# IXXAT USB to CAN v2 compact Linux setting

The IXXAT USB-To-CAN linux driver provides support for the following devices:

- **IXXAT USB-to-CAN V2 compact** / embedded / professional / automotive / plugin / FD compact / FD professional / FD automotive / FD automotive / FD MiniPCIe / IXXAT USB-to-CAR / IXXAT CAN-IDM101
- ECI = Embedded Communication Interface
    1. This driver is used for real time OS i.e. Linux

1. Download a driver from website

https://www.hms-networks.com/support/general-downloads

  — navigate to Linux Drivers → **socketCAN driver for Linux → Download**

2. Open compressed file **socketcan-linux.gz** → uncompressed **ix_USB_can_2.0.529-REL**
3. Navigate to the driver source code Directory

```
cd /home/......./ix_usb_can_2.0.529-REL
```

- check the **Makefile** inside a folder

```
ls Makefile
```

4. Verify kernel Header Installation

- check current kernel version

```
uname -r
```

- Install new header  for running kernel

```
sudo apt install linux-header-$(uname -r)
```

- confirm a header directory exists after,

```
ls /usr/src/linux-headers-6.11.0-17-generic
# output from → uname -r → 6.11.0-17-generic  → this is current kernel version
```

5. Installing Dependencies

- Install flex :

  Flex is a tools for generating lexical analyzers (called scanners) in Linux system.

  Read input files containing descriptions of lexical patterns and corresponding C code

  Generate C source file (`lex.yy.c`) that define a routine called yylex()

  C code produce by flex can be compiled ans linked with `-lfl` library to create a executable scanner

```
sudo apt install flex
```

- Install bison:

  Bison is a tool often required for kernel builds

  It translates formal grammar specification into code that can parse input can parse input data according to specification

  User provide Bison with grammar specification (.y) extension , then this file outline syntax rule for language or data format

  When bison process grammar file, C source code is created using LALR(1) parser

  Then parser recognize valid sequence of token based on grammar and build parse tree or abstract syntax tree(**AST**) from input data.

  Often used with lexical analyzer (i.e. Flex) which break input text into tokens and then processed by Bison parser.

```
sudo apt install bison build-essential
```

5. Reconfigure the kernel Headers

- Navigate to the header directory

```
cd /usr/src/linux-header-6.11.0.17-generic
#output from → uname -r → 6.11.0-17-generic  → this is current kernel version
```

- *Reinstall kernel header (Optional)*

  *This practice is good because header file might be corrupted or incomplete when you try to install for the first time so removing once and installing again is good idea*

```
sudo apt remove linux-header-6.11.0.17-generic
#output from → uname -r → 6.11.0-17-generic  → this is current kernel version
```

- *Install kernel header (Optional)*

```
sudo apt install linux-header-6.11.0.17-generic
#output from → uname -r → 6.11.0-17-generic  → this is current kernel version
```

- prepare the headers

  Here we can see CAN is added to .config and configuration is written

```
sudo make oldconfig
```

6. preparing make file

- go to IXXAT driver location

```
cd /home/nmc/………/ix_usb_can_2.0.520-REL
```

- clean any previous build attempts

```
make clean
```

- try building driver again

```
make
```

- check build output

```
sudo make install
```

7. Install and Test

- If step 6 succeed then ,

  `modprobe` command is used to add or remove kernel modules dynamically

  Here command load `ixxat_usb` module into kernel , which allow **IXXAT USB-TO-CAN** device

```
sudo modprobe ix_usb_can
# this is exclusive for IXXAT USB-to-CAN
```

- Test device:

  No output means success; error will be explicit (message will be shown)

  `ix_usb_can` list can be visible after command below command, which means it''s loaded

```
lsmod | grep ix_usb_can
```

- Verify the device

  Plugin **IXXAT USB-to-CAN V2** compact device to your PC

  Check for **CAN interface** ( message something like can0 is seen if installed correctly)

```
ip link
```

- install `can-utils` if not installed(optional)

```
sudo apt install can-utils
```

- Bring up CAN interface ( if not already up)

```
sudo ip link set can0 up type bitrate 250000
# bitrate =  250000 is defined inside a HAS2 canCtrl.cpp
# can0, can1.. cann = 0,1,..n is a channel number here verfiy from "ip link"
```

- Test 1 :

```
candump can0
```

- Test 2:

Stop can by:

```
ip link set can0 down
```

- Test 3:

  Restart automatically after waiting for 500ms after **bus-off** condition

  **Bus-off** condition is a error state occur within a CAN controller itself

  Bus-off actually means:

  1. Error state →CAN controller transmit error counter exceed 255
  2. Communication halt: CAN no longer acknowledge frames on CAN bus
  3. Error Containment : Overloading the bus with  error message
  4. Recovery process : CAN needs to detect 128 occurrences of 11 consecutive recessive bits on the bus before it can attempts to rejoin the network

```
ip link set can0 type can restart-ms 500
```

- Test 4

setting a `can0` up

```
ip link set can0 up
```

## RUNNING SocketCAN with CAN-to-USB

1. This allow us to read a CAN bus using `can-utils` on Linux system
2. Connect **CAN-to-USB BUS** to PC and device

| command | Remark |
|---|---|
| 1. Verify CAN device | |
| `lsusb` | your can-to-usb device must be in the list |
| `ip link` | can0: <NOARP.... this must in the list<br>This means interface is detected |
| 2. Load Kernel module (optional ) (just to verify purpose only | |
| `sudo modprobe can`<br>`sudo modprobe can_raw`<br>`sudo modprobe peak_usb` | The IXXAT USB-to-CAN V2 Compact uses the `peak_usb` driver for its high-speed CAN interface |
| `sudo dmesg \| grep -i can` | Verify device in system logs<br>check word like<br>`IXXAT` and `can0` |
| 3. Configure CAN Interface | |
| `sudo ip link set can0 type can bitrate 250000` | set bitrate to 250kbps |
| `sudo ip link set can0 up` | enable interface |
| `ip -details link show can0` | check status(optional)<br><br>`state UP` should be there |
| `candump can0` | Start capturing all CAN message |

## Data analysis

1. Filtering by CAN ID

   `1A3` is a mask to filter , ID = `1AA`

```
candump can0,1A3:1AA
```

2. Add timestamped data

```
candump -t z can0
```

3. Log to a file ( for future purpose)

   this generate log file as as `candump-<date>_XXXXX.log`

```
candump -l can0
```