

Software Manual

XELA Sensor Server

Linux and Windows

v1.7.7

Table of contents

General Limitations of the Software	4
Installation	5
Upgrade	5
Major changes	6
Using the Server	7
Compatible CAN converters	7
OS compatibility	9
Prerequisites	9
Set up	10
Additional First Steps for Linux	10
Activating CAN bus	10
Automating activation	11
Esd CAN-USB/2 and general CAN devices	11
VScom USB-CAN Plus and most Serial CAN devices	11
Installing drivers	13
Windows	13
Linux	13
Application Installation	13
Windows	13
Linux	13
Getting ready	14
Application specifics	15
XELA Configurator	15
XELA Server	16
XELA Visualisation	17
XELA Logger	18
Raspberry Pi	19
Compatible HATs	19
Application Installation	19
Limitations of the Raspberry Pi version	19
Using	20
Arguments	20
How to use XELA Logger	22
Configuration file format	22
CAN types	30
esd	30

pcan	30
slcan	30
socketcan	30
sim	30
Special sensors	31
Standard and Individually calibrated sensors	31
Prerequisites	31
Setup	32
Notes	32
Data	34
How to use parameters in other software	34
Sensors with magnetic compensation	35
Prerequisites	35
Setup	35
Notes	35
Data format	36
Sensor Layout	36
Messages	37
Example usage	38
Code	38
Common errors	39
How to report issues (checklist)	41

General Limitations of the Software

- The software is constantly developing and, therefore, is provided AS IS.
- The code is compiled and tested on 1 PC with Virtual Machines and might have limitations related to different hardware and software configurations.
- There might be code-breaking changes between the releases. You might need to adjust your code or settings to make your software work again. Please check the change log for details and test from a temporary directory before installing the new version!

Installation

You can unpack Windows and Linux executables to the directory of your liking.

Note: The default xServ.ini file is set up for simulation.

Before making a new configuration, the compatibility of the apps on your system can be tested by running `./xela_server -f xServ.ini` and `./xela_viz -f xServ.ini`

ApplImage executables

Since the release of 1.7.7, the binaries are also provided as ApplImages. Those are compiled on an Ubuntu 22.04 virtual machine with settings to be compatible with both Wayland and Xorg. Users are encouraged to install the ApplImage files into the `/home/user/.local/bin` directory and make executable there. Renaming the ApplImage executables to match regular app names is optional. If the names are kept, use the correct name when executing the apps in the terminal.

Upgrade

Before replacing Windows or Linux executables, try to run them from a temporary directory to check the compatibility of the apps on the system (See [Installation](#) for more details).

Major changes

1.7.7 (build 151312)	Add support for software-managed temperature compensation Add support for the update rate of the sensor Add AppImage release to simplify the full compiling process
1.7.6 (build 137302)	Fix the issue with calibrated data not working on some sensors Add support for calibrated 4x6 sensors
1.7.6rc (build 133105) Release Candidate	Fully calibrated sensor support Optional sensor modules New model names Upgrade to Python 3.11 on Linux (and Python 3.10 on Windows)
1.7.5 (build 96513)	Added support for Ubuntu 22.04 Upgraded Python version in Windows to 3.9 with new custom ntcan library for Windows and Ubuntu 20.04 and 22.04 Bug fixes
1.7.4 (build 66608)	Fixed an issue of delays on some PCs Added detection of Version 3 controllers if used as earlier one The configuration tool will be able to determine version 3 controllers
1.7.3 (build 64309)	Fixed issues with XR1946 sensors Added allowance for grammar mistakes in the INI file Optimised the INI parser
1.7.2 (build 62302)	Fixed PCAN under Windows for some systems Added Serial CAN to Windows Optimised MBED



Warning

Version 1.7.x will require setting `ctr_ver = 3` on some newer sensors
Version 1.7.0 can use sensor arrays through .xela files - separating
sensors will not be possible when using an array

Using the Server

Compatible CAN converters

Note: The following list includes only devices we have tested on. Other devices might also work.

Converter	Compatibility	Pros	Cons
	Windows (esd) Linux (socketcan)	Sturdy design High-speed	Works very differently in Windows and requires special drivers Reliable in Linux, but a bit slower in Windows Suitable for multi-sensor environments
PEAK USB-CAN (PCAN)	Windows (pcan) Linux (pcan) *Limited support due to the structure of pcan drivers	Cheap Small size	Requires special drivers for all OSs Note: Windows has limited functionality, no automatic module reset Note: It can be challenging to set up in Linux with UEFI and Secure Boot Compiled drivers might not work on some machines
	Linux (socketcan)	Cheap Small size	Does not work on Windows Uses Serial-over-USB communication, which is slower than the standard CAN bus. Suitable for single-sensor environments
	Linux (socketcan) Windows (slcan) Raspberry Pi 4 (socketcan)	Mountable Sturdy design	It is not functional on Windows (only for quick tests) Uses Serial-over-USB communication, which is slower than the standard CAN bus. Suitable for multi-sensor environments

Converter	Compatibility	Pros	Cons
CANable (Pro)	Linux (socketcan)	Small size Micro-USB	No DB9 connector Needs “candlelight” firmware for Linux Suitable for multi-sensor environments
2-Channel CAN-BUS(FD) Shield for Raspberry Pi	Raspberry Pi 4 (socketcan)	Has 2x CAN bus	Requires compilation of kernel drivers No DB9 connector
Waveshare RS485 CAN HAT	Raspberry Pi 4 (socketcan)	Has RS485 and CAN bus	No DB9 connector

If using a CAN converter without a DB9 connector, you can buy a DB9 breakout board online

Connect the cables between the converter and DB9 breakout board as follows:



CAN_H → Pin 7
CAN_L → Pin 2
GND → Pin 3 (highly recommended)

Warning



Even though other CAN devices might work, misconnection can break the sensor.

Please always use CAN devices with the standardised DB-9 connector

OS compatibility

- Windows 8 or later (64-bit with [Visual C++ redistributable 2015-2019 x64](#) installed)
- Ubuntu 18.04, 20.04 and 22.04 (64-bit)
Other 64-bit Linux flavours, i.e. Debian with kernel 5.13 or newer
- Raspberry Pi OS 10 (32-bit) - server only, no extra tools, Raspberry Pi 4 minimum

Prerequisites

For connecting to XELA Server with user-made Python script, the following Python packages would need to be installed:

- websocket-client

For other languages, compatible libraries would be required



Warning

Please remember to give sensors sufficient power. An external USB power supply is recommended.

There are 2 USB cables for sensors. Both MUST be connected

Visualisation can be very CPU-heavy; please use a good PC with a dedicated GPU or powerful CPU (i7+/Ryzen 7+) to run it, or consider visualisation on another computer.

Set up

Additional First Steps for Linux

Activating CAN bus

Linux and ROS only (Windows requires only drivers. Windows users can skip this part)

If you haven't activated CAN bus yet, please run the following commands:

Note: Make sure you have can-utils installed first

- **Esd or any other CAN device:**

```
user@localhost:~$ sudo ip link set up can0 type can bitrate 1000000
```

Explanation: First, we will set up the CAN network on can0 with a bitrate of 1Mbit/s, and then we pull it up

- **Any serial CAN (slcan) device:**

```
user@localhost:~$ sudo slcand -o -s8 -t hw -S 3000000 /dev/ttyUSB0
```

```
user@localhost:~$ sudo ifconfig slcan0 up
```

Explanation: First, we set up a SLCAN device running on /dev/ttyUSB0 with a hardware bitrate of 3Mbit/s and CAN speed of 1Mbit/s (-s8), then we will pull it up

NOTE: Replace can0/slcan0 with the corresponding CAN bus name and ttyUSB0 with the correct device name

NOTE: On some Ubuntu 22.04 installations, the slcand has been upgraded and it will identify the network as can0 instead of slcan0

Automating activation

Linux only

Note: The following examples might not work with your configuration; only use the instructions as a reference.

You can download our pre-made scripts from github.com/mcsix/can-device-rules

Esd CAN-USB/2 and general CAN devices

Step 1.

Open/make new network configuration

```
sudo nano /etc/systemd/network/80-can.network
```

Step 2.

Enter the following into the file

```
[Match]
Name=can*

[CAN]
BitRate=1M
RestartSec=100ms
```

Step 3.

Restart network manager

```
sudo systemctl restart systemd-networkd
```

VScom USB-CAN Plus and most Serial CAN devices

NOTE: Updated as udev rules have changed over time

Step 1.

Make a new udev rule for your device type (slcan in this example) with

```
sudo nano /etc/udev/rules.d/90-slcan.rules
```

The use of a higher rule number is recommended in case of errors

Add the following content (note: ensure the command is a single line when you copy-paste):

```
#slcan autoloader

SUBSYSTEM=="tty", ATTRS{idVendor}=="1d6b", ATTRS{idProduct}=="0002",
SYMLINK+="ttyUSB%n", GROUP="dialout", MODE="0777", TAG+="systemd",
ENV{SYSTEMD_WANTS}="slcan@%n.service"
```

Ensure you remove the line breaks (each ACTION command must be in its own line).

Step 2.

Set up service:

```
sudo nano /etc/systemd/system/slcan@.service
```

```
[Unit]
Description=slcan interface
After=dev-ttyUSB%i.device
BindsTo=dev-ttyUSB%i.device
```

```
[Service]
ExecStart=/usr/local/bin/slcan_add.sh %l
Type=forking
```

Step 3.

Make the script to run activation:

```
sudo nano /usr/local/bin/slcan_add.sh
```

```
#!/bin/sh

sleep 1
slcand -o -c -f -s8 -t hw -S 3000000 /dev/ttyUSB$1 slcan$1
sleep 1
ifconfig slcan$1 up
```

NOTE: On some upgraded machines, the CAN bus identifies as can0 instead of slcan0

Step 4.

Make both scripts executable with

```
sudo chmod +x /usr/local/bin/slcan_add.sh
```

Step 5.

Reload rules with

```
sudo udevadm control --reload-rules
```

Step 6.

Reload systemd config with

```
sudo systemctl daemon-reload
```

Step 7.

Test and adjust whenever needed

Installing drivers

Windows

Please install the drivers for your CAN-USB converter by downloading them from the manufacturer's website.

Linux

Depending on the distribution, there might be a need for PCAN drivers when using PEAK CAN-USB devices. It might also be necessary to build them with DKMS. To do that, run `sudo make -C driver dkms_install` before `sudo make install`.

Please download them from <https://www.peak-system.com/fileadmin/media/linux/index.htm>

If your system uses Secure Boot, you might need to sign the kernel modules. A good article about this is available at

<https://superuser.com/questions/1438279/how-to-sign-a-kernel-module-ubuntu-18-04>.

Instead of `vboxdrv`, you have to use `pcan`.

PCAN will also require the PCANBasic API to be installed on your system.

Application Installation

Windows

Unpack the downloaded files to your preferred folder

Keep all configuration files in the same folder

Important: Make sure you have [Visual C++ redistributable 2015-2019 x64](https://aka.ms/vs/16/release/vc_redist.x64.exe) installed from https://aka.ms/vs/16/release/vc_redist.x64.exe

Linux

Use the installer for your Ubuntu version or unpack the downloaded zip file to your preferred directory (for the latter, we would recommend adding that directory to PATH, especially if you plan to use ROS)

Make `/etc/xela` directory and give it 777 permissions

Keep all configuration files in `/etc/xela` directory

Getting ready

1. Make sure all cables are connected. In the case of CAN devices, there are 2 USB cables, and both MUST be connected, as the one on the sensor side of the CAN-USB converter is for power

2. Unpack the files to a suitable location

In the case of the ROS package, to your catkin workspace folder

3. Open your terminal window (PowerShell or Command Prompt on Windows) and navigate to the correct directory where the files are

4. Run the configuration (first use or when the configuration has changed) or make it manually

NOTE: If XELA Robotics made you a configuration file, please honour that over this application

NOTE: If using calibrated sensors, you need to use the serial number there instead. You may still use it, but a manual change in the xServ.ini file will be required

NOTE: Only to be used with standard 4x4 and 4x6 sensors; otherwise, please make a configuration file manually or consult with XELA Robotics

NOTE: If using a CAN adaptor other than the esd CAN-USB/2, you will need to specify it with the -c flag (i.e. `./xela_conf -c s1can0`)

Windows PowerShell: [./xela_conf.exe](#)

Windows cmd: [xela_conf.exe](#)

Linux: [./xela_conf](#)

5. Run the server (mandatory)

The server will send the data via Websockets. See the arguments section to see all configuration options.

Windows PowerShell: [./xela_server.exe](#)

Windows cmd: [xela_server.exe](#)

Linux: [./xela_server](#)

6. Run additional tools

- Visualizer for showing the sensor status on the screen as dots (size and location based on data)

Windows PowerShell: [./xela_viz.exe](#)

Windows cmd: [xela_viz.exe](#)

Linux: [./xela_viz](#)

- Logger to log sensor data in raw format with UNIX timestamps into the LOG folder as CSV files (all sensors will have a separate filename in the format “sens-[sensor number].csv”. Please see the arguments section to see more detailed information about using it

Windows PowerShell: [./xela_log.exe](#)

Windows cmd: [xela_log.exe](#)

Linux: [./xela_log](#)

Application specifics

XELA Configurator

xela_conf

```
[XELA]          We found 2 sensors

Sensor 1:
    Model: uSPa44 (XR2244)
    ID: 1
    Size: 4x4
    Starting SDA: 0

Sensor 2:
    Model: uSPa46 (XR1946)
    ID: 11
    Size: 4x6
    Starting SDA: 0

If all sensor data is correct, please enter y and press Enter to save the configuration. Save: █
```

XELA Configurator (Sensor AutoConfig) is a tool to set up the xServ.ini automatically. It expects the esd CAN-USB/2 device to be used if no arguments are given. When executing, the user must specify the device type and channel in the terminal for other devices.

NOTE: Only for uSPa 44 and uSPa 46 sensors

Windows examples:

./XELA_Conf.exe -d esd -c 0	Default, esd CAN-USB/2 device on Net Bus 0
./XELA_Conf.exe -d slcan -c COM3	VScom or other serial-can devices that show up on COM3 Note: Serial connection can be buggy
./XELA_Conf.exe -d pcan -c PCAN_USBBUS1	PEAK CAN-USB devices, bus/channel names can differ

Linux examples:

./xela_conf -d socketcan -c can0	Default, esd CAN-USB/2 device on can0
./xela_conf -d socketcan -c slcan0	VScom or other serial-can devices that are set up on slcan0
./xela_conf -d pcan -c PCAN_USBBUS1	PEAK CAN-USB devices, bus/channel names can differ

XELA Server

xela_server

```
[SENSORS]      Sensor 1
    ID: 11
    Size: 4x4
    Model: uSPa44
    File: __internal__/XR2244.xela
    Starting SDA: 0
    Rotation: 0°
    Description: Standard 4x4 sensor
    Modules for uSPa44:
        Module 1: Raw Data of Magnetic Compensation (rawdata) L-mode Status: OFF (default)
        Module 2: Magnetic Compensation (magcomp) L-mode Status: OFF (default)
        Module 3: Force Calibration (calibration) C-mode Status: OFF (default)
        Module 4: Clustering (clustering) C-mode Status: OFF (default)

    Layout:
        [03b0] [02b0] [01b0] [00b0]
        [13b0] [12b0] [11b0] [10b0]
        [23b0] [22b0] [21b0] [20b0]
        [33b0] [32b0] [31b0] [30b0]
```

XELA Server is the main tool connecting the sensors to other software.

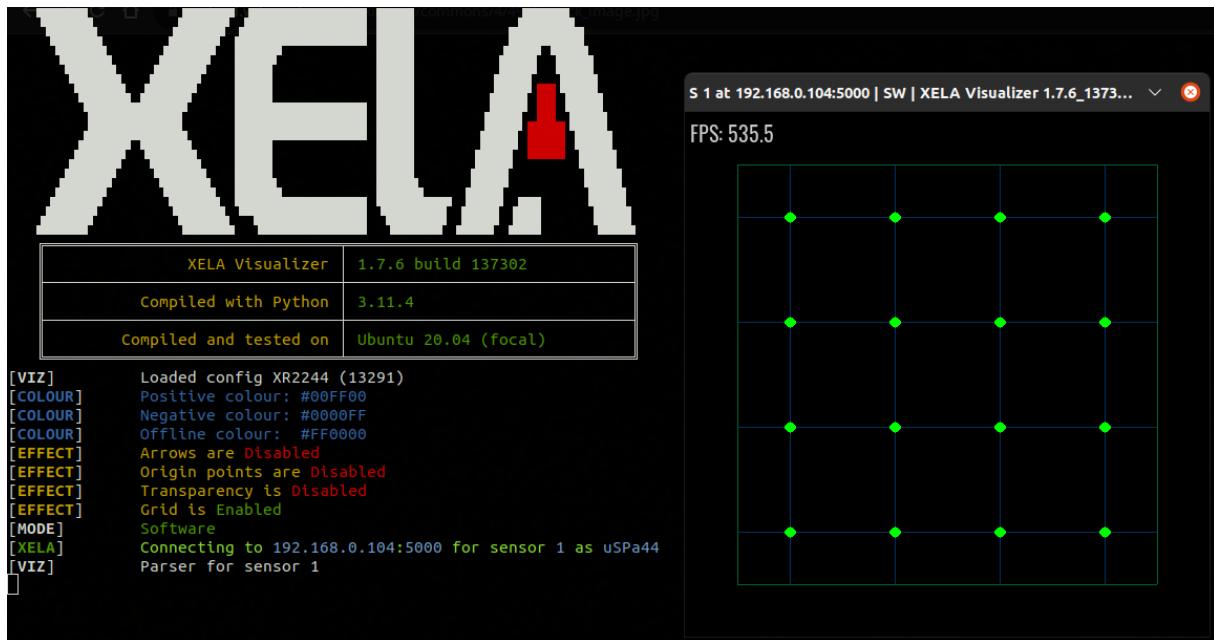
It will require an xServ.ini file to be present or an alternative specified with arguments.

Some of the important arguments:

-f filename	To specify the alternative for xServ.ini
--ip xxx.xxx.xxx.xxx	To specify the device IP in case the device has several network devices installed
--port x	To specify a port other than 5000 enables the use of several server apps on the same computer. Each can have its own CAN buses

XELA Visualisation

xela_viz



XELA Visualisation is the tool to visualise the activity on the sensor.

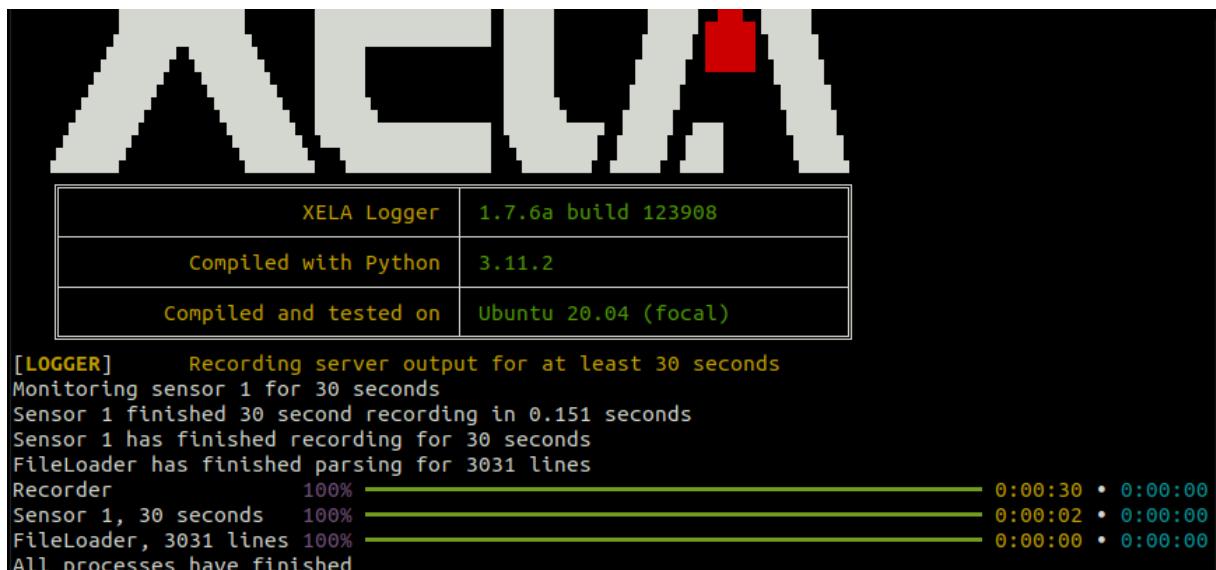
It requires both the Server to be running and the xServ.ini to exist.

Some important arguments:

-f filename	To specify the alternative for xServ.ini
--ip xxx.xxx.xxx.xxx	To specify the IP of the server if the computer has several network devices or the Server is running on a different computer
--port x	To specify the port of the Server or middle-ware if not 5000
--viz x	To specify the sensor to visualise when more than one is available on the Server, and you are not trying to visualise the first one

XELA Logger

xela_log



XELA Logger is the tool for logging sensor readings into the CSV files.

It requires a detailed argument to know which sensors to read and how long. Details are available under [Using -> How to use XELA Logger](#)

Some of the arguments:

--ip xxx.xxx.xxx.xxx	To specify the IP of the server if the computer has several network devices or the Server is running on a different computer
--port x	To specify the port of the Server or middle-ware if not 5000
-s config	Config for the data to be recorded

Note: Since version 1.7.6, XELA Logger logs all messages into a text file and later parses it to make the CSV files; therefore, accessing the data during the recording process will not be possible.

Raspberry Pi

Note: Will work only on Raspberry Pi 4B and is tested on 4 Gb RAM version running Raspian 10 (Raspberry Pi OS)

Note: Will only have major releases and, therefore, might be a few versions behind the main applications

Compatible HATs

	Seeed Studio 2-Channel CAN-BUS(FD) Shield for Raspberry Pi Pro: 2 CAN channels
	WaveShare RS485 CAN HAT

Application Installation

Applications can be installed in any suitable directory. The /etc/xela directory should be made with 777 permissions, and your xServ.ini file should be placed there.

Limitations of the Raspberry Pi version

The Raspberry Pi version only has the server module and will require another computer to be present for visualisation and logging. There is currently no automatic configuration tool either.

Using

Arguments

Following is the list of primary arguments that can be given to the executables:

green – used, red – not used

In Windows, it might need .exe at the end of the executable name.

-h Show all argument options for the script with a short description

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_server -h

-f *filename* The name of the configuration file (with .ini extension) to be used (full path or just name if in the same folder) (default: /etc/xela/xServ.ini (Linux) or xServ.ini (Windows))

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_server -f myconf.ini

--ip *ip* The IP address for the computer running the server (defaults to the first IP assigned to the computer)

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_viz --ip 192.168.0.107

-p *port* The port for communicating with the server (default: 5000)

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_viz -p 5007

--viz *sensor_number* The sensor number to display in visualisation (defaults to 1)

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_viz --viz 3

-s *sensor_config* The string for configuring the logging script (default: 1:100)

More info in How to use XELA Logger

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_log -s 1-3:1000

-lf *path_to_log_folder* Folder, where CSV files will be saved (default: CSV/in Windows or /etc/xela/CSV/ in Linux)

More info in [How to use XELA Logger](#)

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_log -lf MyLogs

--log *log_file* The path to log file (default: /etc/xela/LOG/app.log in Linux or LOG/app.log in Windows where app corresponds to app name)

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_server --log my_server_log.log

-d *device* The can device name (default: socketcan in Linux or esd in Windows)

Usage: [xela_conf, xela_log, xela_server, xela_viz]

Example: ./xela_conf -d pcan

-c *channel* The can device channel (default: *can0* in Linux or *0* in Windows)

Usage: [*xela_conf*, *xela_log*, *xela_server*, *xela_viz*]

Example: *./xela_conf -c PCAN_USBBUS1*

How to use XELA Logger

Entries must be using one of the following formats:

Data	Meaning
1	Sensor* 1 will be read (default 100 broadcast cycles*)
1,2,3	Sensors 1, 2 and 3 will be read (default: 100 broadcast cycles)
1-3	Sensors 1,2 and 3 will be read (default: 100 broadcast cycles) (shortcut)
1:150	Sensor 1 will be read 150 broadcast cycles
1:150,2	Sensor 1 will be read 150 broadcast cycles, and sensor 2 for default 100
1-3:150	Sensors 1, 2 and 3 will all be read for 150 broadcast cycles
1-5,2:150	Sensor 2 will be read 150 broadcast cycles, and sensors 1, 3, 4 and 5 for default 100
1:60t	Sensor 1 will be read for 60 seconds
1-3:30t	Sensors 1, 2 and 3 will all be read for 30 seconds
1-5,2:30t	Sensor 2 will be read for 30 seconds, and sensors 1, 3, 4 and 5 for default 100 broadcast cycles
1-5:200,3:30t	Sensor 3 will be read for 30 seconds, and sensors 1, 2, 3 and 5 will all be read for 200 broadcast cycles

Example:

A	B		C	D	E	F	G
1	time	1X	1Y	1Z	2X	2Y	2Z
2	1587029110.17588	16858	18758	35645	15128	15545	31502
3	1587029110.17615	16858	18758	35645	15128	15545	31502
4	1587029110.27562	19188	16018	35189	17276	15273	32799
5	1587029110.2757	19188	16018	35189	17276	15273	32799
6	1587029110.37499	13737	14693	34553	18699	16100	32949
7	1587029110.37511	13737	14693	34553	18699	16100	32949

Note

*When defining an argument for the -s flag, ensure that there are NO spaces in it (after a comma or a colon)

*If the sensor is defined multiple times, the latest entry counts

*Sensors can be from 1 to 64 (to 16 before 1.7.1)

*Time is recorded as a **Unix timestamp** (on all OSs).

The **Unix epoch** (or **Unix time** or **POSIX time** or **Unix timestamp**) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). The epoch is Unix time 0 (midnight 1/1/1970), but 'epoch' is often used as a synonym for Unix time. Some systems store epoch dates as a signed 32-bit integer, which might cause problems on January 19, 2038 (known as the Year 2038 problem or Y2038).

*Broadcast cycles depend on the computer configuration and are defined as periodic update cycle

Configuration file format

If yourINI needs a custom format, follow these specifics.

Sections in the config file:

[CAN] – defines settings for CAN bus (will be in [server] from 1.8.0 onwards)

[server] – defines settings for the server

[viz] – defines settings for the visualisation

[sensor] – defines primary sensor settings and the number of total boards

[sensorN] – (N is an integer at least 1, sequential) – defines Nth sensors settings; if different from defaults, you can define **[sensor1]** if it will be different from default settings

Alternative names for the element are in grey.

Section/element	notes/description
[CAN]	Note: Will be deprecated, and all elements will be moved to [server] in 1.8.0
bustype = socketcan busmode type	Bus types based on the driver: Linux: socketcan, pcan Windows: esd, pcan, slcan Testing: sim
channel = can0 can	Channel used for CAN communication, examples: pcan: PCAN_USBBUS1 esd: 0 socketcan: can0 / slcan0 With multi-CAN, the channels must be separated by commas (no spaces). I.e. <i>channel = can0,can1</i>

Format details continue on the next page

Format details continued

Section/element	notes/description
[server]	Bustype and channel are supported here since 1.7.6; they will overwrite details given in [CAN]
bustype = socketcan busmode type	Bus types based on the driver: Linux: socketcan , pcan Windows: esd , pcan , sican Testing: sim
broadcast = websocket	Broadcast defines the connection method It will have an effect from 1.8.0 onwards. WebSocket is the supported one in 1.7.x
refreshrate = 100	Set periodic send rate per second for the server The default is up to 100 Hz, but it can be limited. Must be an integer between 1 and 1000 (do not go over 100 unless you have a monster computer as it can cause delays and errors)
mode = 0	(experimental) Broadcast mode: 0 - periodic only (<i>default</i>) 1 - periodic and on CAN message receive 2 - on CAN message receive only Note: modes 1 and 2 can be very taxing on your network and CPU
single = on	(experimental) Defines if the “on CAN message receive” broadcast contains only taxel update info or the entire message. Modes 1 and 2 only on - message will only include sensor number, taxel number and new data (X, Y and Z) off - server will generate an entire message of all sensors and taxels like with the periodic message

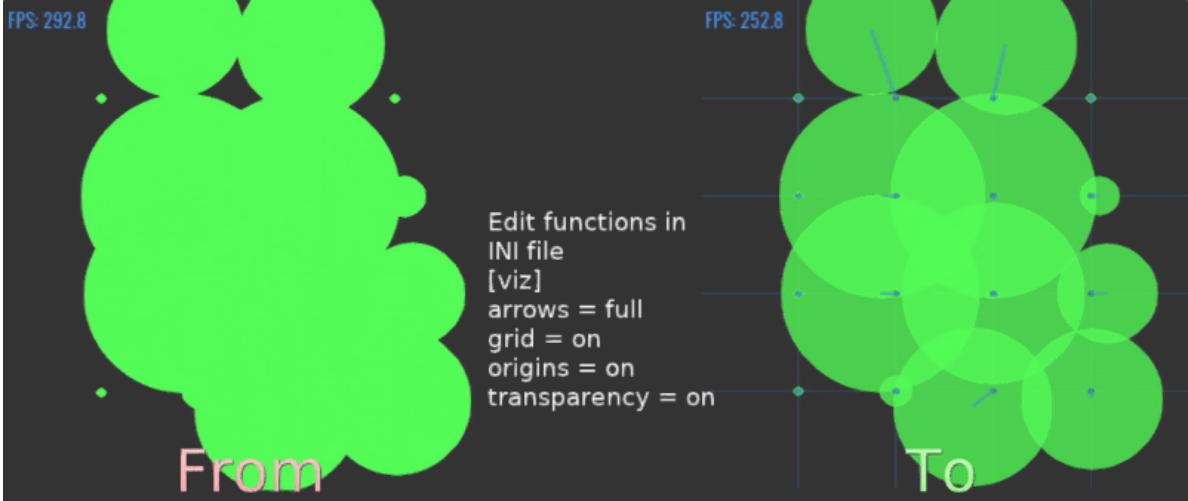
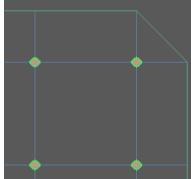
Format details continue on the next page

Format details continued

Section/element	notes/description
[debug]	Extra features for debugging or understanding the operation of the software
<p>layout = minimal server_print sens_print print</p>	<p>full – show all sensor config visually in the terminal with taxel arrangement (<i>default since 1.7.6</i>)</p> <pre>[SENSORS] Sensor 1 ID: 1 Size: 4x6 Model: XR1946X File: /home/mcsix/Projects/xela-server/sensors/XR1946X.xela Starting SDA: 0 Rotation: 0° Description: Standard 4x6 sensor Modules for XR1946X: Module 1: Before Magnetic Compensation (rawdata) L-mode Status: ON (default: ON) Module 2: Magnetic Compensation (magcomp) L-mode Status: OFF (default: ON) Module 3: Clustering (clustering) C-mode Status: OFF (default) Layout: 001 011 021 031 041 051 101 111 121 131 141 151 201 211 221 231 241 251 301 311 321 331 341 351</pre> <p>File: Shows which file the data was loaded from. In case of multiple files with the same name, it helps to troubleshoot. This is necessary when working with several definition file versions.</p> <p>Description: Shows info about the sensor. With some fully calibrated sensors, it will also state ownership</p> <p>Modules: Shows available modules and their statuses. To change the status, add the name in brackets to the [sensorN] section of the INI file and assign ON or OFF as a value. I.e. <i>magcomp = on</i></p> <p>Layout: Shows info about taxel arrangement, including HEX ID and CAN channel sequence (with Multi-CAN, it helps to determine to which CAN the specified sensor belongs to)</p> <p>minimal – only show specifics about sensors (ID, Size and Type)</p> <pre>Sensor 1 (ID: 11, Size: 4x6, Type: XR1946, sSDA: 0, rotation: 0°) Sensor 2 (ID: 0, Size: 4x6, Type: XR1946, sSDA: 0, rotation: 0°) Sensor 3 (ID: 1, Size: 4x6, Type: XR1946, sSDA: 0, rotation: 0°) Sensor 4 (ID: 1, Size: 4x6, Type: XR1946, sSDA: 2, rotation: 0°) Sensor 5 (ID: 2, Size: 4x6, Type: XR1946, sSDA: 0, rotation: 0°)</pre> <p>none – do not show any info about sensors in the terminal</p>
visual	<i>(experimental)</i> on - show detailed debug info in the visualiser off - no detailed info shown (<i>default</i>)

Format details continue on the next page

Format details continued

Section/element	notes/description
[viz]	Specific settings only for XELA Visualizer
 <p>From To</p> <p>FPS: 292.8 FPS: 252.8</p> <p>Edit functions in INI file [viz] arrows = full grid = on origins = on transparency = on</p>	
max_offset = 100	Defines how far the “bubbles” will go in visualisation 100 is the gap between 2 taxels (bubbles) in idle
max_size = 100	Defines the maximum size of the “bubbles” in the visualisation 100 is the gap between 2 taxels (bubbles) in idle
arrows = full	Defines if arrows are shown: full - show from the origin to the centre of the new position half - show from the origin up to halfway to the centre of the new position none - hide all lines between origin and new position
grid = on	Defines if the grid of the sensor is shown: on - shown off - hidden <i>Some sensors will have a custom grid that helps to understand them better</i> 
origins = on	Defines if origin points are shown: on - shown off - hidden
transparency = on	Defines if taxels have transparency: on - shown off - hidden Note: it can be taxing on the CPU without a dedicated GPU

Format details continue on the next page

Format details (for [viz]) continued

Section/element	notes/description
fps = on	<p>Defines if the visualization window shows FPS counter for its re-draw rate</p> <p>In HW mode it will try to match the screen refresh rate In SW mode it will show total maximum of re-draw rate Default mode is HW since 1.7.7</p> <p>Supported values for this setting:</p> <p>on - shown off - hidden (default)</p> <p>Note: can also be defined under [sensor] and [sensorN] sections Note: should only be enabled if visualization performance is low and running in software mode has no change</p>
ups = on	<p>Defines if the visualization window shows UPS counter for sensor update rate</p> <p>It will only be shown if the server broadcasts the information</p> <p>Supported values for this setting:</p> <p>on - shown off - hidden (default)</p> <p>Note: can also be defined under [sensor] and [sensorN] sections</p>
temp_show = on	<p>Defines if temperature info is shown</p> <p>on - shown (default) off - hidden</p> <p>Note: will only be shown if supported by sensor and server</p>

Format details continue on the next page

Format details continued

Section/element	notes/description
[sensor]	Details about the default sensor (unless overridden in [sensorN])
num_of_sensor = 2 num_brd sensors num_of_sensor	Defines the number of sensors to be broadcast by the server. If using multiple small (1x1, 2x1, 2x2) sensors on a single controller, the sensors different from the first must specify it, i.e. channel=2
version = 3 ctrl_ver controller_ver controller_version ctr_ver	Defines the controller version By default (if not specified), it will be 3. Version 1 was supplied in 2018 - early 2019 only. Version 2 and 3 have different X, Y, and Z resolutions (15/16 bit HW) <i>Latest version of the software can automatically adjust when necessary</i>
model = XR1944 mdl	Defines the sensor model (or serial number) The default ones are: XR1844 - old 4 channel 16 taxel sensor (no longer sold) XR1944 - new 2 channel 16 taxel sensor (no longer sold) uSPa11 - 1x1 sensor (formerly XR1911) uSPa21 - 2x1 sensor (formerly XR1921) uSPa22 - 2x2 sensor (formerly XR1922) uSPa46 - 4x6 sensor (formerly XR1946) uSPa44 - 4x4 sensor (formerly XR2144 and XR2244 with additional features) Individually calibrated sensors - will have the serial number here
id = 6 ctr_id controller_id ctrl_id	Defines the controller ID It must be written in decimal as numbers from 0 to 15.
channel = 0 chan sda	Defines the starting channel (SDA) for the sensor
taxel = 0 txl	Defines the taxel ID offset It must be written in decimal as numbers from 0 to 15. Used with some uSPa 11/21/22 sensors
format = formatted data_format	Defines if formatting on the server is enabled or the raw info is broadcast (setting it to raw will disable most functions) modified - (default), all axes will be set by the server to be the same across all sensors raw - all axis, as shown in the hardware manual, can become very confusing when using several different sensors

Format details continue on the next page

Format details continued, section [sensor]

rotation = 0 rot rotation	Defines how much the sensor will be rotated for adjustment 0 - no rotation 1 - 90 degrees clockwise 2 - 180 degrees 3 - 90 degrees anti-clockwise NOTE: Some modules will conflict with rotation
<u>module_name</u> = off	Enable or disable custom modules supported by the sensor model. Most modules are off by default. off - force the module to be disabled on - force the module to be enabled Example modules: magcomp - magnetic compensation clustering - calculated touch clusters calibration - force calibration Warning: enabling modules not supported by your hardware can cause invalid data.

Format details continue on the next page

Format details continued

Section/element	notes/description
[sensor <i>N</i>]	Contains only details that override the default sensor; check alternative naming from [sensor]
version = 3	Defines the controller version
model = uSPa44	Defines the sensor model
id = 6	Defines the controller ID
taxel = 0	Defines the taxel offset
channel = 0	Defines the starting channel (SDA) for the sensor
format = formatted	Defines if formatting on the server is enabled or the raw info is broadcast (setting it to raw will disable most functions)
rotation = 0	Defines how much the sensor will be rotated for adjustment 0 - no rotation 1 - 90 degrees clockwise 2 - 180 degrees 3 - 90 degrees anti-clockwise
<i>module</i> = off	Enable or disable custom modules supported by the sensor model. Most modules are off by default. off - force the module to be disabled on - force the module to be enabled Example modules: magcomp - magnetic compensation clustering - calculated touch clusters calibration - force calibration Warning: enabling modules not supported by your hardware can cause invalid data.

CAN types

esd

The default for esd CAN-USB/2, using *ntcan* drivers provided by esd as Windows lacks support for standardised CAN. The user is required to install the latest drivers and SDK found on esd.eu

Channels are numeric and can be defined in Device Manager to support several devices on the same PC

pcan

Most PEAK devices use a non-standardized communication protocol between the PC and CAN converter. It works both in Windows and Linux.

Channels are named based on the device type and sequence, i.e. PCAN_USBBUS1

slcan

SerialCAN for Windows, covering devices like VScom's USB-CAN Plus.

Channels are COM ports shown in the Device Manager

socketcan

Default for Linux, covering most CAN and SerialCAN devices. Some PEAK devices can also work under this setting.

Channels are actual *can* and *slcan* buses in the format of canN or slcanN

sim

Simulated CAN device, trying to match the maximum CAN messages expected from any sensors.

If the channel is “sequential”, the output will have a sequence based on the computer time, starting every 15 seconds. It affects only X and Y values; Z is random. The sequence is as follows:

1. 4 seconds of tiny changes (wiggly mode)
2. 2-second intervals of 4 diagonal directions each (centre-to-corner movement)
3. 3 seconds of complete randomness

With any other value on the channel parameter, it will produce random values for X, Y and Z

Special sensors

Standard and Individually calibrated sensors

Prerequisites

- Software version 1.7.5 or newer
- Fully calibrated sensors with serial numbers or sensors supporting the calibration
- Calibration and definition files

Sensor compatibility:

Sensor model	Standard calibration* ¹	Individual calibration (extra file needed)
uSPa44	✓	✓
uSPa46	✗	✓
uSPa22	✗	✓

File compatibility:

Software version	.xeladef	.xdef	.xela	.xcal
1.7.5	✓	✗ * ²	✗	✓
1.7.6+	✓	✓	✓	✓

Generic parameters for uSPa44 are included with the software 1.7.6+

.xcal file is not required if using 1.7.6+ with .xela file unless the user needs to adjust the parameters themselves

¹ For sensors getting standard calibration after software release, can be manually added with an additional file. 1.7.6 release includes standard calibration only for uSPa44

² Some .xdef files might work in 1.7.5 release, but are not recommended

Setup

1. Copy the provided files (.xela, .xcal) to the following directory:
 - a. /etc/xela/sensors directory (Linux)
 - b. the same directory you have the application in (Windows)
2. Edit the INI file to contain the following info:
 - a. model = S/N (only for fully calibrated sensors), otherwise model = model_name
Examples: model = XC166EC61 for fully calibrated sensor and model = uSPa44 for standard calibration on uSPa44
 - b. calibration = on (only required for standard calibration)

Notes

Comparing Sensor Calibration: Generic vs. Individual

In this software manual, we explore two calibration variants of the uSPa44 (XR2244) sensor: Generic Calibration and Individual Calibration. They both have the same purpose but with some significant differences.



Generic Calibration:

Generic calibration is a standardised approach calculated over data collected from multiple sensors of the same type. These parameters are designed to provide a baseline level of performance suitable for a range of applications.

Key points to note about Generic Calibration:

Simplicity: Generic calibration ensures a simple activation method by writing a simple line in the INI file compared to changing every model name to a serial and managing extra files.

Immutable Parameters: Generic calibration parameters are set during manufacturing and remain fixed. They are not user-adjustable or accessible from the software interface. This means they are invisible to the user and cannot be modified.

Individual Calibration:

On the other hand, individual calibration is a highly tailored approach explicitly performed on the uSPa44 (XR2244) sensor in question. These parameters are fine-tuned to the unique characteristics of this particular sensor.

Here's what sets Individual Calibration apart:

Precision: Individual calibration offers superior precision and accuracy. It optimises the sensor's performance to its full potential, ensuring the most precise measurements.

Customizability: One of the critical advantages of Individual Calibration is its flexibility. Users have the ability to modify and fine-tune parameters to suit the specific requirements of their application. This means you can adapt the sensor to various tasks or conditions, providing a higher level of control.

External Access: Individual calibration parameters are accessible from the software interface, allowing users to make real-time adjustments. This added functionality empowers users to fine-tune the sensor's behaviour on the fly.

Forces Up to 5N:

It's important to note that Generic and Individual Calibration methods have been calculated with forces up to 5N for each sensing point. This ensures your sensor maintains its precision and reliability within this specified force range.

With all calibration, you need to edit the xServ.ini file manually.

For generic parameters (like with uSPa44 sensors), you need to add `calibration = on` under the `[sensor]` or `[sensorN]` section to enable the output

For individual parameters, you must change the model to the serial number. I.e. instead of `model = uSPa44` you will write `model = XC166EC61`

Data

All calibrated data is available as part of the sensor message. Please see an example below with the information highlighted:

```
{'message': 786, 'time': 1619157045.6195483, 'sensors': 2, '1': {'data': 'FA00,...,FC00', 'sensor': '1', 'taxels': 4, 'model': 'uSPa22', 'calibrated': null}, '2': {'data': 'FB01,...,FC01', 'sensor': 2, 'taxels': 16, 'model': 'uSPa44', 'calibrated': [0.059,0.021,0.15,...]}}
```

For more info about data format, please see paragraph [Messages](#).

Note: The forces are in Newtons.

How to use parameters in other software

You will get the parameters in the XCAL file for individually calibrated sensors. They are in a table, where each row is for 1 taxel.

Note: Their sequence is based on the sequence provided by the Server software, so using them directly from the CAN bus would require making sure the correct taxel IDs first. Refer to the table at the start of the Server to see ID allocations.

You can use the following formula for calculating the force. In the XCAL file, the parameters are given in a list of 27 elements, 9 for each axis (X, Y and Z).

Read:

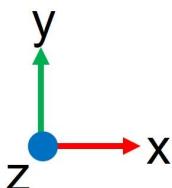
For x, p = params[0...8]

For y, p = params[9...17]

For z, p = params[18...26]

k stands for X, Y or Z

$$F_k = x_d * p_0 + y_d * p_1 + z_d * p_2 + x_d * y_d * p_3 + x_d * z_d * p_4 + y_d * z_d * p_5 + x_d^2 * p_6 + y_d^2 * p_7 + z_d^2 * p_8$$



The parameters, by default, refer to the orientation stated in the hardware manual.

When using the server application, the forces will be adjusted to show in the normal XY plane as it is with regular measurement values corresponding to the image shown here.

Sensors with magnetic compensation

Prerequisites

- Software version 1.7.6 or newer
- Sensors with magnetic compensation support
- (optional) sensor .xdef or .xela files (if not included in the software)

Setup

- Copy the provided files (.xela or .xdef) to the following directory:
 - /etc/xela/sensors directory (Linux)
 - the same directory you have the application in (Windows)
- Edit the INI file to contain the following info:
 - magcomp = on under the [sensor] or [sensorN] section

Notes

It can be provided as an individual model or as a module for the current sensor model. For example, the uSPa44 has a variant with magnetic compensation, and to activate it, the INI file must have `magcomp = on` under the same sensor configuration section to activate.

Note: Applying magnetic compensation for sensors without this feature can cause invalid output.

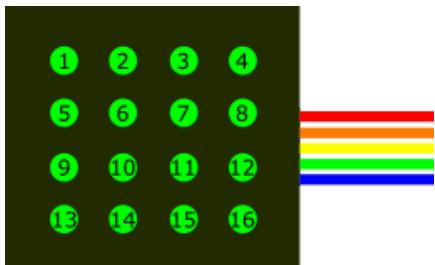
The configuration tool will recognise it as a standard sensor, and the user must change it in the INI file.

This sensor type has 2 extra settings that can be used in the INI file under the [sensor] and [sensorN] sections:

rawmode	On / off (default off)	Allows to see both compensated and raw data in the XELA Viz application Also adds raw data to the message
magcomp	On / off (default on)	Activates the compensation algorithm In the data stream, it will replace the original data Enable <code>rawmode</code> if you need the initial data

Data format

Sensor Layout



Sensor data is read from the top left towards right, line-by-line.
If 2 4x4 sensors are connected to the same controller, the server treats them as 2 different sensors (default behaviour, special configurations will be possible at request).

To determine the layout for the sensor, place it in front of you with cables extruding from the right (or connector side on the right as with XR1946) as per the image on the left side.
Note: 4x6 will have 2 extra columns, meaning the first row is 1 – 6 instead of 1 – 4.

Note: Curved sensors will have a layout in a way where it is seen as flat. Please use the server layout as a reference and remember to keep the cable to the right.

To enable layout in the server application, please set the corresponding variables under [debug] in the INI file

Note: the sensor is considered up-right if the cables exit on the right (attachment) side

Messages

As the data is provided in a JSON string, there are few ways to access it, and it has the following default format (except the welcome message sent at connection; the message will be 'Welcome' there, not int):

```
{'message': 786, 'time': 1619157045.6195483, 'sensors': 2, '1': {'data': 'FA00,...,FC00',  
'sensor': '1', 'taxels': 4, 'model': 'XR1922', 'calibrated': null}, '2': {'data': 'FB01,...,FC01',  
'sensor': 2, 'taxels': 16, 'model': 'XR2244', 'calibrated': [0.059,0.021,0.15,...]}}
```

message has the sequence number of the broadcast (to avoid outdated messages)

time has a UNIX timestamp of publishing time (can be handy when handling messages over the network and possible delays)

sensors mentions the total amount of sensors the server is handling

Each sensor will have the following format:

```
{'data':'FA00,FA00,FA00,FB00,FB00,FB00...', 'sensor': '1','taxels':16,'model':XR1922'}
```

data contains X, Y and Z data for all taxels (sensing points) in order, converted into HEX values (without leading '0x') as text

sensor has the number of the sensor in string format (important when requesting multiple sensors from the server)

taxels has the number of the taxels meant to be sent (for confirmation only)

model shows the correct sensor model as a reference (some special sensors might have the same taxel counts with different positions)

calibrated shows the calculated forces applied to the sensor (if supported by the sensor via XCAL files)

Example usage

Code

```

#!/usr/bin/env python3
import websocket
import json
from time import sleep
import threading

ip = "192.168.0.103" #your computer IP on the network
port = 5000 #the port the server is running on

lastmessage = {"message": "No message"} #default message you will overwrite when you get update
def on_message(wsapp, message):
    global lastmessage #globalise to overwrite the original
    try:
        data = json.loads(message)
    except Exception:
        pass
    else:
        try:
            if data["message"] == "Welcome":#get the Welcome Message with details, print if you like
                print(data)
            else:
                lastmessage = data
        except Exception:
            pass #ignore message as it's probably invalid
    def threader(target, args=False, **targs):
        #args is a tuple of arguments for a threaded function; other key-value pairs will be sent to Thread
        if args:
            targs["args"]=(args,)
        thr = threading.Thread(target=target, **targs)
        thr.daemon = True
        thr.start()
    def mesreader():#this is your app reading the last valid message you received
        while True:#to run forever
            try:
                if lastmessage["message"]!="No message":
                    print("I received: {}\n-----".format(str(lastmessage)))
                    sleep(0.5) #your calculations and processes here (sleep is used as simulation here)
            except KeyboardInterrupt:
                break #break on KeyboardInterrupt
            except Exception as e:
                print("Exception: {}: {}".format(type(e).__name__,e))
        try: #try to close the app once you press CTRL + C
            wsapp.close()
        except Exception:
            exit()
    threader(mesreader, name="Receiver") #start you main app
    websocket.setdefaulttimeout(1) #you should avoid increasing it.
    wsapp = websocket.WebSocketApp("ws://{}:{}".format(ip, port), on_message=on_message)#set up WebSockets
    wsapp.run_forever() #Run until the connection dies
    exit()

```

Common errors

Error	Reason
Anti-virus software sees the applications as potential malware or virus	Due to Python (the language the entire application suite is written in) being interpreted language, most AV software will be unable to differentiate between malware and regular application that uses sockets. To avoid this, we would need to rewrite the whole application suite in other languages, which is also planned for the 1.8.0 release. It will be worked on to remove false positive flagging
XELA_Conf found the sensor, and XELA_Server is working with no errors, but there is no XELA_Viz visual output (Windows)	The AV software may be interfering with the networking of the software by not letting the Visualizer connect to the Server. Turn off the AV software and try again
XELA_Conf is giving: Message not sent: OSError: Transmit operation timed out	Make sure the power USB is connected to a power source and the DSUB9 is firmly attached to the CAN-USB converter
Could not start CAN: OSError: [Errno 19] No such device	Make sure to pull up the network with one of the following commands if using Linux: <code>user@localhost:~\$ sudo ip link set up can0 type can bitrate 1000000</code> or <code>user@localhost:~\$ sudo slcand -o -s8 -t hw -S 3000000 /dev/ttyUSB0</code> <code>user@localhost:~\$ sudo ifconfig slcan0 up</code> In the case of Windows, make sure the adapter is connected and using the specified channel (esd channel might not be 0)
Program not responding to CTRL+C	There are times when the apps might not properly respond to CTRL+C. In those cases, you can try the following: Linux: Ctrl-Shift-\ or pkill -9 xela_server (or whichever function is not responding) Windows: CTRL+Pause_Break or you can simply close the app
Error connecting to CAN: IOError:[Errno 19] No such device	No CAN device was found. Make sure your CAN-USB device is connected, drivers installed for it, accessible for all users (pulled up) and set in the configuration file correctly (<u>see /etc/xela/xServ.ini or xServ.ini</u>)
Error writing config file: IOError: [Errno 2] No such file or directory: '/etc/xela/xServ.ini'	Ensure there is a /etc/xela folder (in Linux or an executable folder in Windows) and that it has 777 permissions

Unable to install Python 3.7 or newer on Ubuntu	To install different Python versions on Ubuntu, use the deadsnakes repository: <pre>sudo add-apt-repository ppa:deadsnakes/ppa</pre>
XELA_Viz fails to launch Showing ImportError	Possible reasons are that your computer software or hardware is incompatible with SDL2 libraries inside the executable. We recommend using a different computer for visualisation in such a case, preferably running Ubuntu 20.04 (similar happens when running Ubuntu 16.04 app in newer Ubuntu release. Check release with <code>uname -a</code>) A similar error on Windows means missing Visual C++ redistributable 2015-2019 x64 . Please install it, restart the computer and try again
The new CAN-USB converter is not working with the software	Please make sure the following steps: <ul style="list-style-type: none"> • CAN-USB device is supported in the OS you are using • You have selected the correct bus type in the configuration file (socketcan/esd/pcan) • You have selected the correct channel in the configuration file (can0/slcan0/COM3, etc.) • Make sure the network is pulled up (Linux only) • <code>ctrl_id</code> matches with the hardware

If you find errors not listed in this file, please send an email regarding it to info@xelarobotics.com

Do not forget to attach files from the `/etc/xela/LOG` directory in Linux or the `LOG` directory in Windows (if using the default log directory), a description of the failure, and additionally, you can add the terminal log (if log directory is empty, as a text, not image). If you have made your own code causing errors, please attach it.

Known issues

Visualization

- Sometimes fails to reconnect to the server (under investigation)

Server

- Crash if esd Net Bus ID is wrong (limitation of the current ntcan library)

Configurator

- Only supporting uSPa 44/46 sensors (will be added in the future)

How to report issues (checklist)

- Have you checked the messages provided in the terminal to see if there is a missing file or permission errors
- Have you checked if the sensors are connected to the PC and powered on
- Have you read the manual fully to see if anything is missing (i.e. server in simulation mode)
- Have you attached all log files to the email
- Have you added additional info covering what steps have been taken to resolve the error on your own
- Have you provided terminal output in text files attached to the email

Once all is checked, feel free to send the email to info@xelarobotics.com

Reminder: we might request additional details and info before being able to fix your issue