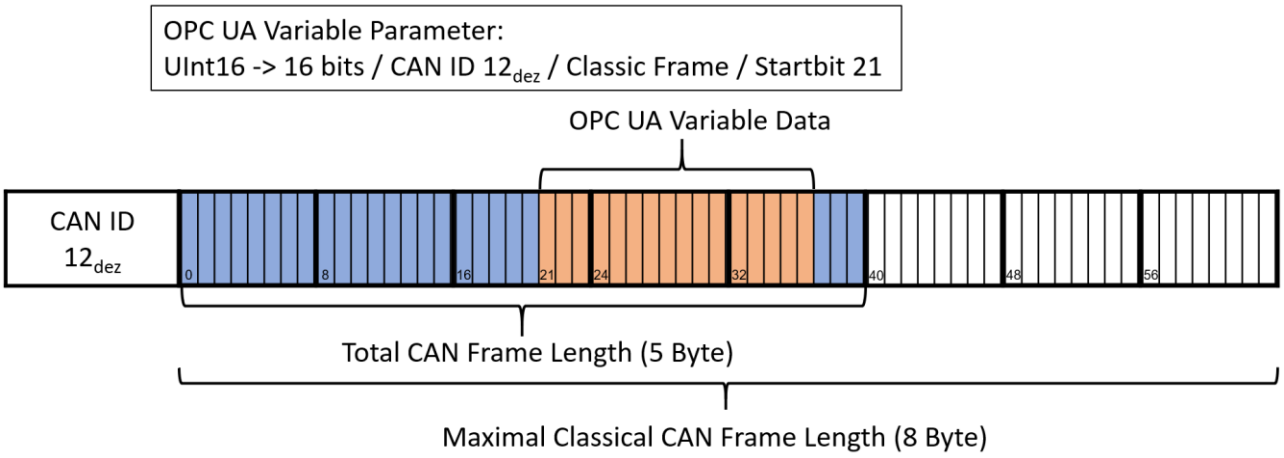


Figure 6: *Mapping schema*

As mentioned in the previous chapters, the mapping of the *Server* is done via CSV files. The *Objects* within a line must be separated by either a comma or a semicolon. This also means, that it is not possible to add any of these characters in one of the parameters. There is also a template XLSX file to create such files with a spreadsheet application, which is recommended to use due to the count of optional parameters and dependencies between them. After finishing the mapping, the CSV file needs to be parsed to a DB file, which consists of an *SQLite* database. To get a closer look how to import or export a CSV file, see subchapter 5.5.

INFORMATION
Changing the bit length on the CAN frame different from the data type bit length can lead to inconsistencies between the sent CAN frame and the OPC UA *Variable* value.
Use this only if the number range is well-known!

4.1 Mapping Classes

The *Server* consists of *Nodes* which are similar to objects in OOP. Each *Node* is defined by a class and can clearly be identified by a unique so-called *Node* ID. Depending on the type, each *Node* ID consists of a valid *Namespace* and a unique identifier as string or 32-bit unsigned integer.
The following objects can be configured during configuration:

Namespaces	*Namespaces* define a room in which *Nodes* are stored. By default, the *Server* contains two *Namespaces*. Namespace '0' is blocked for the necessary OPC items and *Namespace* '1' is blocked for automatically assigned *Nodes* of the *Server*. Therefore, the index of user *Namespaces* starts at '2'. *Namespaces* itself are not defined as *Nodes* and therefore are not defined by a *Node* ID.
Standard Units	The *Server* comes with a big variety of *Standard Units*, which can be assigned to *Variables*. These are automatically be added to the *Server* database. To get an overview, it is also possible to browse through the *Standard Unit* list in the XLSX template or the CSV file s*tandard_units.csv* in the config directory.
Units	The *Server* supports additional user assigned *Units* if the required *Unit* is not part of the *Standard Units*.
Enumeration	The *Server* can handle enumerations, which can be assigned to *Variables*. Each *Enumeration* is defined as a 32-bit integer value.
Enumeration Value	This class is used, to add value to an *Enumeration*.
Objects	To keep a proper overview, the *Server* allows the use of *Objects* for structuring purpose, which can hold an indefinite number of *Variables*.
Variables	*Variables*, including *Properties*, are the process items, which are connected to a CAN interface. The *Server* includes several adjustments to define the behavior of the *Variable*.

The following subchapters describe the individual parameters for each item type. Parameters which are not presented in bold are optional.

4.1.1 Namespaces

A *Namespace* consists of two parameters:

Namespace ID	Unique ascending ID of the *Namespace*. The first two *Namespaces* being blocked, the user-assigned *Namespaces* start at ID '2'. Because it is not possible to skip a number, there it is no way to assign a specific ID to a *Namespace*.
Namespace URI	Unique identifier for the *Namespace*.

4.1.2 Standard Units / Units

Units consists of up to four parameters:

Unit ID / Extended Unit ID	Unique identifier for every *Unit*. *Units* and *Standard Units* are handled separately and therefore can share the same ID.
Browse- / Displayname	Name of the *Unit*. To prevent duplicates, every name needs to be unique within its class.
Namespace URI	Assigned *Namespace* URI of the *Unit*.
Description	Additional description.

4.1.3 Enumeration

Enumeration consists of up to eight parameters:

Enumeration ID	Unique identifier for every *Enumeration*.
Browse- / Displayname	Name of the *Enumeration*. To prevent duplicates, every name needs to be unique within its class.
Node ID Namespace / Type / Identifier	Unique *Node* identifier within the *Namespace*.
Value Node ID Namespace / Type / Identifier	Unique *Node* identifier of the assigned *Enumeration Values* within the *Namespace*. Because all assigned *Enumeration Values* are saved as one array, there is only a single *Node* ID needed for all *Enumeration* values.

4.1.4 Enumeration Value

Every *Enumeration* value represents one value and is assigned to an *Enumeration*. This offers an easy way to manage the *Enumeration* value by adding and deleting values without recreating the whole *Enumeration*. The maximal number of values per *Enumeration* is limited to 100 values.
One value consists of up to four parameters:

Enumeration ID	ID of the assigned *Enumeration*.
Browse- / Displayname	Name of the value. *Enumeration Values* can share the same name.
Value	Defines the 32-bit integer value which should be added to the *Enumeration*. It is recommended to set each value unique within its *Enumeration* to prevent inconsistencies.
Description	Additional description.

4.1.5 Object

An *Object* consists of up to six parameters:

Object ID	Unique identifier for every *Object*.
Browse- / Displayname	Name of an *Object*. It is possible that two *Objects* share the same name. However, this is not recommended.
Node ID Namespace / Type / Identifier	Unique *Node* identifier within the *Namespace*.
Description	Additional description.

4.1.6 Variable

The *Variable* class is the most important class and is the only one, that interacts with the CAN interface. It contains by far the most parameter. However, most of the parameters are optional:

Variable ID	Unique identifier for every *Variable*.
Object ID	Object ID of the assigned *Object*. If none or an invalid ID is selected, the *Variable* is added to the base *Object* folder.
Browse- / Displayname	Name of the *Variable*. *Variables* can share the same name.
Node Namespace / Type / Identifier	Unique *Node* identifier within the *Namespace*.
Data Type	OPC UA datatype of the *Variable*. For an overview of all supported datatypes, see table 7.2.
Enumeration ID	Enumeration ID of the assigned *Variable*.
CAN Net	*Net ID* of the connected CAN interface.
CAN ID Type	Defines whether the assigned CAN ID is an 11- or 29-bit identifier. By default, it is an 11-bit identifier.
CAN ID	Assigned CAN ID as decimal number.
CAN Frame Type	Defines whether the assigned CAN frame is a classic CAN or CAN FD frame. All *Variables* on the same frame need to be consistent about the frame type. Moreover, CAN FD must be enabled on the assigned CAN interface. By default, it is a Classical CAN frame.
Position	Bit exact position on the CAN frame. Its recommended to use a position that is divisible to eight to increase performance. By default, the *Variable* starts at bit position ‘0’.
Length	Bit exact length on the CAN frame. Its recommended to use a length that is divisible to eight to increase performance. By default, the length is defined by the chosen datatype.

Endianness	Defines wheater the *Variable* should be handled as *Little Endian* or *Big Endian. Big Endian* is only available for *Variables* that start and end on the byte boundaries. By default, the endianness is *Little Endian.*
Send Behavior	There are two options when it comes to the send behavior of a *Variable*. The assigned CAN frame can be send immediately after a value change or when the flag on the CAN interface *Node* is triggered. For more information see chapter 6. By default, it is sent immediately after a value change.
Description	Additional description.
Accessibility	Defines whether the *Variable* is read-only or read-write. By default, it is read-only.
SendRTR	Option to send an RTR frame of the assigned CAN ID. This option is only available for read-only *Variables* and Classical CAN frames.
Value	Initial value before the value of the variable is changed or an assigned CAN frame is received. To clarify that a value is initial, every *Variable* gets the status code *Uncertain_InitialValue*. By default, the value is '0'.
Variable type	Defines whether the added class should be created as *Variable* or as *Property*. Usually, *Variables* are being used for process data and can hold additional information like *Units*, while *Properties* are used for flags and status information. If the *Variable* is defined as *Property,* it is not possible to add other *Properties* due to restrictions of OPC UA. By default, it is set to *Variable*.
Minimal Sampling Time	Interval in which a *Client* can read the *Variable*.
Unit ID	ID of the assigned *Standard Unit*. Set to '-1' to use a user specific *Unit*.
Extended Unit ID	ID of the assigned user specific *Unit*.
EURange_Low/_High	Maximal numerical range of the *Variable*. Defined by two double parameters.
InstrumentRange _Low/_High	Nominal numerical range of the *Variable*. Defined by two double parameters.
Value Precision	Value precision of the *Variable*'s value.
Gradient	Gradient of a possible pre-processing of the *Variable*'s value. For additional information see subchapter 4.2.
Offset	Offset of a possible pre-processing of the *Variable*'s value. For additional information see subchapter 4.2.
Definition	Additional definition of a *Variable*.
Additional Node Namespace / Type / Identifier for every *Property*:	Unique *Node* identifier of every possible *Property* within the *Namespace*.

INFORMATION
The *Server* can manage *Node* IDs on its own. Whenever no *Node* ID is set, the *Server* automatically assigns a unique numeric identifier to the *Variable*. However, to ensure that the *Node* IDs are only assigned once per configuration, the new assigned *Node* IDs are saved in the *Server*'s database. When a new configuration is imported, all unassigned *Node* IDs are reallocated. To back up your data, it is possible to export the current configuration as a CSV file.

Every text parameter is limited to a maximum of 254 characters.

4.2 Additional Properties

As mentioned in the previous subchapters, it is possible to attach further information to *Variables* by adding *Properties* to them. Whenever at least one parameter of the *Property* is set, the related *Property* is added. At the current version, all *Properties* except for *SendRTR* are read-only and have no effect on the *Variable*'s value or status code.

The following *Properties* can be assigned:

Engineering Unit	*Property* of type *EUInformation*. Displays the selected *Standard* or user specific *Unit*.
EURange	*Property* of type *Range*. Contains the minimal and maximal value range.
InstrumentRange	*Property* of type *Range*. Contains the minimal and maximal instrument value range.
Value Precision	*Property* of type *Double*. Contains the value precision.
Scaled Value	*Property* of type *Double*. Displays the scaled value of the *Variable* with the help of the parameters *Gradient* and *Offset*. This can be used, for example, to scale analog values into their physically correct values assuming there is a linear relation.
Definition	*Property* of type *String*. Offers the possibility to add an additional definition to the *Variable*.
SendRTR	*Property* of type *Boolean*. *Property* can be used to send an RTR frame on the configured CAN frame. This *Property* is only available for Classical CAN frames and read-only *Variables*.

INFORMATION
Due to the restrictions of the OPC UA Informationmodel it is not possible to add a *Property* to another *Property*. Therefore, if a *Variable* is configured as *Property*, it is not possible to add any additional *Properties*.

4.3 Spreadsheet Template

As mentioned in the previous chapters, there is the possibility to build the configuration in a spreadsheet editor by using the provided XLSX file which is located at the document's directory under *ESD\OPC_UA*. It is easy to use and offers a lot of assistance. It consists of a table for every addable class. It is recommended to process the individual tables from left to right, since they build up on each other. A change to, for example, the *Namespace*, can invalidate all *Objects* building up on them. Some parameters might differ slightly from the parameter description in subchapter 4.1 to improve the user experience. For example, the *Enumeration Values* are assigned to the *Enumeration* name instead of its ID. In the CSV export, the values are exchanged. After adding all *Node*s, simply go to the table *CSV* and save it as CSV UTF-8 file.

INFORMATION
Use '.' as the decimal separator.

4.4 Structure of a Text Line

Format:

<ElementType>, <Variable ID>, <Object ID>, <Enumeration ID>, <Namespace ID>, < Node ID Type>, <Node ID Identifier>, <Browse- / Displayname>, <Description>, <Namespace URI>, <Value Namespace ID>, <Value Node ID Type>, <Value Node ID Identifier>, <Data Type>, <CAN Net>, <CAN ID Type>, <CAN ID>, <CAN Frame Type>, <Position(Bit)>, <Length(Bit)>, <Send Behavior>, <Accessibility>, <SendRTR>, <Value>, <Variable Type>, <Minimal Sampling Time>, <Unit ID>, <Extended Unit ID>, <EURange_Low>, <EURange_High>, <InstrumentRange_Low>, <InstrumentRange_High>, <Value Precision>, <Gradient>, <Offset>, <Definition>, <Unit Namespace ID>, <Unit Node ID Type>, <Unit Node ID Identifier>, <EURange Namespace ID>, <EURange Node ID Type>, <EURange Node ID Identifier>, <InstrumentRange Namespace ID> , <InstrumentRange Node ID Type>, <InstrumentRange Node ID Identifier>, <ValuePrecision Namespace ID>, <ValuePrecision Node ID Type>, <ValuePrecision Node ID Identifier>, <Definition Namespace ID>, <Definition Node ID Type>, <Definition Node ID Identifier>, <SendRTR Namespace ID>, < SendRTR Node ID Type>, < SendRTR Node ID Identifier>,<Endianness>

For a detailed description of each parameter, see the previous subchapters. In the following, only the parameters that have fixed selection options are defined.

ElementType	VAR	Variable
	OBJECT	Object
	NS	Namespace
	UNIT	Unit
	ENUM	Enumeration
	ENUMVAL	Enumeration Value

… Node ID Type:	0	Numeric
	3	String

Datatype	-1	ENUM	0	BOOLEAN
	1	SBYTE	2	BYTE
	3	INT16	4	UINT16
	5	INT32	6	UINT32
	7	INT64	8	UINT64
	9	FLOAT	10	DOUBLE

CAN ID Type	0	11-bit identifier

	1	29-bit identifier
CAN Frame Type	0	Classical CAN Frame
	1	CAN FD Frame
Send Behavior	0	Send after change
	1	Send on demand
Accessibility	0	Read-Only
	1	Read-Write
Send RTR	0	No
	1	Yes
Variable Type	0	Variable
	1	Property
Endianness	0	Little Endian
	1	Big Endian

INFORMATION
When no value is provided, it is treated as a ‘0’. Therefore, it is not possible to use only a ‘0’ for parameters defined by a *string*.

5 Server Configuration

This chapter deals with the settings and describes in detail, how to setup and run the *Server*.
The *Server* itself runs as *Windows Service* in the background. To control the *Service*, it comes with a control tool. By default, this tool is launched minimized at system start as a tray icon.
If auto start is disabled, it can be started via the Start menu (default: Start → Programs → esd → CAN-OPC UA Server Control).
By hovering over the tray icon, the actual status of the *Service* is displayed. To get additional information, you can either right-click on the icon to get a small toolbar which can be used to control the *Service*, open *Settings*, *Logger*, or *Mapping Menus* or close the control tool.
On double-left-click the *Settings Menu* appears immediately. To prevent the tray icon from start automatically, deactivate the event in the *Windows Task Schedular*.

INFORMATION
To start / stop *Windows Services* it is required to have administrator rights. Therefore, the control tool needs administrator rights.

5.1 Settings Menu

The *Settings Menu* provides all available options to configure the *Server*. On top there is a small toolbar which contains the tabs *Options, Logging, Mapping, Help*. *Options* provide the possibility to import or export the settings.
Logging and *Mapping* open other menus and *Help* provides additional information and assistance for the software.
Under the toolbar there are the *Server*'s settings separated into four tabs, which are described in detail within this chapter. Beneath the settings there are six buttons in total, which can be used to save, reload, or reset the configuration and to start, restart and stop the *Service*.
Most recently, located at the bottom, there is a small status bar, which provides connection information and current *Server* state. On the following pages, all settings tabs are explained in detail.

5.1.1 General Settings

The *General* tab contains some universal settings for the *Server*:

Database	Path to the *Server*'s database.
Portnumber	Port under which the *Server* is launched.
Hostname	Hostname / IP Address to connect to the *Server*.
Default Namespace ID	Namespace ID for all Node IDs, which were not defined by the user specifically.
Anonymous Access	Option to enable and set permissions for anonymous access.
Authenticated Access	Option to enable authenticated access. To get more information about the access control, see subchapter 5.4.
Read-Only / Read-Write Login	Username and password for authenticated *Server* access.
Start OPC UA Server at System Start	Controls whether the *Server* starts with the system start or need to be launched manually.
Init with uncertain Status Code	Option to initialize all variables with the status code *Uncertain_InitialValue* until a CAN Frame regarding the variable was either read or send.

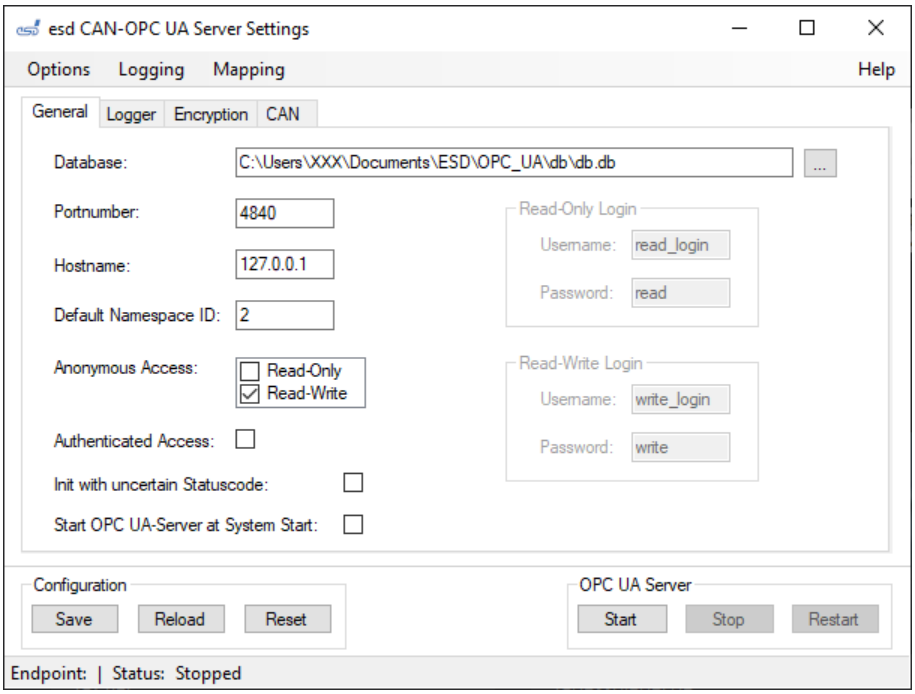


Figure 7: *Settings Menu General* Tab

5.1.2 Logger Settings

The *Logger* tab provides control over the logging options of the *Server*:

Logger Destination	Defines where the log entries are displayed / saved.
Logger Origin	Defines which log origins should be logged.
Logger Level	Defines which log level should be logged.
Log File	Path to a txt file, which is used if logger destination *Text File* is enabled.

A more detailed description of the logger is given in the subchapter 0.

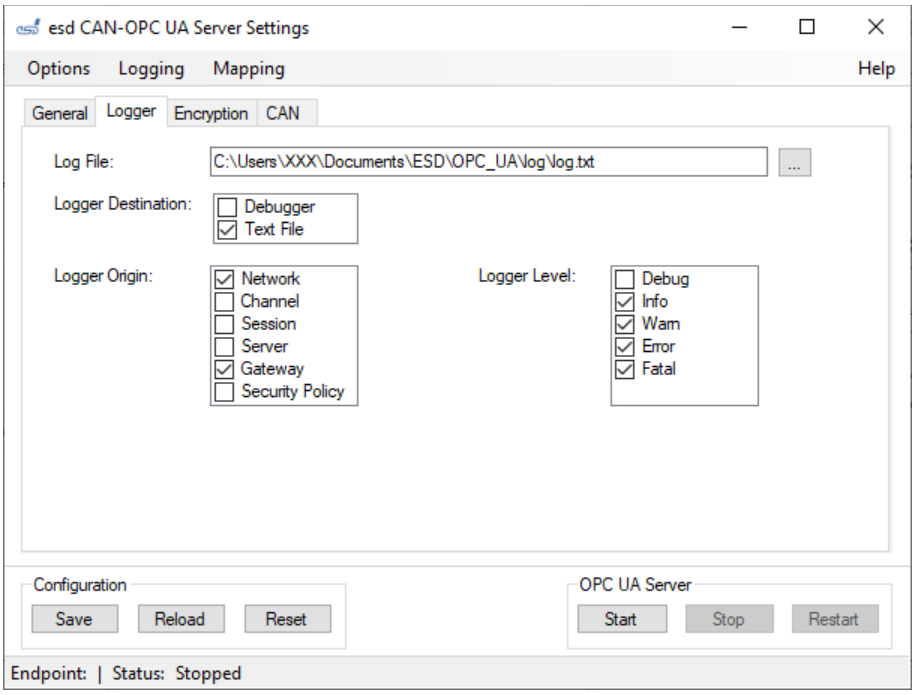


Figure 8: *Settings Menu Logger* Tab

5.1.3 Encryption Settings

The *Encryption* tab provides control over the encryption feature of the *Server*:

Encryption Mode	Defines which encryption algorithms are enabled.
Certificate	Path to public *Server* certificate.
Private Key	Path to private *Server* key.
Trusted List / Issuers List / Revocation List	List of paths that define, which *Client* is able to build an encrypted connection to the *Server*.
Generate Certificate / Key	Opens the certificate / key generator.

For a more detailed description of encryption see subchapter 5.3.

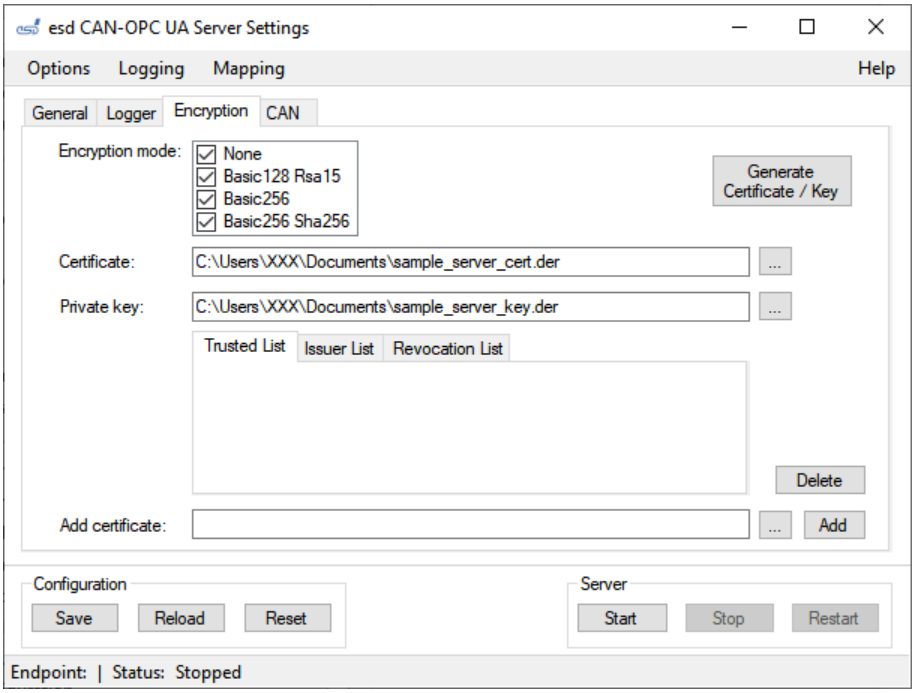


Figure 9: *Settings Menu Encryption* Tab

5.1.4 CAN Settings

The *CAN* tab provides control over the CAN configuration:

Available CAN Net IDs	List of all available unused CAN interfaces.
Used CAN Net IDs	List of all used CAN interfaces.
Additional CAN Information	Hard- and Software information of the selected CAN interface (*Board ID, Net ID, Firmware Version, Serial Number*).
CAN FD	Option to enable CAN FD.
Listen-Only Mode	Option to enable Listen-Only mode.
Source Timestamp	Option to enable source timestamps, which uses CAN timestamps to calculate the time, when the CAN frame is received.
Time Sync Interval	Interval in which the *Server* and CAN timestamps are synced.
CAN State Control	Option to enable bus state control.
CAN Check Interval	Interval in which the CAN state is checked.
CAN Debug Information	Option to enable additional debug information to every *Variable*.
Tx Queue Size	Buffer size for outgoing CAN frames.
Baudrate	Baudrate during arbitration and data phase. The data phase baudrate is only used for CAN FD. For Classical CAN only the arbitration phase baudrate is being used.

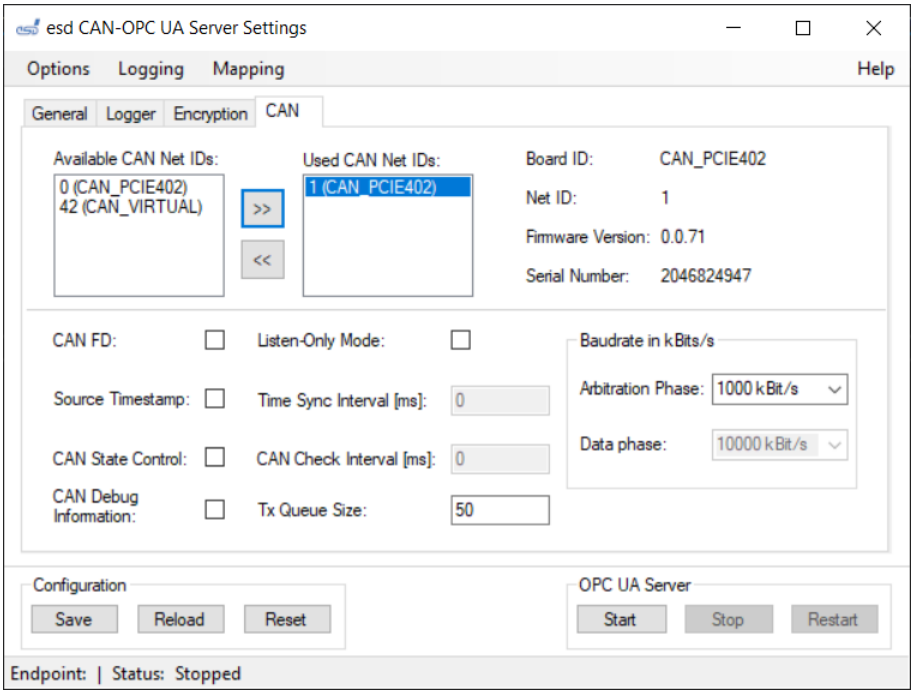


Figure 10: *Settings Menu CAN* Tab

For a more detailed description of the CAN parameter, take a closer look at the NTCAN-API Manual.

5.2 Logger

The *Server* has extensive logging possibilities such as multiple log levels and origins, which can be configured freely. It is highly recommended to enable at least the levels *Warning*, *Error* and *Fatal* during commissioning. *Debug* and *Info* provide additional information of the *Server*, which can be useful in an error case.
At the origins, it is recommended to use at least *Gateway* and *Network* because they log every information about *Variable* processing and established connections. The logging entries can either be displayed in a *Debugger* or saved in a TXT file. If the latter is enabled, the menu option *Logging* can be used to keep track of the log entries without reloading the text file.

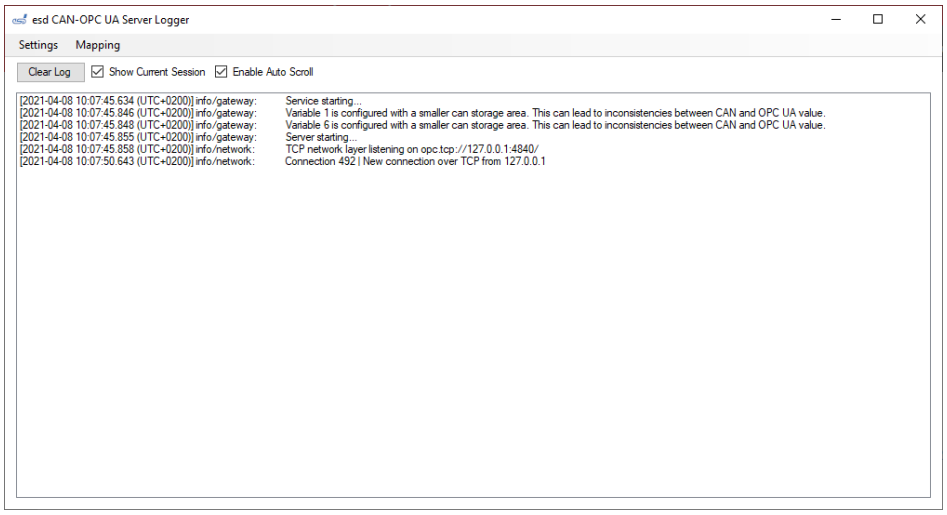


Figure 11: *Logging Menu*

The *Logging Menu* provides a window in which the last up to 200 log entries are displayed.
With the option *Show current session* only the log entries since the last *Server* start are displayed. This function looks for a specific log entry and does not work when the log level *Info* und the log origin *gateway* are disabled.
The option *Enable Auto Scroll* automatically jumps to most recent log entry.
The button *Clear Log* deletes all log entries within the logging file.

INFORMATION
Logging, especially of the log levels *Debug and Info* can decrease performance. It is recommended to stop logging these levels in normal operation.

5.3 Encryption

Before explicitly mentioning the possibilities of the *Server*, this chapter gives a short introduction on encryption in OPC UA. OPC UA uses so-called X509 certificates for message signing and encryption. Signing means that the *Server* signs all messages to assure recipients that the *Server* is authentic and not an imposer. Encryption, on the other hand, provides confidentiality by guaranteeing that only the receiver is able to read the message and prevents attackers from reading plain text messages from the *Server*.

Basically, it works by assigning a private key and a certificate, which consists of identity information and a public key, to every participant. Asymmetric encryption methods are being used to connect *Server* and *Client*. First, *Client* and *Server* both exchange their public keys. To connect successfully, both participants need to trust each other's keys. In case of this *Server*, add the *Client*'s certificate to the trust list in the *Encryption* Tab in the *Settings Menu*. On the *Client*'s side that might differ depended on the vendor of the OPC UA *Client*. Once the trusted connection is established, the *Client* and *Server* are connected. After that, a secure channel is created, to encrypt and sign messages. For subsequent messages, a common secret is used for symmetric encryption which has the advantage of being much faster.

In practice, the *Server* supports the following security policies: *None*, *Basic128Rsa15*, *Basic256*, *Basic256Sha256*.
None, as the name suggests, is an unsecure unencrypted connection, which is only recommended for use in closed local networks. If the *Server* is configured without it, an unencrypted endpoint is provided, which is used for discovery purposes only. The other three algorithms are able to establish an encrypted connection, but it is recommended to use *Basic256Sha256* because the other two algorithms are deprecated.
When at least one encryption method is enabled, the *Server* creates an endpoint for *Sign* and *SignAndEncrypt*, which can be used to connect. They either just sign the messages or sign and encrypt the messages. The *Server* only accepts certificates and keys in form of DER (Distinguished Encoding Rules) encoded binary data files. The config directory provides some sample certificates and keys for test purposes. Another widely used format for the private keys is the PEM format. The control tool offers a generator, which is able to generate unsigned certificate / key pairs in both DER and PEM file format. To use it, *OpenSSL* needs to be installed and added to the system path. To open it, go to the *Encryption* tab in the *Settings Menu* and press the button *Generate Certificate/Key*.

INFORMATION
The *Server* checks if the provided application uri matches the one in the certificate. The *Server* has the application uri *urn:esd.server.application*. The included demo *Client* uses the application uri *urn:esd.client.application*.
When no trusted certificate is given, any certificate is accepted.

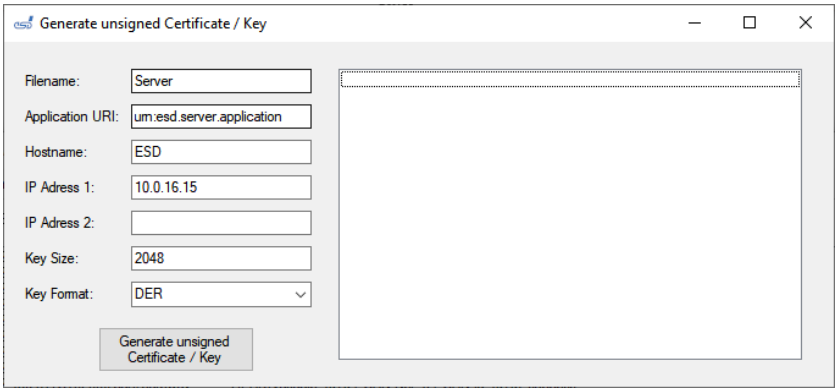


Figure 12: *Certificate / Key Generator*

5.4 Access Control

The *Server* offers two options to control the access by either the user or the attribute permission.

The user permission describes which *Client* should be able to establish a connection to the *Server* and what write permission the *Client* gets. It is possible to choose between anonymous and authenticated access, or a combination of both of them. To activate anonymous access, simply enable the option in the *General* tab in the *Settings Menu* with the desired permission. To activate authenticated access, enable the option and choose usernames and passwords in the *General* tab in the *Settings Menu*. The *Server* adds two logins, which either have read-only or read-write permission.

The attribute permission restricts the write permission of each *Variable*. This can be done by configuring a *Variable* as read-only. It does not allow any *Client* to write the *Variables* value, even though the *Client*'s user permission is set to read-write.

INFORMATION
At the current version it is not possible to write anything else than the *Variables* value. All other parameters are read-only and cannot be changed while running the *Server*.

5.5 Import and Export Configuration

To import or export a configuration, open the *Mappings Menu.* It is able to parse between a CSV and DB file in both directions. The output of the parser is displayed in the window below. When the parser finishes with *Finish: Success*, the parsing operation was successful. Otherwise the parser provides additional information about the failure. There is also a possibility to update the net number of the CAN interface without editing the CSV file. Just press the button *Update CAN net* under the tab *Options.* Set the net number, which should be replaced as *Old net number* and the new one as *New net number.* If all *Variables* in the configuration should change to the requested net number, set '-1' as old net number.

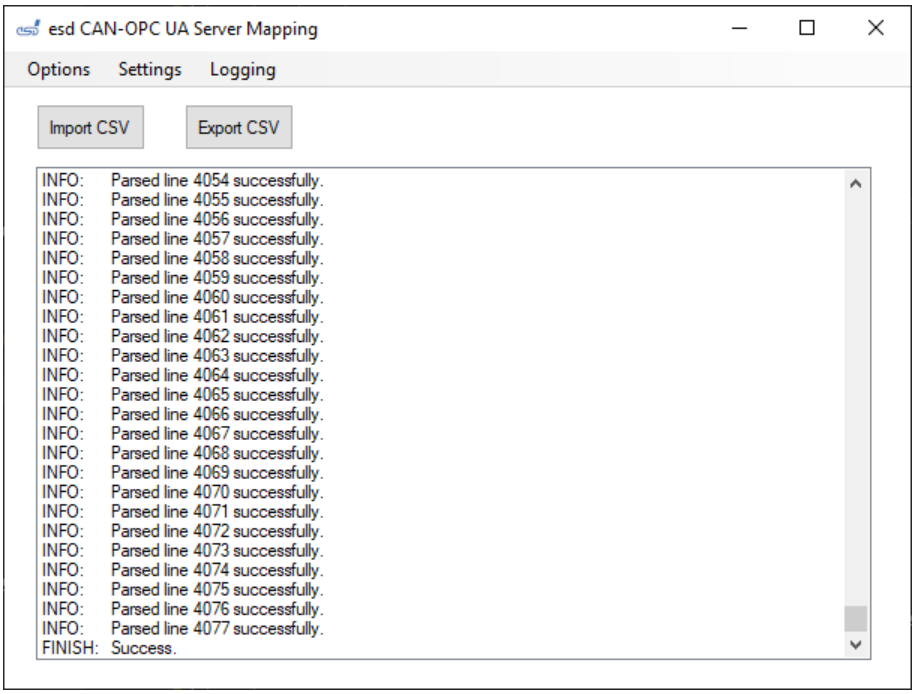


Figure 13: *Mapping Menu*

INFORMATION
The *Server* builds a new database if it is not available. If the database already exists, the *Server* reuses all static tables to enhance performance. If that is not wanted, delete the old database manually.

5.6 CAN Information

To get additional information about the connected CAN interfaces, there are automatically created *Objects*, which can be read by browsing the *Server*. For every CAN interface, an *Object* is created, which is named after the net number of the CAN interface. It contains several *Variables* including information about statistic, current bus state or the baud rate. It also contains a flag, which can be used to trigger *Variables* which are configured as *SendOnDemand*. Moreover, by enabling the feature *CAN Debug Information*, an *Object* is attached to each *Variable*, which contains the linked CAN information like CAN ID and startbit. However, due to the extra number of *Nodes* needed, it is recommended to disable this feature during common operation.

5.7 CAN State Control

The *Server* can be used to check the state of the bus. To activate this feature, enable the option in the *CAN* tab in the *Settings Menu*. It opens a second CAN handle, which checks the bus state in the time interval configured by the parameter *CAN Check Interval*. It uses *NTCAN Events* to detect the bus state. Therefore, it is able to detect an error even though the error disappears within a time interval. When an error appears, which means that the bus reaches the state *OFF*, the status code of all effected *Variables* is changed. However, because the *Server* does not wait for every frame to be sent, it is not able to detect if a frame was sent successfully. The status code of the *Variables* is reset, when either a CAN frame is successfully received or sent. The CAN state control can be configured for each CAN interface separately.

5.8 Timestamps

Each OPC UA *Variable* has a server and a source timestamp.

The server timestamp is updated on each successful value change or when the *Server* knows that the value is accurate. Depended on the type, this has a different meaning for read-write and read-only *Variables*. For read-write *Variables* it reflects the time when the *Variable* value was changed the last time. For read-only *Variables* it represents the time when the *Server* lastly checked for receiving CAN frames. When the status code of a *Variable* changes to *Bad* or *Uncertain*, the server timestamp reflects the time that the *Server* last tried to recover from the status.

The source timestamp is used to reflect the timestamp that is applied to a *Variable* value by the data source. It also differentiates between read-only and read-write *Variables*.
When the *Server* sends a CAN frame, it is the data source himself. Therefore, for read-write *Variables*, it reflects the time when the last CAN frame of the connected *Variable* was sent. For *Variables*, which send the CAN frame directly after the value has changed, it is equal to the server timestamp. For *Variables*, which send the CAN frame on demand, it is updated when the frame with the new assigned value is send.

For read-only *Variables*, there are two configuration options. When the option *Source Timestamp* is disabled, the source timestamp represents the time, when the Server read the CAN frame. Because the *Server* checks the value only when it is requested, this is not the time, when the frame is received. However, by enabling the option in the *Settings Menu* it uses the total timestamp of the OPC UA *Server* and the related timestamp of the CAN interface to calculate the timestamp of the last received CAN frame. Both times are synced at start up. To enhance the accuracy, it is possible to sync the times in an interval which can be configured by the parameter *Time Sync Interval [ms]*. If *Time Sync Interval [ms]* is ‘0’, both times are only synced at start up.

When the status code changes to *Bad* or *Uncertain*, the source timestamp reflects the time when the error is detected. Thus, every *Client* is able to detect when the error happened.

At start up, both timestamps will represent the build time of the server.

5.9 Variable Status Codes

Every *Variable* contains a status code, which provides information about the current state of the *Variable*. Whenever an operation in conjunction with the *Variable* is not executed as usual, the status code is adjusted to the corresponding state.

In the following all used status codes are described:

Good	Operational.
Uncertain_InitialValue	On start up every *Variable* has this status code if the option *Init with uncertain Status Code* is enabled. It means, that the value of the *Variable* is set as initial value and is not the result of a CAN interaction. After receiving or sending a connected CAN frame, the status code is changed.
Uncertain_LastUsableValue	This status is set when a read CAN operation in conjunction with a *Variable* fails, for example, when the bus state changes to *OFF*. When a CAN frame is received successfully, the status code is set back to *Good*.
Bad_NotConnected	This status is set when a write CAN operation in conjunction with the *Variable* fails. When a frame is sent without an error and the bus state is ok, the status code is adjusted to *Good.*
Bad_ConnectionClosed	If an error occurs at the configuration of the CAN interface or a CAN interface is missing, every *Variable* connected to the interface gets this status code. Usually, it is the result of a configuration error. To resolve this status code, a restart is required.
Bad_DataUnavailable	When no value is written to a *Variable*, this status code is set, and no CAN frame is sent.
Bad_InternalError	This status is set if an internal operation of the OPC UA *Server* fails. This should not happen during normal operation.

Information
To get additional information, why a status code is set, see the log entries. They provide additional information about the failure.

6 Running the Server

As mentioned in chapter 1, the *Server* runs as a *Windows Service*, and as such, uses the states *Stopped*, *Running*, *Start Pending* and *Stop Pending*. Whenever the *Service* is in state *Running*, it is possible to establish a connection to the *Server* using the endpoint displayed at the status bar on the bottom of the *Settings Menu*. After a connection is established successfully, it is possible to interact with the *Server Nodes*. This can be done by either browsing the *Server* or accessing a *Node* directly over the *Node* ID. Whenever a read or write request for a *Variable* is received, a callback is triggered.

When the value of a *Variable* is requested, the *Server* calls either the callback for read-only *Variables* or read-write *Variables*. If the requested OPC UA *Variable* is read-only, the *Server* checks if a new relevant CAN frame is received. That is done via a CAN handle which is set in the so-called *Object Mode.* In this mode, only the latest received CAN frame is saved. Frames which were received between the newest and the last read frame is ignored. This increases *Server* performance and improves scalability. After a new frame is received, the value of the *Variable* is set based on the configured datatype and if needed, a new status code is set to the *Variable*. If the requested *Variable* is a read-write *Variable*, the *Server* just updates the timestamps and the status code of the *Variable* if needed.

On the other hand, when the *Variable* value is changed by a *Client*, the *Server* changes the linked CAN frame depended on its new value. The parts on the CAN frame, which are not used by the *Variable*, are not changed. Depending on whether the *Variable* is configured as *AfterValueChange* or *SendOnDemand*, the frame is sent directly after the value is changed or when a flag is set. The specific flag is located in the automatically created CAN *Object*, see subchapter 5.6. When it is set to *true*, every CAN frame on the interface, for which *SendOnDemand* flag is true, is sent. If an error occurs during any of the send operations, the *Server* sets the status code of the *Variable* to a non-good status code. When the configured TX buffer is not large enough, the frames are sent in multiple chunks.

INFORMATION
The reception of CAN frames is limited to the *Base Frame Format* (11-bit CAN IDs) in *Object Mode* in CAN driver versions before 3.x. To support CAN frames in *Extended Frame Format* (29-bit CAN IDs) aswell, the server uses a separate handle in FIFO-Mode, which checks periodically every 50ms if new CAN frames are received.

7 Additional Information

7.1 OPC UA Server Limits

Maximum Variables	4000
Maximum Objects	2000
Maximum Namespaces	100
Maximum Enumerations	500
Maximum Values per Enumeration	50
Maximum Variables per CAN interface	2048
Maximum Secure Channels	50
Secure Channel Timeout [min]	5
Maximum Sessions	100
Session Timeout [min]	60
Minimum Publish Interval [ms]	10
Minimum Sampling Interval [ms]	10
Maximum Subscriptions	400
Maximum Monitored Items	1000
Maximum Nodes per Read / Write / Browse	10000

7.2 Supported Datatypes

Data Type	Bit Length	Description
Boolean	8	A two-state logical value (true or false).
SByte	8	An integer value between -128 and 127.
Byte	8	An integer value between 0 and 255.
Int16	16	An integer value between -32.768 and 32 .767.
UInt16	16	An integer value between 0 and 65.535.
Int32	32	An integer value between -2.147.483.648 and 2.147.483.647.
UInt32	32	An integer value between 0 and 4.294.967.295.
Int64	64	An integer value between -9.223.372.036.854.775.808 and 9.223.372.036.854.775.807.
UInt64	64	An integer value between 0 and 18.446.744.073.709.551.615.
Float	32	An IEEE single precision (32 bit) floating point value.
Double	64	An IEEE double precision (64 bit) floating point value.
Enum	32	An Enumeration defined by a Int32 value.

Table 2: Supported Datatypes

8 Order Information

8.1 Software

Type	Properties	Order No.
CAN-OPC UA Server	CAN FD to OPC UA Server for *Windows* with flexible *Object* mapping	C.1103.31

Table 3: Order information software

8.2 Manuals

PDF Manuals
For the availability of the manuals see table below.
Please download the manuals as PDF documents from our esd website https://www.esd.eu for free.

Manuals		Order No.
CAN-OPC UA Server-ME	Software manual in English	C.1103.41
CAN-API-ME	NTCAN-API: Application Developers Manual NTCAN-API: Driver Installation Guide	C.2001.21

Table 4: Available Manuals

Printed Manuals
If you need a printout of the manual additionally, please contact our sales team (sales@esd.eu) for a quotation. Printed manuals may be ordered for a fee.