CLX000 Technical Manual (v5.5X)

CSS Electronics

(Updated 2018-04-15)

Figure 1: CL1000, CL2000 & CL3000

Updated: 2018-04-15

Contents

1 About This Document 1

2 Introduction 2

3 Technical Specification 2

4 Modes of Operation 5

5 CAN-bus Connector 8

6 Memory Card Installation 9

7 Device Firmware Upgrade 10

8 Serial Interface 11

9 Wireless Connectivity 14

10 Log File Format 19

11 Device Configurations 22

12 Troubleshooting 39

1 About This Document

This manual describes the functionality of the `CLX000` series of CAN-bus data loggers with firmware `v5.5X`.

1.1 Typical application

The `CLX000` series finds its use in all applications with a need to store CAN-bus communication data. Examples of typical applications are:

- Automotive (e.g. J1939 or OBD)
- Industrial application monitoring
- Development and testing
- Monitoring and fault identification
- Monitoring of remote and mobile systems

1.2 References

This manual, firmware updates, drivers and additional software tools can be found on the product web page: `http://csselectronics.com`

2 Introduction

The CLX000 offers simple and user-friendly logging of data from a CAN-bus without the use of a computer. Data are stored on a standard SD-card in a simple format such that it can be directly loaded into an analysis tool for further post processing. The logged data can be transferred to a PC using a standard mini-USB cable and requires no special software. For advanced users, the device offers a wide range of configuration possibilities, such as message filtering, downsampling, automatic bit-rate detection, cyclic-logging, heartbeat signal, transmit messages and run-time logging control. The ability to enable message filtering and configure which data fields to log combined with the huge available storage capacity makes the device suitable for long-term data logging. The device configuration can be done on a PC and requires no special software. Also, the device can act as an USB interface to allow monitoring of real-time CAN-bus traffic from a PC.

The CL3000 supports WiFi such that log files can be extracted wirelessly.

! The default device configuration enables the logger to detect the CAN-bus bit-rate and immediately start logging data automatically. Simply connect the device to the CAN-bus using the connector pinout listed in Table 3 on page 8.

3 Technical Specification

3.1 CL1000 specification

The specification of CL1000 is given below:

General:

- Quick and easy plug-and-play installation
- Logger status is indicated using three externally visible LEDs
- Supports firmware updates such that future features can be added
- Mini-USB port to access storage and to monitor real-time CAN-bus traffic

CAN-bus:

- One physical port compliant with physical layer standard ISO 11898-2
- Compliant with CAN specifications 2.0A (11-Bit ID) and 2.0B (29-Bit ID)
- Bit rates up to 1 Mbps (manually configured or automatically detected)
- Bit-rate auto-detection[1]
- Protocol independent
- Standard D-sub 9 (DB9) connector
- Advanced message filtering on four configurable virtual channels
- CAN-bus silent/listen-only mode
- The device does not terminate the CAN-bus internally

[1] Auto-detectable bit-rates: 5K,10K,20K,33.333K,47.619K,50K,83.333K,95.238K,100K,125K,250K,500K,800K,1M

Data logging:

- Supports standard high-capacity (SDHC) SD-cards[2]
- CAN messages are time-stamped with 1ms resolution
- Highly configurable timestamp format
- Configurable data fields to log
- Logging state (enabled/disabled) can be changed run-time using a CAN-bus control message
- Supports heartbeat signal to indicate logger status periodically[3]
- Standard FAT file system
- Supports cyclic logging mode (oldest log file is deleted when file system is full)
- Channel specific message down-sampling to reduce frequency of logged messages
- Up to 20 single shot or periodic transmit messages

PC Interface:

- Real-time CAN-bus traffic monitoring
- Configurable serial interface filters
- Configurable channel to interface routing

Supply:

- Power can be supplied from CAN-bus or USB[4]
- Power supplied from the CAN-bus: +7.0V to +36V DC[5]
- Reverse voltage protection on CAN-bus supply
- Transient voltage pulse protection on CAN-bus supply
- Power supplied by USB: +5.0V DC
- Consumption:
  - `CL1000` / `CL2000`: 0.5W
  - `CL3000`: 1.0W

Mechanical:

- Robust enclosure
- Dimension: 66.7 x 42.7 x 23.5 mm (L x W x H)
- Weight: 45g

Operation:

- Operating temperature[6]: −20°C to +80°C

[2]Tested with up to 32 GB
[3]Payload includes logging state, time and space left on memory card
[4]Logging only possible with supply from CAN-bus connector
[5]7.0V to +28V DC on HW rev < 4.0
[6]Without SD-card

3.2 CL2000 specification

The specification of CL2000 is identical to the specification of CL1000 given above with the following additions:

Real-time clock:

- Built-in real-time clock with calendar and battery backup
- Absolute date and time information are added to log files
- Battery backup with more than two years of battery capacity [7]

3.3 CL3000 specification

The specification of CL3000 is identical to the specification of CL2000 with the following additions:

Wireless connectivity:

- WiFi standard IEEE802.11 b/g/n
- Stand alone Access Point (AP mode) or connected as a Station (STA mode)
- Range
  - AP mode: Up to 5 meters
  - STA mode: Up to 15 meters
- Transfer speeds up to 70Mbps
- Security
  - AP mode: WPA2
  - STA mode: WEP, WPA, WPA2
- DHCP server in AP mode with support of up to 7 connected clients
- Build in HTTP server with direct log file download
- Configurable HTTP authentication security (user name and password)
- Machine friendly HTTP-based REST interface
- Automatic push of log files to remote server using FTP
- Internal antenna

[7] Backup battery is replaceable

4 Modes of Operation

The CLX000 has two basic modes of operation:

- Mass storage device mode (MSD-mode)
- Logging mode

The device decides which mode to activate based on the mini-USB connection. If the device is connected by USB to a PC during start-up, it will activate Mass storage device mode. Otherwise, it will activate Logging mode. The flow of the modes is illustrated in Figure 2.

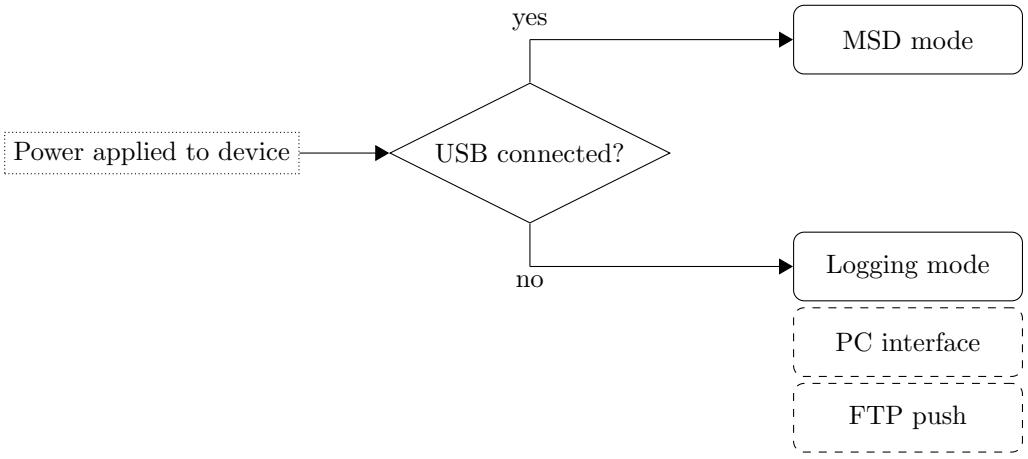


Figure 2: The modes of operation: Mass storage device mode & Logging mode

Three visible LEDs on the back of the device indicate the status of operation in the two states. The LEDs are located as illustrated in Figure 3.

Figure 3: Red, yellow and green LEDs indicate the status of the device

Below the modes and the interpretation of the LEDs are described in detail.

4.1 Mass storage device mode

If the device is powered on by the USB-cable connected to a PC, the device will activate Mass storage device mode (MSD). In this mode the internal memory will be reserved for PC access. The PC will recognise the device as a standard USB memory device such that the device storage can be easily accessed. The logged data can now be transferred to the PC without the need of special software. Additionally, the device configuration can be accessed and modified (refer to Section 11 on page 22 for details).

! To prevent corruption of the device file system, always eject the device from the PC safely. Safe ejection of the MSD on Windows is illustrated in Figure 4.

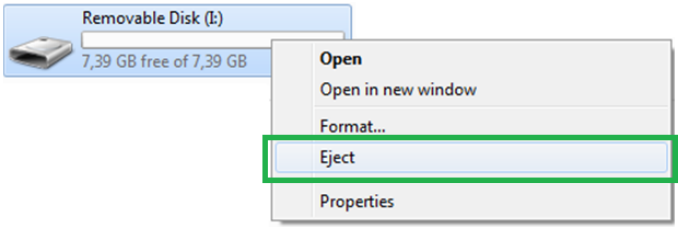


Figure 4: A mass storage device is safely ejected in Windows by right clicking the device drive in *My Computer* and selecting *Eject*

4.1.1 LED interpretation

The interpretation of the device LEDs (Figure 3) in Mass storage device mode is given in Table 1.

State	Description	Green	Yellow	Red
1	Power on	☼	○	○
2	USB connected	☼	○	☼
3	Memory error	○	☼	☼

Table 1: MSD-mode LED interpretation. LED solidly on: ☼, LED flashing: ☼, LED off: ○

Detailed LED interpretation description by state:

1. The green LED is on to indicate that the device is booting.
2. The green LED is on to indicate that the device in on. The red LED flashes as the PC accesses (reads/writes) the internal memory.
3. The yellow and red LEDs are constantly on to indicate a problem with the internal memory card. Please refer to **TC.1** on page 39.

4.2 Logging mode

If the device is powered on without the USB-cable connected to a PC (power is supplied through the CAN-bus connector), it will activate Logging mode. When the device enters Logging mode the internal memory is reserved for data-logging and cannot be accessed through the USB-port. The serial interface activates if the USB-cable is inserted after the device has entered Logging mode, refer to Section 8 on page 11 for further details on the serial interface.

As the device enters Logging mode it will attempt to read the device configuration file located on the memory-card (the device configuration is described in section 11 on page 22). If a valid configuration file is found, the configuration is loaded by the device. If no valid configuration is found, a default configuration is written to the memory card which is then loaded by the device.

The device is now ready to communicate with the CAN-bus and log messages according to the loaded device configuration file.

4.2.1 LED interpretation

The interpretation of the device LEDs (Figure 3) in Logging mode is given in table 2.

Detailed LED interpretation description by state:

1. The green LED is on to indicate that the device is booting.
2. The red LED flashes three times **if no valid configuration is found** and a valid default configuration is written to the memory.

State	Description	Green	Yellow	Red
1	Power on	☼	○	○
2	Configuration file found	☼	☼	○
3	Configuration file not found	☼	○	☼
4	Detecting bit-rate	☼	☼	○
5	Logging	☼	☼	☼
6	Memory error	○	☼	☼

Table 2: MSD-mode LED interpretation. LED solidly on: ☼, LED flashing: ☼, LED flashing three times: ☼, LED off: ○

3. The yellow LED flashes three times to indicate that a valid configuration is found and loaded.
4. The yellow LED remains solid on until a valid bit-rate is detected, **if auto-detection is configured**.
5. The yellow LED flashes to indicate CAN-bus traffic (note that the LED will be very dim if the CAN-bus load is low). The red LED flashes to indicate that data are written to the memory card.
6. The yellow and red LEDs are constantly on to indicate a problem with the internal memory card. Please refer to **TC.1** on page 39.

5 CAN-bus Connector

The CLX000 can be directly connected to a CAN-bus using a standard D-sub 9 connector. The connector supports both the CAN-bus communication signals and the supply to the CLX000. The connector is located at the end of the device, as illustrated in Figure 5.

Figure 5: View of the CLX000 CAN-bus connector.

The pinout of the D-sub 9 CAN-bus connector is illustrated in Figure 6 with the same orientation as in Figure 5. The connector pin assignment is listed in Table 3.

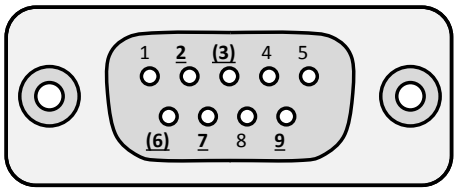


Figure 6: Front view of the CLX000 CAN-bus connector

Pin No.	Function
1	Not connected
2	CAN-L
(3)	GND (3 and 6 internally connected)
4	Not connected
5	Not connected
(6)	GND (3 and 6 internally connected)
7	CAN-H
8	Not connected
9	Device supply (see section 3 on page 2)

Table 3: CLX000 connector pin assignment

Note that only the pins with underlined numbers are used internally. The unused pins can be left unconnected. Pin 3 and 6 share the same function and are internally connected.

! Note that some CAN-buses might not include the supply wires to supply the CLX000. If this is the case, the device supply pins must be connected to an external supply.

! Note that the voltage supplied between pin 3 (or 6) and 9 cannot exceed the limits given in section 3 on page 2. If the limits are exceeded, the device might get permanently damaged.

6 Memory Card Installation

The CLX000 uses a standard SD-card to store data[8]. The SD-card is placed inside the device and can be replaced. If not already present, an SD-card should be inserted into the device to enable data logging. To insert an SD-card, it is necessary to open the device.

! Please note that it is not necessary to open the device if it comes with a preinstalled SD-card.

! Please note that it is not necessary to remove the SD-card to access logged data. The device can be directly connected to the PC using a USB-cable as described in section 4.1.

To open the device a 4 mm flat-bladed screwdriver is needed. Insert the screwdriver into one of the opening slots as indicated in Figure 7.

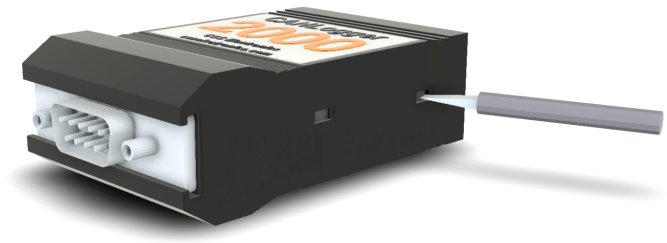


Figure 7: Side view of the CLX000 enclosure showing the opening slots

To open the device, gently twist the screwdriver to separate the device enclosure.

When open, locate the SD-card connector slot. The slot is of the push-push type (push to both insert and eject the card). Insert the SD-card into the connector slot and push until it clicks as illustrated in Figure 8.

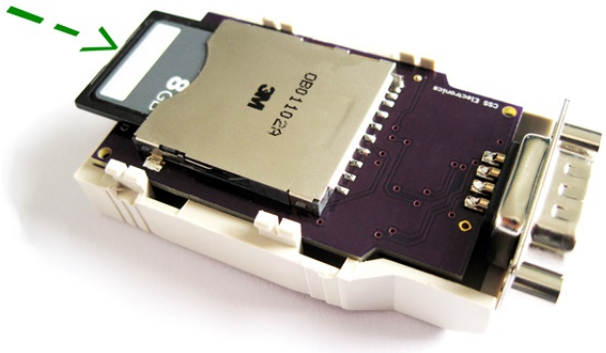


Figure 8: Simplified view of the CLX000 with one piece of the enclosure removed. The SD-card is pushed into the connector slot as indicated by the green arrow.

To close the device, make sure that the printed circuit board is perfectly aligned with the edges of the enclosure and then press the two pieces of the enclosure together. When the enclosure pieces snap together the device is closed.

[8] Refer to the device specification on page 2

7 Device Firmware Upgrade

The device supports USB Device Firmware Upgrade (DFU) such that new firmware updates can be uploaded to the device when new features become available. Firmware can be uploaded to the device using tools supporting the DFU protocol.

The device will only be able to accept new firmware images when in DFU mode, therefore, to enter DFU mode, the device should be powered on with the DFU button pressed (the device enclosure exposes three LEDs and one DFU button slot), as illustrated in Figure 9. The sequence for entering DFU mode and updating firmware is:

1. Make sure that device is powered completely off
2. Gently push and hold the DFU button, as illustrated in Figure 9
3. Connect the USB cable to power on the device
4. The device will power on and enter DFU mode (red LED constant on, green and yellow LEDs off)
5. Release the DFU button
6. Open tool with DFU support
7. Select new compatible firmware image and upload image to device
8. When upload is complete, power cycle the device to start the new firmware

! Note that the device will only accept officially released firmware images

! Please make sure to use a non-conductive tool to activate the DFU button

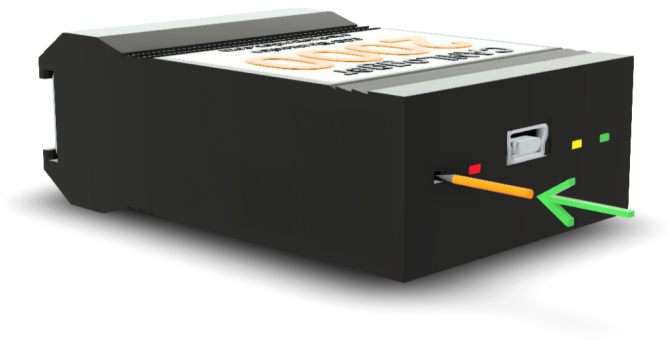


Figure 9: The DFU button is activated with a pointed non-conducting object.

8 Serial Interface

The CLX000 supports a serial interface which can be accessed using the USB connection. The interface can be used to communicate with the CLX000 in real time. Refer to section 8.1.3 for supported application interfaces.

To enable the serial interface, the device should be powered up using the CAN-bus connector. When the device has booted, the the USB-cable can be connected to activate the interface.

On MS Windows, the active serial interface will be listed as a "COM-port" in the device manager.

8.1 Serial Protocol

The serial protocol consists of three simple layers: Physical Layer (PHY, 8.1.1), Data-Link Layer (DLL, 8.1.2) and Application Layer (AL, 8.1.3). The layered model ensures effective and robust communication between the CLX000 and the host (e.g. PC).

In the following sections, the protocol layers will be described.

8.1.1 Physical layer

The USB connection creates a virtual serial interface with the following configuration:

- Bit-rate: 115200 bit/s
- Data bits: 8
- Stop bits: 1
- Parity: None
- Flow control: None

8.1.2 Data Link Layer

The purpose of the DLL is to ensure safe and error free transmission of message frames between the device and the host. The DLL ensures that the start/stop of a message frame can always be uniquely identified and that the integrity of the data payload can be verified by the receiver.

The design of the DLL is heavily inspired by the widely used HDLC specification. The DLL has the following structure:

Start flag	Payload	CRC		Stop flag
0x7E	...	MSB	LSB	0x7E

Frame identification

The start and end of a frame are always uniquely defined by the start/stop flag with a unique one-byte value 0x7E.

Byte stuffing is used to avoid that the start/stop byte appears randomly in the message Payload or CRC fields (which could lead to confusion regarding the start/stop of the frame). The CLX000 uses a control byte with value 0x7D to escape occurrences of the unique start/stop byte in the data.

To decode the message frame, the receiver shall examine the data between the start/stop flags. If the control byte is detected, the received shall perform the following actions:

1. Discard the control byte (0x7D)

2. Restore the following byte by complementing its 5th bit (indexed from zero)

Frame integrity verification

When the frame has been destuffed, the integrity of the data can be verified using the CRC field. To verify the integrity of the Payload, the receiver shall perform the following actions:

1. Calculate the CRC16 checksum over the Payload
2. Compare the calculated checksum with the checksum stored in the CRC field (2 bytes)
3. If the calculated and the stored values are equal, the frame is valid

The CRC16 calculation uses the standard CRC-16 polynomial (also known as CRC-16-IBM):

$$x^{16} + x^{15} + x^2 + 1$$

The polynomial representation is 0x8005.

The details of the CRC calculation are beyond the scope of this manual.

Example 1 - Decoding of received message frame

The following message frame is received on the serial interface, indicated by the start/stop byte pair:

Start flag	Stuffed frame data										Stop flag
7E	AA	BB	CC	7D	5E	DD	EE	FF	A3	84	7E

Scanning the bytes in between the start/stop flags reveals one occurrence of the control byte 0x7D. By following the destuffing steps, the control byte is removed and the following byte restored to the original value (the value before the stuffing performed by the `CLX000`) by complementing the 5th bit:

Start flag	Destuffed frame data									Stop flag
	Payload							CRC		
7E	AA	BB	CC	7E	DD	EE	FF	A3	84	7E

To validate the integrity of the Payload, the CRC16 is calculated over the Payload bytes:

$$\text{CRC16}(AA, BB, CC, 7E, DD, EE, FF) = A384.$$

The validation shows that the Payload is valid as the calculated and the stored CRC values are equal.

8.1.3 Application Layer

The AL is the decoded Payload of the DLL and contains an application ID (1 byte) and the application data:

ID	Application data

The interpretation of the application data is given by the ID field. Below are the supported application IDs listed:

ID	Description
1	Received CAN-bus message
2	Transmitted CAN-bus message

The following describe above listed application data interpretations.

Received CAN-bus message

Description: The application data contain a CAN-bus message received by the logger.

Format:

ID	Application data				
1	Time	Time ms	Message ID	Data length	Data
	4 byte	2 byte	4 byte	1 byte	0..8 byte

The `Time` field is encoded as "Epoch" seconds. The message ID is extended if bit 29 (indexed from zero) is set. Multi-byte fields shall be interpreted MSB (Most-Significant- Byte) first.

Transmitted CAN-bus message

Description: The application data contain a CAN-bus message transmitted by the logger.

Format:

ID	Application data				
2	Time	Time ms	Message ID	Data length	Data
	4 byte	2 byte	4 byte	1 byte	0..8 byte

The `Time` field is encoded as "Epoch" seconds. The message ID is extended if bit 29 (indexed from zero) is set. Multi-byte fields shall be interpreted MSB (Most-Significant- Byte) first.

9 Wireless Connectivity

The CL3000 supports wireless connectivity over WiFi. The WiFi specifications are given in Section 3.3.

9.1 WiFi mode

The CL3000 can be configured to function either as an Access Point (AP mode) or as a Station (STA mode). When in AP mode, the device creates a local WiFi access point and accepts incoming connections. In STA mode, the device functions as a wireless LAN client and connects to an existing access point. The wireless functionality is active in both MSD and logging mode.

The AP and STA network topologies are illustrated in Figure 10 and 11 respectively.

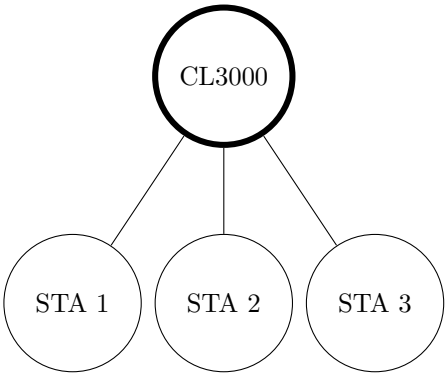


Figure 10: CL3000 in AP mode

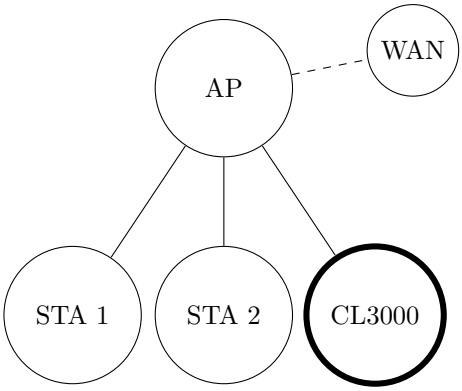


Figure 11: CL3000 in STA mode

Refer the Section 11 for specifics on how to configure the wireless features.

! A change to the wireless configuration is loaded next time the CL3000 is booted in logging mode.

9.1.1 AP mode

When configured in AP mode, the logger creates a local wireless access point. The access point supports up to seven connected clients, such as PCs, tablets and smart-phones.

The access points is protected using WiFi security type WPA2. The access point name and password are configurable. Network nodes connecting to the device access point are automatically assigned an IP-address by a DHCP service running on the device.

! Note that the access point created by the CL3000 does not provide internet access to connected devices (as illustrated in Figure 10).

9.1.2 STA mode

When configured in STA mode, the logger attempts to connect to an existing access point as a wireless LAN client. The name and password of the access point to connect to is set in the device configuration file. It is possible to connect to access points using WiFi security types WEP, WPA or WPA2. The CL3000 expects to be assigned an IP-address by a DHCP service running on the access point.

When connected to an access point, the CL3000 is accessible by all devices on the local network. If the access point provides internet access (*WAN* in Figure 11), then the CL3000 is remotely accessible by any internet connected device[9].

! In STA mode, the CL3000 will be accessible by all stations connected to the same network.

! If multiple loggers are setup to connect to the same local network, make sure each uses an unique network application name (host names).

! The wireless range in STA mode is generally better compared to AP mode.

9.1.3 Web server

The CL3000 implements a build in web server, which can be accessed in both AP and STA mode.

The web server hosts one of more web pages which can be accessed through any standard web browser. The main web page can always be accessed directly using either the CL3000 IP-address or the application name (as set in the configuration file).

The main web page lists the log files on the device which are ready to download and links to any additional pages running on the CL3000.

Example 1 - Web page access in AP mode

The APPNAME field in the configuration file is set to "id0001".

The CLX000 main web page can be accessed by entering either of the following into a web browser address bar:

- http://id0001/
- http://192.168.0.1/

! In AP mode, the CL3000 is on IP-address 192.168.0.1

Example 2 - Web page access in STA mode

The APPNAME field in the configuration file is set to "id0001" and the CL3000 has been assigned IP-address 192.168.1.100 by the access point DHCP service.

The device main web page can be accessed by entering either of the following into a web browser address bar:

- http://id0001/
- http://192.168.1.100/

! The IP-address assigned to the CL3000 may change on each connection to the access point.

9.2 FTP remote push service

Refer to section 11 on page 22 for more information on the FTP push service configuration.

The CL3000 supports automatic push of log files to a remote server using the FTP protocol.

The FTP push service is active in logging mode if ftpEnb is set to true (refer to the configuration section). When a log file is closed (e.g. when fileSplitLimit is reached), the CL3000 will automatically attempt to upload the file to the remote server.

! The final log file in a logging session is closed when power is disconnected. This file will be uploaded the next time the logger is powered on.

[9] Remote access may require some additional configuration of the access point

9.2.1 File naming

Uploaded log file names are prefixed the logger ID (can be set in the configuration file), e.g. log file "0000001.TXT" is uploaded as "myloggerid_0000001.TXT". Prefixing the logger ID prevents file name collisions on the server in setups where multiple loggers upload to the same server. The `serverDir` setting can be used to further separate log files from different loggers into separate folders on the server.

! Use unique logger IDs to avoid name collisions when multiple loggers upload to the same FTP server.

! Set a server upload directory to upload log files to a specific folder on the server.

9.2.2 FTP push network topology

FTP push allows the logger to push log files to a local or a remote FTP server. In both cases, the logger WiFi shall be configured in STA mode (refer to Section 9.1.2) to enable access to the FTP server. The local and remote FTP server network topologies are described in detail in the following.

Local network topology

The local network topology is illustrated in Figure 12. In the local topology, the logger does not require internet access, as the FTP server is located on the same local network.

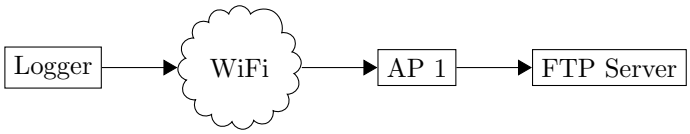


Figure 12: FTP push network local

The following points shall be considered:

- The logger WiFi connection in STA mode shall be setup to connect to AP1 (refer to Section 9.1.2)
- AP1 firewalls etc. shall allow FTP traffic on port 21
- Any firewalls running on the FTP server shall allow traffic on port 21
- The FTP login credentials (user name and password) used by the logger, shall allow the logger to create and modify files in the upload directory on the FTP server
- The log file upload directory shall exist on the FTP server (logger cannot create new directories)

! If multiple loggers are used, the FTP server should either allow multiple connections using a shared login or provide separate logins for each logger

Remote network topology

The remote network topology is illustrated in Figure 13. In this setup, the FTP server is not located within the same local network as the logger. The internet is used to connect the local network of the logger with the local network of the FTP server. This setup is more complicated compared with the simpler local setup explained above. With the remote setup, it is possible to push log files to an FTP server located anywhere in the world.

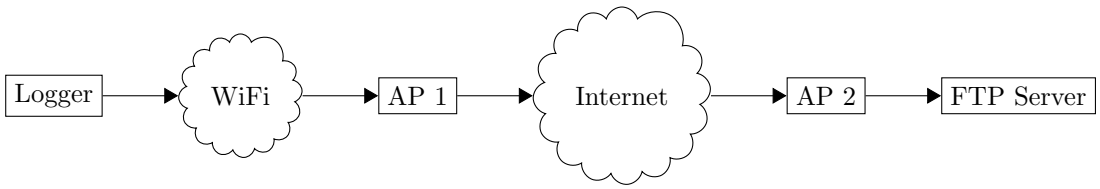


Figure 13: FTP push network remote

The following points shall be considered in addition to the points listed in the local setup:

- AP1 shall provide access to the internet
- AP2 firewalls etc. shall allow incoming traffic on port 21
- AP2 shall forward incoming traffic on port 21 to the FTP server (port forwarding)

! If a hosted FTP service is used, then the points above is likely setup correctly as default.

FTP push sequence

The complete FTP push sequence is illustrated in Figure 14 and a detailed description is given below. The sequence runs continuously in loop when the FTP feature is enabled.

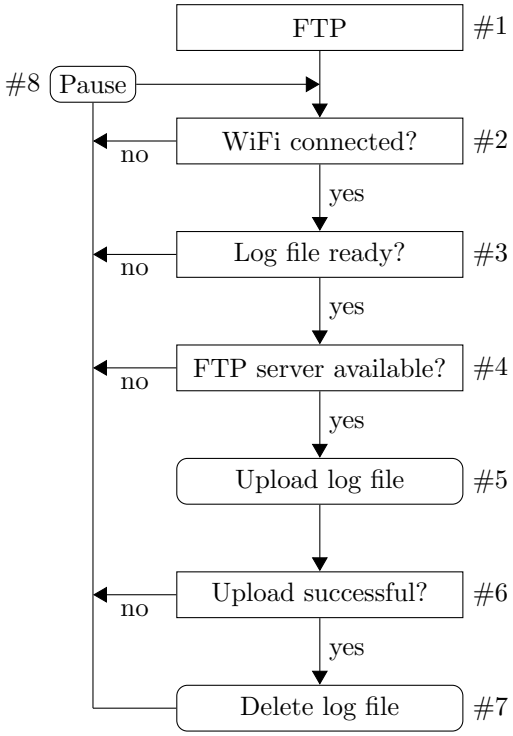


Figure 14: FTP push sequence

1. The FTP push sequence runs if FTP is enabled in the `CL3000` configuration file.
2. Is the hotspot set in the `CL3000` configuration file in range?
3. Is a (or more) log file ready to be uploaded to the server?
4. Is the remote FTP server available?
5. The log file is uploaded to the server. This can take several minutes depending on the log file size and the transfer speed.

6. Did the upload complete successfully?
7. The uploaded log file is deleted from the `CL3000` if it was successfully uploaded.
8. Pauses a few seconds before next cycle

! Note that log files are only deleted when they have successfully transferred to the server

10 Log File Format

The following section describes the log file format.

An example log file is given for reference below:

```
# Logger type: CL2000
# HW rev: 6.xx
# FW rev: 5.31
# Logger ID: OPEL
# Session No.: 70
# Split No.: 1
# Time: 20170101T022232
# Value separator: ";"
# Time format: 4
# Time separator: ""
# Time separator ms: ""
# Date separator: ""
# Time and date separator: "T"
# Bit-rate: 500000
# Silent mode: false
# Cyclic mode: false
Timestamp;Type;ID;Data
01T022231683;8;7df;02010d5555555555
01T022231688;0;7e8;03410d00aaaaaaaa
01T022231693;8;7df;02010c5555555555
01T022231701;0;7e8;04410c0000aaaaaa
01T022231783;8;7df;02010d5555555555
01T022231788;0;7e8;03410d00aaaaaaaa
01T022231793;8;7df;02010c5555555555
01T022231801;0;7e8;04410c0000aaaaaa
```

10.1 File format

Log files are written in a CSV-like format using the reduced ASCII character set. The log files can be opened directly in most simple text editors. Log entries are formatted according to the settings in the configuration file and terminated by a single "line feed" character.

! Note that some text editors do not recognise a single "line feed" character as a new line.

10.2 File name

The log file names are built using a unique sequential 7 digit number. Each time a new log file is created, the log file name number is increased by one. The log file extension is *.txt, e.g. "0000001.txt".

! Note that the log name number is **not** reset on a power-cycle.

10.3 Header

The log file header is added to all log files. The header contains information needed to identify and interpret the log file. Log file header lines start with "#".

Below the log header entries are described:

- `Logger type`: The `CLX000` type
- `HW rev`: The hardware revision of the `CLX000`
- `FW rev`: The running firmware revision of the `CLX000`
- `Logger ID`: The logger identification string

- `Session No.`: The session sequential number
- `Split No.`: The log file split number
- `Time`: The log file start time with fixed format "YYYYMMDDThhmmss"
- `Value separator`: The value separator used in the log file
- `Time format`: The time stamp format used in the log file
- `Time separator`: The timestamp time separator
- `Time separator ms`: The timestamp ms separator
- `Date separator`: The timestamp date separator
- `Time and date separator`: The timestamp time to date separator
- `Bit-rate`: The manually set or auto detected CAN-bus bit rate
- `Silent mode`: The state of silent mode
- `Cyclic mode`: The state of cyclic mode

10.3.1 Log file numbering

Session No.
Each logging session is given a unique number. Each time the logger powers up, the session number is increased by one. The session number can be used to group multiple log files from the same logging session (log sessions can be split across several log files, configured by the `fileSplitSize` field).

! Note that the session number is **not** reset on a power-cycle.

Split No.
The split number is reset each time the logger powers up. The purpose of the `Split No.` is to ensure an unambiguous order of log files within the same logging session (log files with the same `Session No.`)

10.3.2 Log entry format

The format of the log file data is unambiguously given by the `Time format` and separator entries in the log file.

10.4 Data fields

The date field header identifies the column values and how they are separated (`Value separator`). Which date fields to include in log files can be set in the configuration file.

The following data fields can be included in the log file:

- `Timestamp`: Timestamp in ms (1ms resolution)
- `Lost`: Message lost flag. If the flag is set (1 in log file), one (or more) CAN-message(s), before the message with the flag, was (were) potentially lost
- `Type`: Message type indicator. Is either 0 (received standard), 1 (received extended), 8 (transmitted standard) or 9 (transmitted extended) in the log file
- `ID`: CAN-bus message identifier
- `Length`: The length in bytes of the CAN-message data field

- `Data`: CAN-bus message data bytes (0-8)

All fields except the timestamp are written in hexadecimal format. To ensure unambiguous interpretation, the data field bytes are left zero padded, such that each byte is always written using two characters, e.g. ”010203”.

! Note that the timestamp is either relative to power up or absolute depending on the `CLX000` hardware (real-time-clock).

11 Device Configurations

The device configuration file (*.ini file format) is used to configure the behaviour of the CLX000. The configuration file can be accessed from a PC when the device is in MSD-mode. To restore the default configuration, delete the configuration from the device storage and power cycle.

The default configuration file of the CL1000 is:

```
; CLX000 configuration file

[revision]                    ; Configuration file revision (do not modify)
revision = 12                 ; Configuration revision number

[log]                         ; Log file configuration
loggerID = id0001             ; Logger identification string (max 10 characters)
loggingEnb = true             ; Default logging state
valueSeparator = 59           ; Log file separator ASCII char (DEC)
timestampFormat = 4           ; Timestamp format ( 0 = kkk, 6 = YYYYMMDDhhmmsskkk)
timestampTimeSeparator = 0    ; Timestamp time separator ASCII char (DEC, 0 = none)
timestampTimeMsSeparator = 0  ; Timestamp millisecond separator ASCII char (DEC, 0 = none)
timestampDateSeparator = 0    ; Timestamp date separator ASCII char (DEC, 0 = none)
timeTimeDateSeparator = 84    ; Timestamp date / time separator ASCII char (DEC, 0 = none)
fileSplitLimit = 100          ; File split size in MB
cyclicLogging = false         ; Delete oldest stored file when full
compression = false           ; Enable data compression (experimental)

[heartbeat]
heartbeatEnb = false          ; Enable heartbeat signal
extendedID = true             ; Use extended 29 bit message ID (2.0B)
msgID = 00435353              ; CAN message ID of heartbeat signal (HEX)

[control]
controlEnb = false            ; Enable control signal
extendedID = true             ; Use extended 29 bit message ID (2.0B)
msgID = 00435354              ; CAN message ID of control signal (HEX)

[dataFields]                  ; Data fields in log file
timestamp = true              ; Log message timestamp
lost = false                  ; Log if a message was lost
type = true                   ; Log type of identifier
id = true                     ; Log message ID
dataLength = false            ; Log number of message data bytes
data = true                   ; Log message data

[can]                         ; CAN bus configuration
bitrate = 0                   ; CAN-bus bit rate (DECIMAL), 0 = auto-detect
silent = false                ; Listen-only mode

[channel1]                    ; CAN channel 1 configuration
channelEnb = true             ; Enable CAN channel
destination = 3               ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false            ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1       ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false          ; Enable below message filtering
msgID = 00000000              ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF          ; Message ID filter mask (HEX) (0 is invalid)

[channel2]                    ; CAN channel 2 configuration
channelEnb = true             ; Enable CAN channel
destination = 3               ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = true             ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1       ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false          ; Enable below message filtering
msgID = 00000000              ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF          ; Message ID filter mask (HEX) (0 is invalid)

[channel3]                    ; CAN channel 3 configuration
channelEnb = false            ; Enable CAN channel
destination = 3               ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false            ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1       ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = true           ; Enable below message filtering
msgID = 00000001              ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF          ; Message ID filter mask (HEX) (0 is invalid)
```

```
[channel4]                      ; CAN channel 4 configuration
channelEnb = false              ; Enable CAN channel
destination = 3                 ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false              ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1         ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = true             ; Enable below message filtering
msgID = 00000002                ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF            ; Message ID filter mask (HEX) (0 is invalid)

;[transmit1]                    ; Transmit message 1
;destination = 0                ; 0 = None, 1 = Logger, 2 = Interface, 3 = Both
;period = 100                   ; Period in ms (DEC, 10 ms resolution)
;delay = 0                      ; Delay in ms (DEC, 10 ms resolution)
;extendedID = false             ; Use extended 29 bit message IDs (2.0B)
;msgID = 00000001               ; Transmit message ID (HEX)
;msgData = {0102030405060708}   ; Message data (HEX)
```

The default configuration file of the `CL2000` is:

```
; CLX000 configuration file

[revision]                  ; Configuration file revision (do not modify)
revision = 12               ; Configuration revision number

[log]                       ; Log file configuration
loggerID = id0001           ; Logger identification string (max 10 characters)
loggingEnb = true           ; Default logging state
valueSeparator = 59         ; Log file separator ASCII char (DEC)
timestampFormat = 4         ; Timestamp format ( 0 = kkk, 6 = YYYYMMDDhhmmsskkk)
timestampTimeSeparator = 0  ; Timestamp time separator ASCII char (DEC, 0 = none)
timestampTimeMsSeparator = 0 ; Timestamp millisecond separator ASCII char (DEC, 0 = none)
timestampDateSeparator = 0  ; Timestamp date separator ASCII char (DEC, 0 = none)
timeTimeDateSeparator = 84  ; Timestamp date / time separator ASCII char (DEC, 0 = none)
fileSplitLimit = 100        ; File split size in MB
cyclicLogging = false       ; Delete oldest stored file when full
compression = false         ; Enable data compression (experimental)

[heartbeat]
heartbeatEnb = false        ; Enable heartbeat signal
extendedID = true           ; Use extended 29 bit message ID (2.0B)
msgID = 00435353            ; CAN message ID of heartbeat signal (HEX)

[control]
controlEnb = false          ; Enable control signal
extendedID = true           ; Use extended 29 bit message ID (2.0B)
msgID = 00435354            ; CAN message ID of control signal (HEX)

[dataFields]                ; Data fields in log file
timestamp = true            ; Log message timestamp
lost = false                ; Log if a message was lost
type = true                 ; Log type of identifier
id = true                   ; Log message ID
dataLength = false          ; Log number of message data bytes
data = true                 ; Log message data

[can]                       ; CAN bus configuration
bitrate = 0                 ; CAN-bus bit rate (DECIMAL), 0 = auto-detect
silent = false              ; Listen-only mode

[channel1]                  ; CAN channel 1 configuration
channelEnb = true           ; Enable CAN channel
destination = 3             ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false          ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1     ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false        ; Enable below message filtering
msgID = 00000000            ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF        ; Message ID filter mask (HEX) (0 is invalid)

[channel2]                  ; CAN channel 2 configuration
channelEnb = true           ; Enable CAN channel
destination = 3             ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = true           ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1     ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false        ; Enable below message filtering
msgID = 00000000            ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF        ; Message ID filter mask (HEX) (0 is invalid)

[channel3]                  ; CAN channel 3 configuration
channelEnb = false          ; Enable CAN channel
destination = 3             ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false          ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1     ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = true         ; Enable below message filtering
msgID = 00000001            ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF        ; Message ID filter mask (HEX) (0 is invalid)

[channel4]                  ; CAN channel 4 configuration
channelEnb = false          ; Enable CAN channel
destination = 3             ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false          ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1     ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = true         ; Enable below message filtering
msgID = 00000002            ; Message ID filter (HEX)
```

```
msgIDMask = 1FFFFFFF          ; Message ID filter mask (HEX) (0 is invalid)

;[transmit1]                  ; Transmit message 1
;destination = 0              ; 0 = None, 1 = Logger, 2 = Interface, 3 = Both
;period = 100                 ; Period in ms (DEC, 10 ms resolution)
;delay = 0                    ; Delay in ms (DEC, 10 ms resolution)
;extendedID = false           ; Use extended 29 bit message IDs (2.0B)
;msgID = 00000001             ; Transmit message ID (HEX)
;msgData = {0102030405060708} ; Message data (HEX)

[RTC]                         ; Real Time Clock
epochTime = 0                 ; Epoch time (0 to keep current time)
adjustment = 0                ; Adjustment in sec to current time (max +-129600 sec)
```

The default configuration file of the `CL3000` is:

```
; CLX000 configuration file

[revision]                    ; Configuration file revision (do not modify)
revision = 12                 ; Configuration revision number

[log]                         ; Log file configuration
loggerID = id0001             ; Logger identification string (max 10 characters)
loggingEnb = true             ; Default logging state
valueSeparator = 59           ; Log file separator ASCII char (DEC)
timestampFormat = 4           ; Timestamp format ( 0 = kkk, 6 = YYYYMMDDhhmmsskkk)
timestampTimeSeparator = 0    ; Timestamp time separator ASCII char (DEC, 0 = none)
timestampTimeMsSeparator = 0  ; Timestamp millisecond separator ASCII char (DEC, 0 = none)
timestampDateSeparator = 0    ; Timestamp date separator ASCII char (DEC, 0 = none)
timeTimeDateSeparator = 84    ; Timestamp date / time separator ASCII char (DEC, 0 = none)
fileSplitLimit = 100          ; File split size in MB
cyclicLogging = false         ; Delete oldest stored file when full
compression = false           ; Enable data compression (experimental)

[heartbeat]
heartbeatEnb = false          ; Enable heartbeat signal
extendedID = true             ; Use extended 29 bit message ID (2.0B)
msgID = 00435353              ; CAN message ID of heartbeat signal (HEX)

[control]
controlEnb = false            ; Enable control signal
extendedID = true             ; Use extended 29 bit message ID (2.0B)
msgID = 00435354              ; CAN message ID of control signal (HEX)

[dataFields]                  ; Data fields in log file
timestamp = true              ; Log message timestamp
lost = false                  ; Log if a message was lost
type = true                   ; Log type of identifier
id = true                     ; Log message ID
dataLength = false            ; Log number of message data bytes
data = true                   ; Log message data

[can]                         ; CAN bus configuration
bitrate = 0                   ; CAN-bus bit rate (DECIMAL), 0 = auto-detect
silent = false                ; Listen-only mode

[channel1]                    ; CAN channel 1 configuration
channelEnb = true             ; Enable CAN channel
destination = 3               ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false            ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1       ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false          ; Enable below message filtering
msgID = 00000000              ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF          ; Message ID filter mask (HEX) (0 is invalid)

[channel2]                    ; CAN channel 2 configuration
channelEnb = true             ; Enable CAN channel
destination = 3               ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = true             ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1       ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false          ; Enable below message filtering
msgID = 00000000              ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF          ; Message ID filter mask (HEX) (0 is invalid)

[channel3]                    ; CAN channel 3 configuration
channelEnb = false            ; Enable CAN channel
destination = 3               ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false            ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1       ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = true           ; Enable below message filtering
msgID = 00000001              ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF          ; Message ID filter mask (HEX) (0 is invalid)

[channel4]                    ; CAN channel 4 configuration
channelEnb = false            ; Enable CAN channel
```

```
destination = 3                 ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false              ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1         ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = true             ; Enable below message filtering
msgID = 00000002                ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF            ; Message ID filter mask (HEX) (0 is invalid)

;[transmit1]                    ; Transmit message 1
;destination = 0                ; 0 = None, 1 = Logger, 2 = Interface, 3 = Both
;period = 100                   ; Period in ms (DEC, 10 ms resolution)
;delay = 0                      ; Delay in ms (DEC, 10 ms resolution)
;extendedID = false             ; Use extended 29 bit message IDs (2.0B)
;msgID = 00000001               ; Transmit message ID (HEX)
;msgData = {0102030405060708}   ; Message data (HEX)

[RTC]                           ; Real Time Clock
epochTime = 0                   ; Epoch time (0 to keep current time)
adjustment = 0                  ; Adjustment in sec to current time (max +-129600 sec)

[WIFI]
appName = id0001                ; Application name (1-15 characters)
appMode = 4                     ; WIFI mode (4 = Access Point, 5 = Station)
appSSID = id0001                ; WIFI SSID (1-32 characters)
appNetworkKey = 12345678        ; WIFI password (8-32 characters)
httpdMode = 0                   ; Use access authentication (0 = no, 1 = yes)
httpdUser = user                ; Access authentication user name (0-32 characters)
httpdPass = 12345678            ; Access authentication password (0-32 characters)

[FTP]
ftpEnb = false                  ; Enable FTP upload service
serverAddr = 192.168.1.1        ; Hostname / IP of the FTP server (0-32 characters)
serverDir = /                   ; Directory on FTP server (0-32 characters)
user = user                     ; FTP user name (0-32 characters)
pwd = 12345678                  ; FTP password (0-32 characters)
```

11.1 Configurations file description

The INI file format is a simple text based configuration file format which can be edited by any simple text editor (such as MS Notepad). The configuration of the `CLX000` uses the INI file format and syntax. Comments are given by the semicolon symbol (;) and can be added by the user as helpful notes directly in the configuration. Comments will not be interpreted by the device and have not to follow any specific syntax.

Below each section of the configuration file are explained (in the INI file syntax, a section is marked by square brackets [*section name*]):

Configuration file revision:

```
[revision]                    ; Configuration file revision (do not modify)
revision = 12                 ; Configuration revision number
```

The [revision] section should not be modified and is simply used to ensure compatibility between the running device firmware and the format of the configuration file.

Log file configuration:

```
[log]                         ; Log file configuration
loggerID = id0001             ; Logger identification string (max 10 characters)
loggingEnb = true             ; Default logging state
valueSeparator = 59           ; Log file separator ASCII char (DEC)
timestampFormat = 4           ; Timestamp format ( 0 = kkk, 6 = YYYYMMDDhhmmsskkk)
timestampTimeSeparator = 0    ; Timestamp time separator ASCII char (DEC, 0 = none)
timestampTimeMsSeparator = 0  ; Timestamp millisecond separator ASCII char (DEC, 0 = none)
timestampDateSeparator = 0    ; Timestamp date separator ASCII char (DEC, 0 = none)
timeTimeDateSeparator = 84    ; Timestamp date /time separator ASCII char (DEC, 0 = none)
fileSplitLimit = 100          ; File split size in MB
cyclicLogging = false         ; Delete oldest stored file when full
compression = false           ; Enable data compression (experimental)
```

The [log] section specifies some general settings of the log file. The section contains the following entries:

- `loggerID`: A logger identification string. The string is copied to the log file header such that log files from different loggers can be identified. The string can be up to 10 characters long.
- `loggingEnb`: Sets the default logging state. If the default logging state is set to `false`, the logger will not log messages until it has received a valid control signal enabling logging.
- `valueSeparator`: Specifies the separator symbol used in the log file. Each separator symbol is represented by a value (ASCII). Valid symbol values are listed in Table 4.
- `timestampFormat`: Specifies the detail level of the timestamp used in the log file. Note that a longer timestamp will take up more space and reduce maximum logging speed. The format supports seven levels, from 0 to 6. Timestamp separators can be selected using the time separators settings. The detail levels are listed below with *kkk* as milliseconds:
  0. kkk
  1. sskkk
  2. mmsskkk
  3. hhmmsskkk
  4. DDhhmmsskkk
  5. MMDDhhmmsskkk
  6. YYYYMMDDhhmmsskkk

- `timestampTimeSeparator`: Specifies the time separator in the timestamp. Each separator symbol is represented by a value (ASCII). Valid symbol values are listed in Table 4. Additionally, setting the separator to 0 will disable the separator. The separator will be inserted into the timestamp as:
  YYYYMMDDhhXmmXsskkk.
- `timestampTimeMsSeparator`: Specifies the millisecond separator in the timestamp. Each separator symbol is represented by a value (ASCII). Valid symbol values are listed in Table 4. Additionally, setting the separator to 0 will disable the separator. The separator will be inserted into the timestamp as:
  YYYYMMDDhhmmssXkkk
- `timestampDateSeparator`: Specifies the date separator in the timestamp. Each separator symbol is represented by a value (ASCII). Valid symbol values are listed in Table 4. Additionally, setting the separator to 0 will disable the separator. The separator will be inserted into the timestamp as:
  YYYYXMMXDDhhmmsskkk
- `timeTimeDateSeparator`: Specifies the date to time separator in the timestamp. Each separator symbol is represented by a value (ASCII). Valid symbol values are listed in Table 4. Additionally, setting the separator to 0 will disable the separator. The separator will be inserted into the timestamp as:
  YYYYMMDDXhhmmsskkk
- `fileSplitLimit`: Specifies the maximum file size of log files stored on the device in MB. When the file split size is reached, a new file is created, and the logging continues. Each new split log file is given an updated header. The file split limit shall be in the range 1 - 512 MB.
- `cyclicLogging`: Setting this entry to `true` enables cyclic logging mode. With cycling logging mode enabled the oldest log files are deleted when the memory card becomes full - allowing the logging to continue.
- `compression`: Setting this entry to `true` enables data compression. The compression feature is currently **experimental**.

Table of valid separator symbol values:

32: (space)	42: *	52: 4	62: >	72: H	82: R	92: \	102: f	112: p	122: z
33: !	43: +	53: 5	63: ?	73: I	83: S	93:]	103: g	113: q	123: {
34: ”	44: ,	54: 6	64: @	74: J	84: T	94: ^	104: h	114: r	124: |
35: #	45: -	55: 7	65: A	75: K	85: U	95: _	105: i	115: s	125: }
36: $	46: .	56: 8	66: B	76: L	86: V	96: ‘	106: j	116: t	126: ~
37: %	47: /	57: 9	67: C	77: M	87: W	97: a	107: k	117: u	
38: &	48: 0	58: :	68: D	78: N	88: X	98: b	108: l	118: v	
39: ’	49: 1	59: ;	69: E	79: O	89: Y	99: c	109: m	119: w	
40: (	50: 2	60: <	70: F	80: P	90: Z	100: d	110: n	120: x	
41:)	51: 3	61: =	71: G	81: Q	91: [	101: e	111: o	121: y	

Table 4: List of valid separators

! Consider not to use the value separator (`valueSeparator`) in the timestamp

Example 1 - Timestamp settings

```
timestampFormat = 2              ; mmsskkk
timestampTimeSeparator = 0       ; No separator
timestampTimeMsSeparator = 0     ; No separator
timestampDateSeparator = 0       ; No separator
timeTimeDateSeparator = 0        ; No separator
```

Using the above settings, the date and time 2016/06/05 12:30:45,525 will in the log file be formatted as 3045525

These settings will result in a short timestamp suitable for high-frequency logging.

Example 2 - Timestamp settings

```
timestampFormat = 4             ; DDhhmmsskkk
timestampTimeSeparator = 0      ; No separator
timestampTimeMsSeparator = 0    ; No separator
timestampDateSeparator = 0      ; No separator
timeTimeDateSeparator = 84      ; 'T' As date to time separator
```

Using the above settings, the date and time 2016/06/05 12:30:45,525 will in the log file be formatted as 05T123045525

Note that these settings comply with the ISO 8601 time format standard.

Example 3 - Timestamp settings

```
timestampFormat = 6             ; YYYYMMDDhhmmsskkk
timestampTimeSeparator = 58     ; ':' As time separator
timestampTimeMsSeparator = 44   ; ',' As millisecond separator
timestampDateSeparator = 47     ; '/' As date separator
timeTimeDateSeparator = 32      ; ' ' (space) As date to time separator
```

Using the above settings, the date and time 2016/06/05 12:30:45,525 will in the log file be formatted as 2016/06/05 12:30:45,525

Note that with these settings, the raw log file can be directly loaded into MS-Excel (assuming that the MS-Excel decimal delimiter is set to ',').

Heartbeat signal:

```
[heartbeat]
heartbeatEnb = false          ; Enable heartbeat signal
extendedID = true             ; Use extended 29 bit message ID (2.0B)
msgID = 00435353              ; CAN message ID of heartbeat signal (HEX)
```

The device can transmit a 1-second periodic heartbeat signal. The signal payload contains logging state (enabled/disabled), the device wall time (if supported by the hardware) and space left on the memory card in MB. The section contains the following entries:

- `heartbeatEnb`: Set to `true` to enable the periodic heartbeat signal.
- `extendedID`: If set to `true`, the signal message will be using extended ID format (29-bit).
- `msgID`: The message ID used for the heartbeat signal.

The interpretation of the 8-byte data payload of the heartbeat signal is given below:

Byte No.	0	1	2	3	4	5	6	7
Content	0xAA	State	Epoch time				Space left	

Byte 0 has the reserved value 0xAA. Multi-byte fields should be interpreted MSB (Most-Significant-Byte) first.

Example - Heartbeat signal message

Byte No.	0	1	2	3	4	5	6	7
Content	0xAA	0x01	0x57	0x42	0x01	0x56	0x1D	0x93

Interpretation of example message payload:

- State (0x01): Logging state is active
- Epoch time (0x57420156): Device time in epoch format. Translates to 22/5/2016 18:58:30
- Space left (0x1D93): Space left on device is 7571 MB

! Note that the heartbeat signal will not be active until the logger has successfully received at least one message from the bus.

! The use of the heart beat signal requires that the `silent` entry in the [CAN] section is set to `false`.

! It is strongly recommended to disable the heartbeat signal in safety critical applications.

Control signal:

```
controlEnb = false              ; Enable control signal
extendedID = true               ; Use extended 29 bit message ID (2.0B)
msgID = 00435354                ; CAN message ID of control signal (HEX)
```

The device logging state (enabled/disabled) can be configured run-time using the control signal. The control signal can be used to enable logging only when CAN-bus messages should be captured and disable otherwise. Logging state is on power-on set as defined in the [log] section. The section contains the following entries:

- `controlEnb`: Set to `true` to enable the control signal.
- `extendedID`: If set to `true`, the signal message will be using extended ID format (29-bit).
- `msgID`: The message ID of the control signal.

The control signal payload should be of length 1. The `State` byte is set to 0x01 to enable logging and 0x00 to disable logging:

Byte No.	0
Content	State

When the device receives a valid control message, it will immediately set the logging state accordingly.

! Care should be taken when selecting a message ID of the control signal - if the message ID is used for another purpose, it might result in unexpected behaviour.

CAN message data fields:

```
[dataFields]                    ; Data fields in log file
timestamp = true                ; Log message timestamp
lost = false                    ; Log if a message was lost
type = true                     ; Log type of identifier
id = true                       ; Log message ID
dataLength = false              ; Log number of message data bytes
data = true                     ; Log message data
```

The [dataFields] section specifies which message fields that should go into the log file. Each entry can be set to either `true` or `false`. If an entry is set to `true` the message field will be included. The section contains the following entries:

- `timestamp`: Include timestamp
- `lost`: Include lost flag
- `type`: Include message type
- `id`: Include message identifier
- `dataLength`: Include message length
- `data`: Include message data

Refer to Section 10.4 on page 20 for more information on the fields.

CAN-bus configuration:

```
[can]                           ; CAN bus configuration
bitrate = 0                     ; CAN-bus bit rate (DECIMAL), 0 = auto-detect
silent = false                  ; Listen-only mode
```

The [can] section specifies the CAN-bus settings. The section contains the following entries:

- `bitrate`: Specifies the bit rate of the CAN-bus (in bit/s). The value cannot exceed the limits given in section 3 on page 2. If the value 0 is entered, the device will enable bit-rate auto detection and attempt to detect the bit-rate of the CAN-bus.

- `silent`: When silent is set to `true` the device logs CAN-bus traffic without interfering with the bus, i.e. the device does not acknowledge (ACK) messages placed on the CAN-bus by other devices.

! It is strongly recommended to enable silent mode for safety critical systems!

! The automatic CAN-bus bit-rate detection mechanism requires that at least two nodes communicate on the bus. The mechanism cannot be used if only the logger and a single node are connected to the bus.

Channel configuration (same for channel 1 to 4):

```
[channel1]                      ; CAN channel 1 configuration
channelEnb = true               ; Enable CAN channel
destination = 3                 ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false              ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1         ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false            ; Enable below message filtering
msgID = 00000000                ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF            ; Message ID filter mask (HEX) (0 is invalid)
```

The [channelX] sections configure the channels and the associated message filtering. A CAN-message will pass the filtering mechanism if the message ID is accepted by one of the four channel-filters. The section contains the following entries:

- `channelEnb`: Sets the enabled/disabled state of the specific channel (`true` is enable)
- `destination`: Messages can either be logged (1), send to the interface (2) or both (3)
- `extendedID`: Sets the ID format used by the channel (`true` is extended format)
- `downSamplePrescaler`: Down-samples the data according to the prescaler value
- `filteringEnb`: Enables the filtering mechanism
- `msgID`: The filter message ID
- `msgIDMask`: The filter ID mask

The down-sample prescaler can be used to limit the number of messages which are logged to the memory card. The prescaler down-samples incoming messages based on the message ID. A prescaler value of 1 effectively disables the prescaler and passes all messages (which are accepted by the channel filter). A prescaler value of 2 passes every second message with a specific message ID. The maximum value of the prescaler is 256. The prescaler mechanism is limited to 25 unique message IDs pr. channel. If more than 25 unique message IDs are received (and passed by the filter) on a specific channel, then only the 25 first received unique message IDs are down-sampled according to the prescaler value - the remaining messages are all passed (as if the prescaler is 1).

A filter is configured using a message ID-filter field (`msgID`) and a mask field (`msgIDMask`), both values are set as HEX values in the configuration. Both the ID-filter and the mask work on a bitwise level. The configuration of the filters is described in detail below.

! In the design of the filters it is helpful to use a tool which converts from and to binary, decimal and hexadecimal and which can perform the bitwise AND operation used in the filtering.

Below a few examples demonstrate the use of filters. The examples use x_y to indicate the number base, where $y = 2$ is binary, $y = 10$ is decimal and $y = 16$ is hexadecimal. Additionally "&" is used as a bitwise AND operation.

Example 1 - Filter configuration which accepts one specific message ID: $2000_{10} = 11111010000_2$

The filter is set to the value of the message to accept. The mask is set to all ones, such that all bits of the filter are considered, as given in (1).

Filter	11111010000_2		ID	11111010000_2	
Mask	$\&11111111111_2$	(1)	Mask	$\&11111111111_2$	(2)
Masked filter	11111010000_2		Masked ID	11111010000_2	

To test if the message passes the filter, we apply the mask to the message ID as given in (2). The masked filter and the masked ID are the same - the message will pass the filter.

The corresponding configuration of channel 1 is (note that the configuration uses HEX format):

```
[channel1]
channelEnb = true              ; The channel is enabled
destination = 1                ; The messages are logged
extendedID = false             ; The message uses 11 bit ID
downSamplePrescaler = 1        ; Down-sampling disabled
filteringEnb = true            ; Enable below message filtering
msgID = 7D0                    ; Message ID filter (HEX)
msgIDMask = 7FF                ; Message ID filter mask (HEX)
```

Note that the HEX value $7FF$ is simply 11 ones in binary - which is the ID bit-length of a standard CAN-message (11-bit identifier).

Example 2 - Filter configuration which accepts two message IDs: $2000_{10} = 11111010000_2$ and $2001_{10} = 11111010001_2$

Note that the two binary numbers are identical except for the rightmost bit. To design a filter which accepts both IDs, we can use the mask field to mask out the rightmost bit - such that it is not considered when the filter is applied.

In (3) the mask is set such that the rightmost bit is not considered (indicated in red).

Filter	11111010000_2		ID	11111010001_2	
Mask	$\&1111111111\color{red}{0}_2$	(3)	Mask	$\&1111111111\color{red}{0}_2$	(4)
Masked filter	11111010000_2		Masked ID	11111010000_2	

To test if the messages pass the filter, we apply the mask to the message ID 11111010001_2 as given in (4). The masked filter and the masked ID are the same - the message will pass the filter. Note that both 11111010000_2 and 11111010001_2 can pass the filter, as the rightmost bit is not considered by the filter (the rightmost bits are masked out).

The above examples demonstrate very simple uses of the channel filters. The filter and mask fields can be used to design more advanced filters to meet specific needs.

The four channels can each be configured using a filter and a mask. If a message passes one of the channel filters, it will be accepted. As indicated above, the filters can each be set to accept specific messages or a range of messages using the mask.

! It is recommended to filter out all unwanted messages. By filtering out unwanted messages, the risk of losing wanted messages becomes lower, and the logger will be able to log for a longer period before it runs out of storage memory.

Configuration of transmit messages:

```
[transmit1]                    ; Transmit message 1
destination = 0                ; 0 = None, 1 = Logger, 2 = Interface, 3 = Both
period = 100                   ; Period in ms (DEC, 10 ms resolution)
delay = 0                      ; Delay in ms (DEC, 10 ms resolution)
extendedID = false             ; Use extended 29 bit message IDs (2.0B)
msgID = 00000001               ; Transmit message ID (HEX)
msgData = {0102030405060708}   ; Message data (HEX)
```

The [transmitX] sections configure up to 20 transmit messages. The sections shall be named "transmit1" through "transmit20". Each section shall include all configuration fields given above.

Transmit messages are transmitted on the CAN-bus by the `CLX000` and can be setup to be either single shot or periodic. A transmit configuration is enabled by adding a [transmitX] section with fields to the config file and disabled by deleting the section.

! Note that the default configuration file contains a **commented-out** transmit message configuration. Remove the leading ";" comment markers to activate the configuration.

- `destination`: The destination of the transmit message, none (0), logger (1), interface(2), both (3). Note that all transmit messages are transmitted on the CAN-bus, regardless of the destination value
- `period`: Transmit period in steps of 10 ms in range 10 - 4294967290 ms (max ≈ 50 days). If set to zero, the message will be transmitted only once (single shot)
- `delay`: Delay time in ms in steps of 10 ms can be used to offset messages with the same period or to delay single shot messages. The delay shall be less than the set period
- `extendedID`: Use extended 29 bit message IDs (2.0B)
- `msgID`: Message ID
- `msgData`: Message data bytes as hexadecimal values enclosed by curly brackets. 0-8 data bytes can be set

! Make sure to change the section name when adding new transmit messages, e.g.: [transmit1],... [transmit2], etc.

Example 1 - One single shot message

```
[transmit1]
destination = 3
period = 0
delay = 6000
extendedID = false
msgID = 00000001
msgData = {01}
```

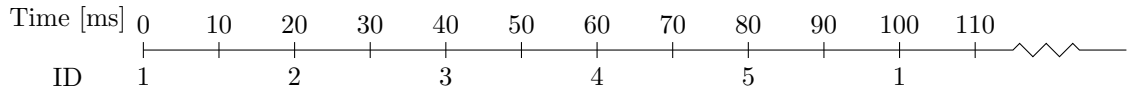
The period is set to zero to configure the transmit message as a single shot. With delay set to 6000, the single shot message is delayed 60 sec after power up. The message type is standard, the ID is 1, and the message data holds a single byte, 0x01.

Example 2 - Multiple periodic messages

```
[transmit1]
destination = 3
period = 100
delay = 0
extendedID = false
msgID = 00000001
msgData = {AA}

[transmit2]
destination = 3
period = 100
delay = 20
extendedID = false
msgID = 00000002
msgData = {AABB}

[transmit3]
destination = 3
period = 100
delay = 40
extendedID = false
msgID = 00000003
msgData = {AABBCCDD}

[transmit4]
destination = 3
```

```
period = 100
delay = 60
extendedID = false
msgID = 00000004
msgData = {AABBCCDDEE}

[transmit5]
destination = 3
period = 100
delay = 80
extendedID = false
msgID = 00000005
msgData = {AABBCCDDEEFF}
```

Five periodic transmit messages are each set with a period of 100 ms. The messages are *delayed* in time by 20 ms each. Below timeline shows the IDs of the transmit messages given the configuration above:



Real-time-clock (RTC) configuration:

```
[RTC]                          ; Real Time Clock
epochTime = 0                  ; Epoch time (0 to keep current time)
adjustment = 0                 ; Adjustment in sec to current time (max +-129600 sec)
```

The [RTC] section configures the real-time-clock of the device. The real-time-clock enables the `CLX000` to add wall-time timestamps to log files, such that the absolute date and time of a message can be determined from the log file. The section contains the following entries:

- `epochTime`: The field is used to set the time in case time stored in the `CLX000` is lost. The field uses the epoch time format. When set to 0, the `CLX000` will keep the currently stored time.
- `adjustment`: The field is used to adjust the device time. The adjustment is in seconds and can be in range ±129600s (±36h)

When the RTC is available on the hardware, the device will be able to maintain the wall time even when it is not connected to an external power source. If the time maintained by the device has to be adjusted (e.g. to local time zones), this can be done using the `adjustment` configuration entry. In case the time is completely lost (e.g. due to the replacement of the internal battery) or if it cannot be sufficiently adjusted using the range of the `adjustment` entry, the time can be set using the `epochTime` entry. When `epochTime` is set to a value different from zero, the value will be loaded into the time stored by the device. When the time has been set, the entry should be reset to 0 to avoid the time being set each time the device is powered on. Several tools exist which can be used to convert to the epoch time format.

! If the device time needs only minor adjustments, use the `adjustment` field and leave `epochTime` to 0.

Wireless connectivity configuration:

The [WIFI] section configures the wireless connectivity of the device. Some fields have multiple uses depending on the setting of the `appMode` field.

```
[WIFI]
appName = id0001               ; Application name (1-15 characters)
appMode = 4                    ; WIFI mode (4 = Access Point, 5 = Station)
appSSID = id0001               ; WIFI SSID (1-32 characters)
appNetworkKey = 12345678       ; WIFI password (8-32 characters)
httpdMode = 0                  ; Use access authentication (0 = no, 1 = yes)
httpdUser = user               ; Access authentication user name (0-32 characters)
httpdPass = 12345678           ; Access authentication password (0-32 characters)
```

- `appName`: The application network name
- `appMode`: The WiFi mode, either Access Point (4) or Station (5)
- `appSSID`: Multipurpose field depending on the value of `appMode`
  - `appMode` = 4: The WiFi SSID of the network created by the `CLX000`
  - `appMode` = 5: The WiFi SSID of an existing network the `CLX000` will attempt to connect to
- `appNetworkKey`: Multipurpose field depending on the value of `appMode`
  - `appMode` = 4: The WiFi password of the network created by the `CLX000`
  - `appMode` = 5: The WiFi password of an existing network the `CLX000` will attempt to connect to
- `httpdMode`: The security method used by the `CLX000` web server. Access authentication can be disables (0) or enabled (1)
- `httpdUser`: The user name required if access authentication is enabled
- `httpdPass`: The password required if access authentication is enabled

The `appName` becomes the device network name and can be used to access the device on a network. If `appName` is e.g. set to `id0001`, then the main web page can be access on `http://id0001/` from a web browser.

Using the access authentication feature, the web server can be user name and password protected. This is particularly useful if the device is used in station mode on a shared network.

! The device web page can be accessed using the `appName` from a web browser, e.g. `http://id0001/`

File Transport Protocol (FTP) configuration:

The [FTP] section configures the FTP push service. The push service allows the `CLX000` to automatically push closed log files to a remote FTP server.

```
[FTP]
ftpEnb = false                  ; Enable FTP upload service
serverAddr = 192.168.1.1        ; Hostname / IP of the FTP server (0-32 characters)
serverDir = /                   ; Directory on FTP server (0-32 characters)
user = user                     ; FTP user name (0-32 characters)
pwd = 12345678                  ; FTP password (0-32 characters)
```

- `ftpEnb`: Sets the enable/disable state of the FTP push service (`true` is enable, `false` is disable)
- `serverAddr`: The address of the remote FTP server. Can be either an IP-address or a hostname.
- `serverDir`: The upload directory of the FTP server ("/" is the root folder)
- `user`: The FTP login user name
- `pwd`: The FTP login user password

Refer to section 9.2 on page 15 for more information on the FTP push service.

Example 1 - Server IP-address

```
[FTP]
ftpEnb = true
serverAddr = 216.58.213.195
serverDir = /logfiles/
user = canlogger
pwd = fudesp4KeH7r
```

The FTP server is specified using an IP-address. Log files are uploaded to the directory "logfiles" located on the FTP server.

Example 2 - Server hostname

```
[FTP]
ftpEnb = true
serverAddr = somecompany.com
serverDir = /
user = canlogger
pwd = fudesp4KeH7r
```

The FTP server is specified using a hostname. Log files are uploaded the root directory of the FTP server.

12 Troubleshooting

TC.1 In case of a fault with the internal memory please make sure to allow the device to properly power-down before re-applying power (>20sec). If the power is re-applied too fast, the device may not initialise properly. If the fault persists, make sure that the internal memory card is properly inserted in the connector slot.