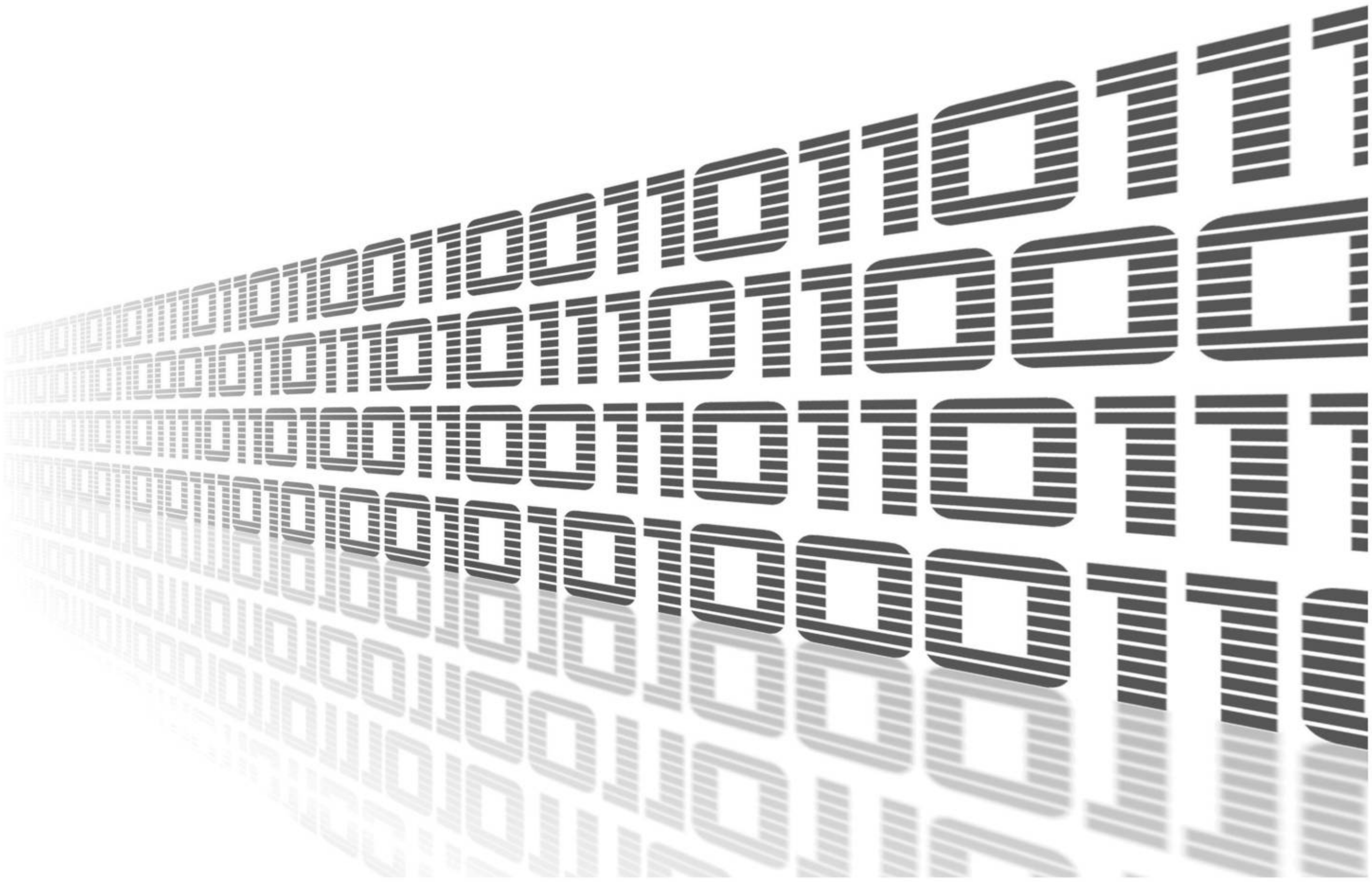
Bluetooth



Advantech Czech s.r.o., Sokolska 71, 562 04 Usti nad Orlici, Czech Republic
Document No. APP-0098-EN, revision from 11th October, 2023.

Used symbols

Danger – Information regarding user safety or potential damage to the router.

Attention – Problems that can arise in specific situations.

Information – Useful tips or information of special interest.

Example – Example of function, command or script.

Contents

List of Figures

List of Tables

1. Changelog

1.1 Bluetooth Changelog

v1.0.0 (2021-01-10)

- First release of bluetooth support.
- With Bluez 5.55.
- With D-Bus 1.12.20.

2. Introduction

This router app is not installed on *Advantech* routers by default. Uploading of this router app is described in the Configuration manual (see Chapter Related Documents).

The router app is compatible with ICR-32xx routers only.

Bluetooth is a wireless technology standard used for exchanging data between fixed and mobile devices over short distances using UHF radio waves in the industrial, scientific and medical radio bands, from 2.402 GHz to 2.480 GHz. It was originally conceived as a wireless alternative to RS-232 data cables.

There are two main variants of Bluetooth and those are classic Bluetooth and Bluetooth Low Energy. Even that they exists within same standard, they are considerably different.

Classic Bluetooth is here from the beginning. In some texts you can find Classic Bluetooth called Bluetooth BR/EDR. This variant aims to bigger data transmissions - file transfer, audio casting etc. and requires to handshake connection between devices and it usualy creates long data streams.

Bluetooth Low Energy (BLE), previously known as Wibree, is a subset of Bluetooth v4.0 and newer with an entirely new protocol stack for sending short packets, what is useful in IoT. Compared to the Bluetooth standard protocols that were introduced in Bluetooth v1.0 to v3.0 and continues in 4.0 and on, it is aimed at very low power applications powered by a coin cell for several years. In terms of lengthening the battery life of Bluetooth devices, BLE represents a significant progression. In the version 4 the reach distance of BLE was fairly short, but was significantly improved in version 5.

Bluetooth implementation in our routers have three parts:

1. Kernel Bluetooth support + drivers (from 6.2.6 firmware)
2. Bluetooth Router App containing Bluez – Linux Bluetooth stack
3. Applications – now Node-RED Bluetooth node only

The current implementation targets to Bluetooth Low Energy sensors only.

3. Web Interface

Once the installation of the module is complete, the module's GUI can be invoked by clicking the module name on the Router apps page of router's web interface.

Left part of this GUI contains menu with Information menu section with Status and Nearby Devices items and Configuration menu section. General menu section contains list of Licenses used and Return item, which switches back from the module's web page to the router's web configuration pages. The main menu of module's GUI is shown on Figure 1.

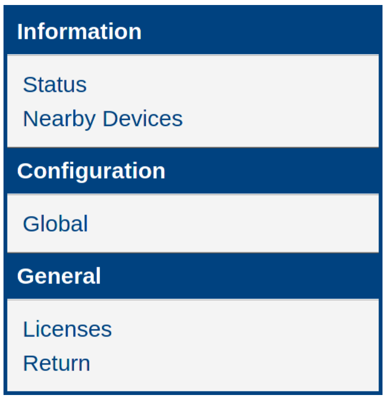
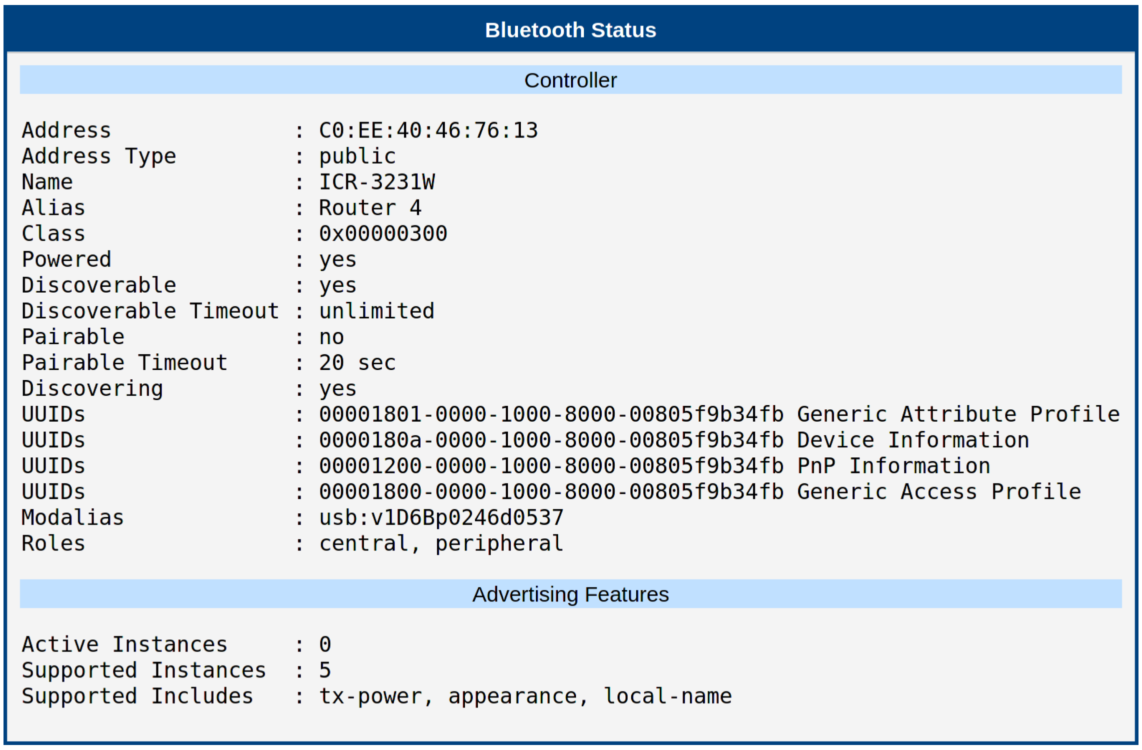


Figure 1: Menu

4. Information

4.1 Status

Actual settings of Bluetooth adapter is displayed here when Bluetooth is active. Address, data presented by the device, wether is device discovering and discoverable and what services provides.



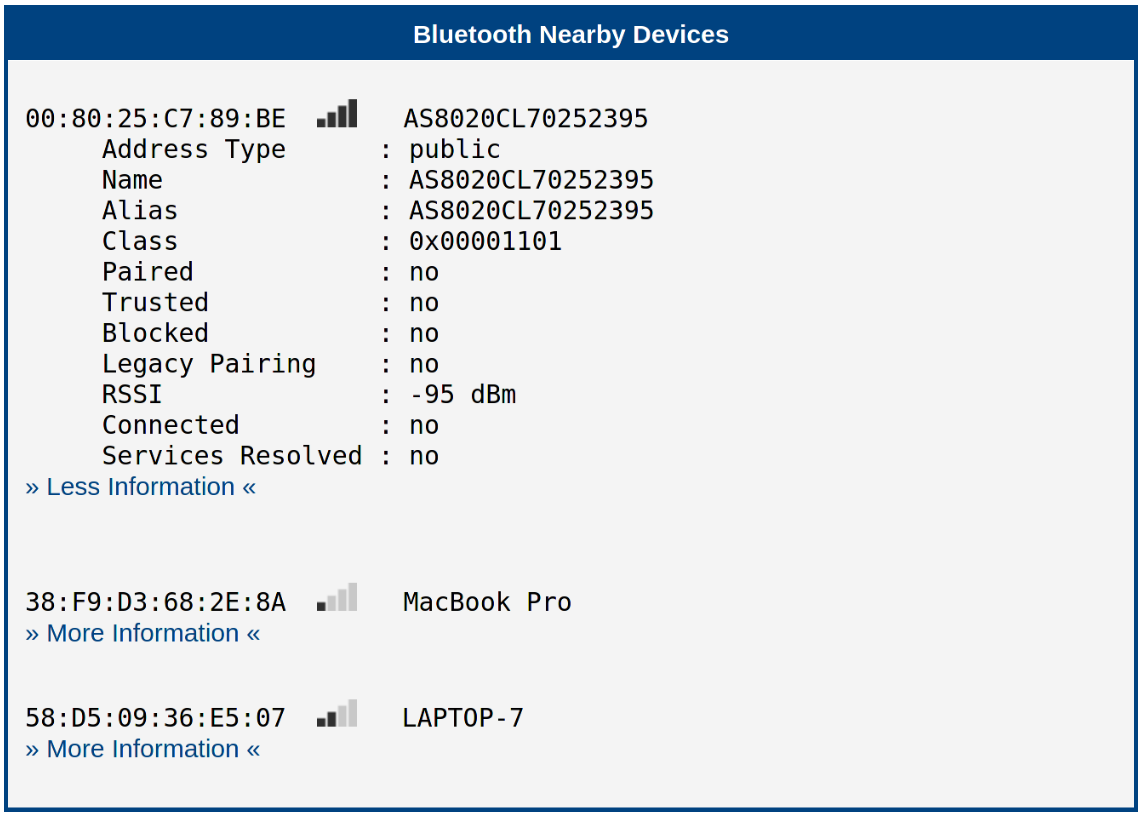
```
Address              : C0:EE:40:46:76:13
Address Type         : public
Name                 : ICR-3231W
Alias                : Router 4
Class                : 0x00000300
Powered              : yes
Discoverable         : yes
Discoverable Timeout : unlimited
Pairable             : no
Pairable Timeout     : 20 sec
Discovering          : yes
UUIDs                : 00001801-0000-1000-8000-00805f9b34fb Generic Attribute Profile
UUIDs                : 0000180a-0000-1000-8000-00805f9b34fb Device Information
UUIDs                : 00001200-0000-1000-8000-00805f9b34fb PnP Information
UUIDs                : 00001800-0000-1000-8000-00805f9b34fb Generic Access Profile
Modalias             : usb:v1D6Bp0246d0537
Roles                : central, peripheral
```

```
Active Instances     : 0
Supported Instances  : 5
Supported Includes   : tx-power, appearance, local-name
```

Figure 2: Status

4.2 Nearby Devices

List of discoverable Bluetooth devices nearby. List is dynamic and the device is discarded when does not cast more than 30 seconds. But the list itself does not refresh automatically on the screen, manual refresh is needed. Detailed information about device is displayed after expanding.


Bluetooth Nearby Devices

```
00:80:25:C7:89:BE    AS8020CL70252395
     Address Type     : public
     Name             : AS8020CL70252395
     Alias            : AS8020CL70252395
     Class            : 0x00001101
     Paired           : no
     Trusted          : no
     Blocked          : no
     Legacy Pairing   : no
     RSSI             : -95 dBm
     Connected        : no
     Services Resolved : no
```
» Less Information «

```
38:F9:D3:68:2E:8A    MacBook Pro
```
» More Information «

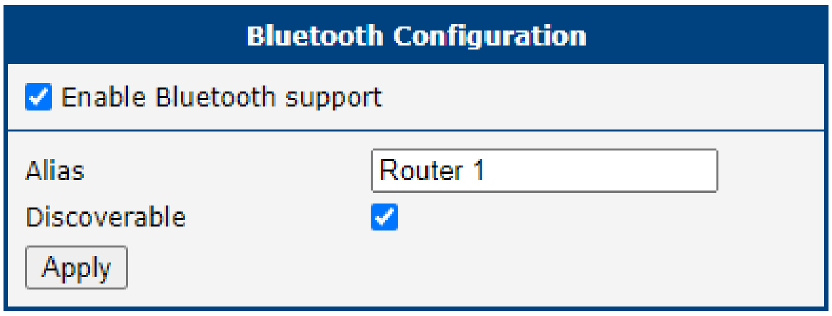
```
58:D5:09:36:E5:07    LAPTOP-7
```
» More Information «

Figure 3: Nearby devices

5. Configuration

5.1 Global

All Bluetooth router app settings can be configured by clicking on the *Global* item in the main menu of module web interface. An overview of configurable items is given below.



Bluetooth Configuration

- [✓] Enable Bluetooth support

Alias	Router 1
Discoverable	✓

Figure 4: Configuration

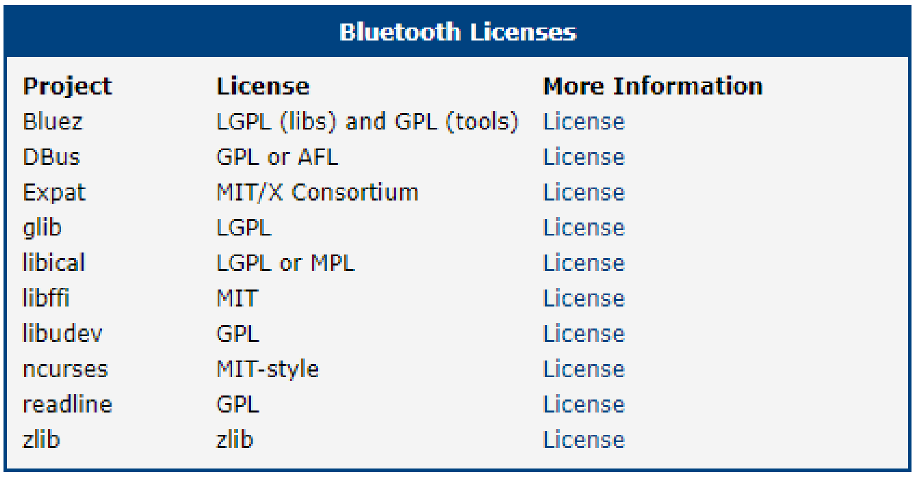
Item	Description
Enable Bluetooth support	Enables Bluetooth functionality.
Alias	Alias of the router when displayed in search results on the foreign devices
Discoverable	Indicates wether the router could be discovered via search on the foreign devices

Table 1: Configuration items description

6. General

6.1 licenses

Summarizes Open-Source Software (OSS) licenses used by this module.



Bluetooth Licenses

Project	License	More Information
Bluez	LGPL (libs) and GPL (tools)	License
DBus	GPL or AFL	License
Expat	MIT/X Consortium	License
glib	LGPL	License
libical	LGPL or MPL	License
libffi	MIT	License
libudev	GPL	License
ncurses	MIT-style	License
readline	GPL	License
zlib	zlib	License

Figure 5: licenses

7. Command line tools

- **bluetoothctl** - powerful command line utulity for discovery, connect, disconnect, scan, pair etc. You will find more on how to use this tool in shell scripts to work with BLE sensors in examples 1 and 2
- **btmon** - Bluetooth monitor.
- **dbus-monitor** - command is used to monitor messages going through a D-Bus message bus. *dbus-monitor* has two different output modes, the 'classic'-style monitoring mode and profiling mode. The profiling format is a compact format with a single line per message and microsecond-resolution timing information. The –profile and –monitor options select the profiling and monitoring output format respectively. If neither is specified, dbus-monitor uses the monitoring output format. *dbus-monitor* is not part of the Bluetooth, but its closely connected with BluZ, which D-Bus uses primarily for communication with applications.
- **dbus-send** - used to send a message to a D-Bus message bus. This tool is useful for testing end debuging. *dbus-send* is not part of the Bluetooth, but its closely connected with BluZ, which D-Bus uses primarily for communication with applications.
- **l2ping** - L2ping sends a L2CAP echo request to the Bluetooth MAC address *bd_addr* given in dotted hex notation. This tool is useful for testing end debuging.
- **l2test** - Tool for testing Bluetooth communication on lower level.
  Example:
  On router run "l2test -r" and on PC with Bluetooth run "l2test -s BT_ROUTER_ADDRESS" and you should see on router that Bluetooth data are received.

8. Examples

Bluetooth is used to transfer general data. In addition BLE provided more ways how to send data - very short data are often send as manufacturer data item in iBeacon advertising packets. For more complex cases are services/characteristics used as defined in Bluetooth standard. In any cases you will need another software for specific data.

The following examples demonstrate how to Bluetooth use capabilities in customer projects. They cover different Bluetooth communication types as well as different programming languages and environments. We hope that a combination of all these examples covers most of your project issues.

You can also find other Bluetooth examples in the Node-RED guide (examples 4 and 5).

8.1 Reading from a BLE sensor in the Shell script

In this example we tell the temperature from a TokenCube BLE tag on the SMS request.

```
E4:AA:EC:37:05:A1       standard demo
     Address Type      : public
     Name              : standard demo
     Alias             : standard demo
     Paired            : no
     Trusted           : no
     Blocked           : no
     Legacy Pairing    : no
     RSSI              : -41 dBm
     Connected         : no
     Service Data      : 0000fe95-0000-1000-8000-00805f9b34fb 0x30588b0958a10537ecaae408
     Services Resolved : no
     Advertising Flags : 0x06
```

Figure 6: Manufacturer data for Example 1

The above-mentioned sensor advertises data via the manufacturer data item. We need to know the data structure to process it. Advantech does not provide this information. You have to ask the sensor vendor. Brief information for this particular example follows:

Byte Nr.	Description	Value
0..1	Manufacturer ID	0xFFEE
2	Hardware ID	0x04 – Token version 4
3	Firmware version	0x01
4	Page number – first nibble is total number of pages, second nibble is page number	0x21 or 0x22 – two pages, first or second page
5	Sensor identifier	0x01..0x0A – normal mode, 0x81..0x8A – alarm mode; see next table for normal mode

Continued on the next page

Continued from previous page

Byte Nr.	Description	Value
6..x1	Sensor value	—
(x1+1)	Next sensor identifier	—
(x1+2)..x2	Next sensor value	—

Table 1: Data structure

ID	Sensor Type	Values	Interpretation	Value
0x01	Temperature	°C	int16, BE	0x14 0x03 = 5123 = 51.23 °C
0x04	Humidity	% RH	int16, BE	0x10 0x87 = 4231 = 42.31 %
other sensor IDs . . .				

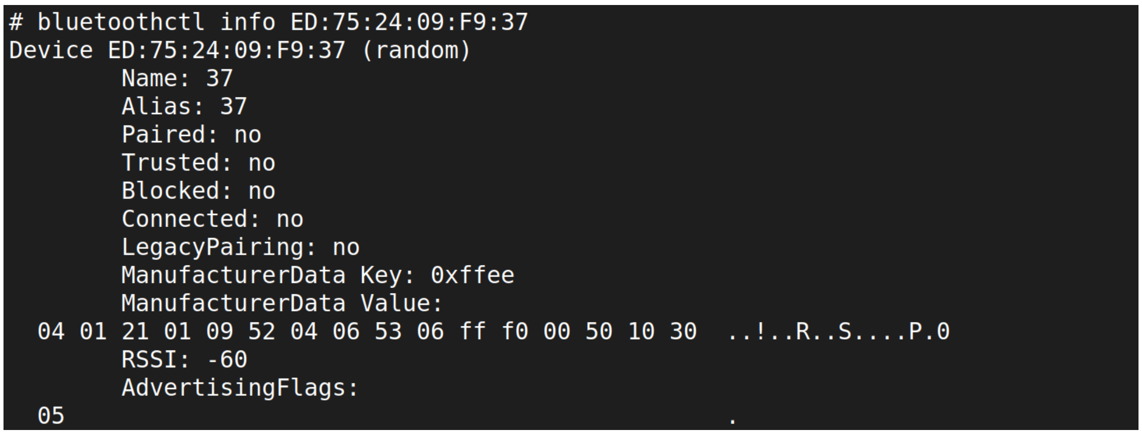
Table 2: Data interpretation

The tag advertises data alternately in two different records due to the limited size of manufacturer data. You can see an example of that data with marked temperature bytes representing 22.3 °C bellow:

```
Manufacturer Data : 0xffee 0x0401210108b60409c206ff80ff40f020
Manufacturer Data : 0xffee 0x0401221f00054ec50a64
```

Figure 7: Manufacturer data for example 2

We will use the standard BlueZ tool – *bluetoothctl* to get data



```
# bluetoothctl info ED:75:24:09:F9:37
Device ED:75:24:09:F9:37 (random)
        Name: 37
        Alias: 37
        Paired: no
        Trusted: no
        Blocked: no
        Connected: no
        LegacyPairing: no
        ManufacturerData Key: 0xffee
        ManufacturerData Value:
  04 01 21 01 09 52 04 06 53 06 ff f0 00 50 10 30  ..!..R..S....P.0
        RSSI: -60
        AdvertisingFlags:
  05                                                .
```

Figure 8: Use of bluetoothctl

And we will process it with the *awk* tool. We try to read data several times, as we need the page containing temperature data.

Note: Many sensors have a much easier data structure then this tag (one data block only, values defined by fixed position in data stream instead prefixes and so on). Even though the temperature data should be found by the ID 0x01, our tested tags return temperature as the first value every time, thus we read it fixed as first in our example to code be clearer and simpler to understand.

Put result script to `/var/scripts/sms`. Don't forgot to replace address ED:75:24:09:F9:37 with your tag address and set the script as executable.

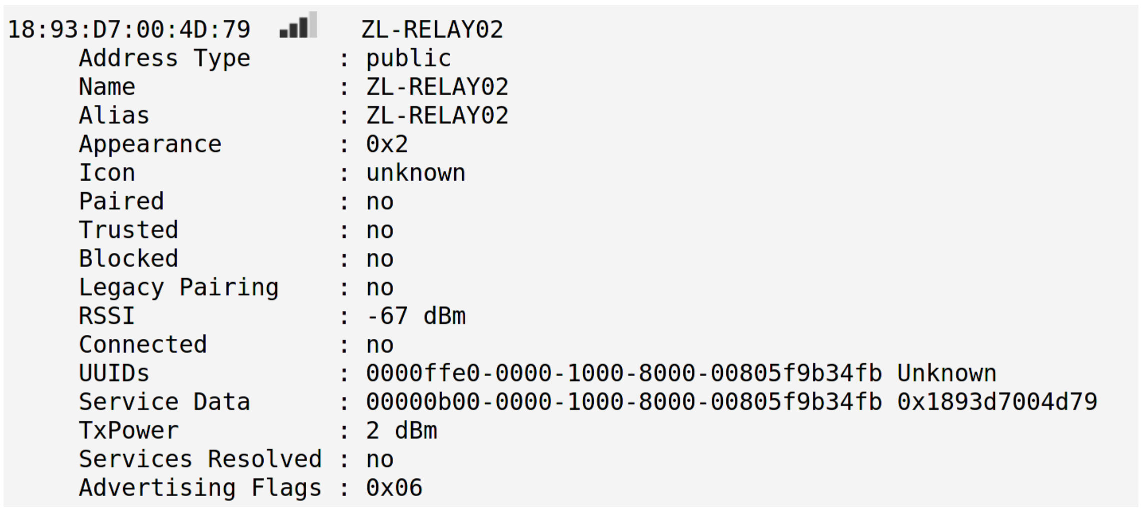
```
#/bin/sh

if [ "$3" == "TEMPERATURE" ]; then
  ATTEMPT=200
  while [ $ATTEMPT -gt 0 -a "$TEMPERATURE" == "" ]; do
    TEMPERATURE=‘bluetoothctl info ED:75:24:09:F9:37 | \
    awk ’/ManufacturerData Value:/ {next} /^\s+04 01 21/ \
        {print (("0x"$5)*256 + ("0x"$6))/100;exit;}’‘
    ATTEMPT=$(($ATTEMPT - 1))
  done

  if [ "$TEMPERATURE" != "" ]; then
    gsmsms $2 "Temperature is $TEMPERATURE degree Celsius"
  else
    gsmsms $2 "Temperature value is not available"
  fi
fi
```

Now when you send SMS “temperature” (mwan daemon passes it as argument $3 to you script, case insensitive) to your router’s SIM phone number, you will get back the temperature value to the original phone number (argument $2). Note, you need mwan up so that SMS receiving works. It is also good to limit accepting originated phone numbers in the real application (consult it with the router configuration manual). Note, also the `/var/scripts` folder is erased on reboot so you must renew the script every time, e.g. copy it with the system Startup script or with the *init* script in your own Router App.

Figure 9: TokenCube BLE tag

8.2 Writing to a BLE device in the Shell script

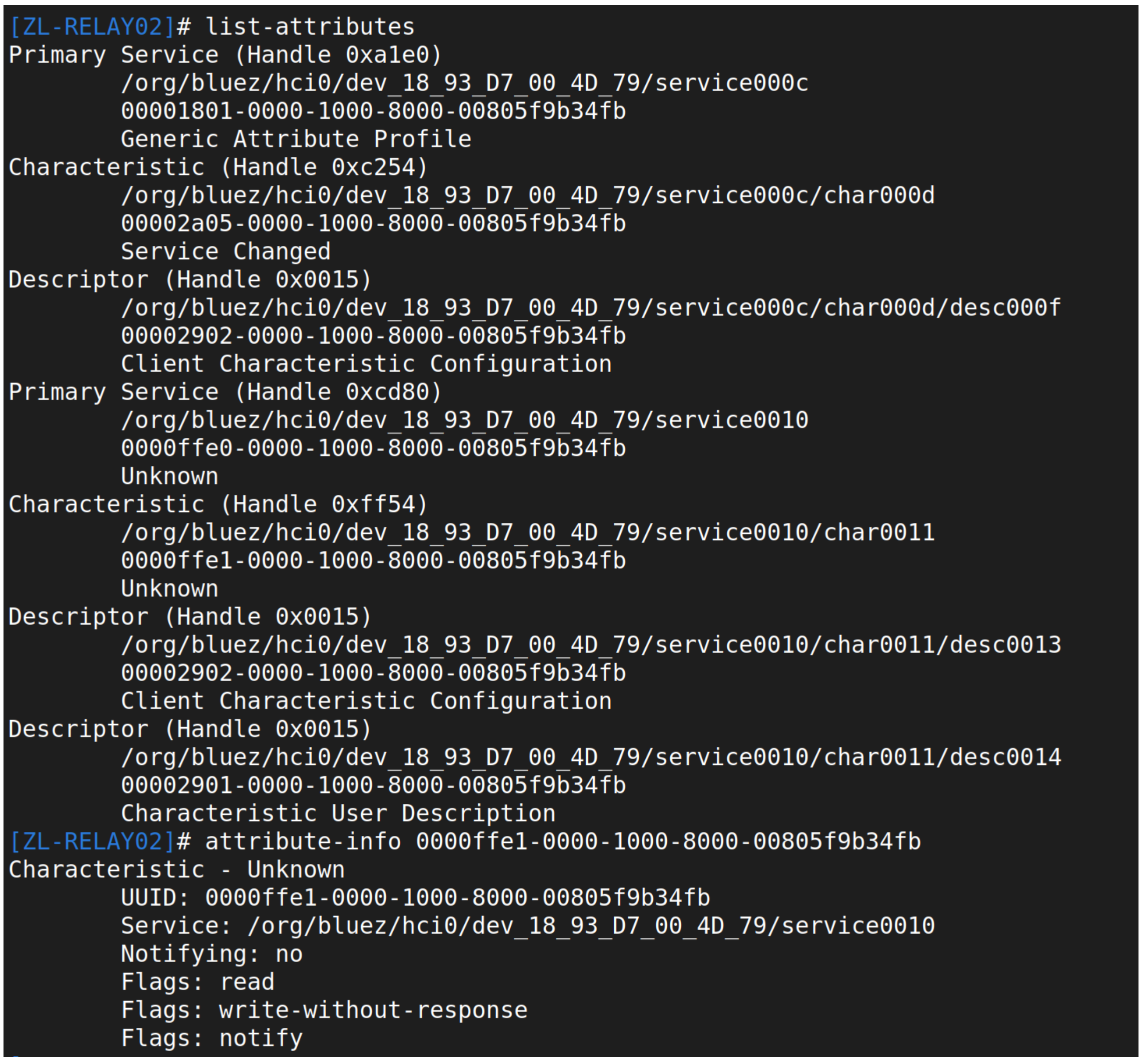
Some devices are switched On/Off via Jolland IoT ZL-R02 BLE relay module at specified times in this example.



```
18:93:D7:00:4D:79        ZL-RELAY02
     Address Type      : public
     Name              : ZL-RELAY02
     Alias             : ZL-RELAY02
     Appearance        : 0x2
     Icon              : unknown
     Paired            : no
     Trusted           : no
     Blocked           : no
     Legacy Pairing    : no
     RSSI              : -67 dBm
     Connected         : no
     UUIDs             : 0000ffe0-0000-1000-8000-00805f9b34fb Unknown
     Service Data      : 00000b00-0000-1000-8000-00805f9b34fb 0x1893d7004d79
     TxPower           : 2 dBm
     Services Resolved : no
     Advertising Flags : 0x06
```

Figure 10: Manufacturer data

Namely the module ZL-RC02V3 uses the BLE service and characteristic to control relays. You can explore BLE charcteristics with *bluetoothctl* tool. Start it, connect to the relay module with *connect* command (e.g. *connect 18:93:D7:00:4D:79*) and go to the gatt submenu with *menu gatt*. Then you can use *list-attributes* and *attribute-info*. We are interested in *0000ffe1-0000-1000-8000-00805f9b34fb* characteristic.



```
[ZL-RELAY02]# list-attributes
Primary Service (Handle 0xa1e0)
        /org/bluez/hci0/dev_18_93_D7_00_4D_79/service000c
        00001801-0000-1000-8000-00805f9b34fb
        Generic Attribute Profile
Characteristic (Handle 0xc254)
        /org/bluez/hci0/dev_18_93_D7_00_4D_79/service000c/char000d
        00002a05-0000-1000-8000-00805f9b34fb
        Service Changed
Descriptor (Handle 0x0015)
        /org/bluez/hci0/dev_18_93_D7_00_4D_79/service000c/char000d/desc000f
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
Primary Service (Handle 0xcd80)
        /org/bluez/hci0/dev_18_93_D7_00_4D_79/service0010
        0000ffe0-0000-1000-8000-00805f9b34fb
        Unknown
Characteristic (Handle 0xff54)
        /org/bluez/hci0/dev_18_93_D7_00_4D_79/service0010/char0011
        0000ffe1-0000-1000-8000-00805f9b34fb
        Unknown
Descriptor (Handle 0x0015)
        /org/bluez/hci0/dev_18_93_D7_00_4D_79/service0010/char0011/desc0013
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
Descriptor (Handle 0x0015)
        /org/bluez/hci0/dev_18_93_D7_00_4D_79/service0010/char0011/desc0014
        00002901-0000-1000-8000-00805f9b34fb
        Characteristic User Description
[ZL-RELAY02]# attribute-info 0000ffe1-0000-1000-8000-00805f9b34fb
Characteristic - Unknown
        UUID: 0000ffe1-0000-1000-8000-00805f9b34fb
        Service: /org/bluez/hci0/dev_18_93_D7_00_4D_79/service0010
        Notifying: no
        Flags: read
        Flags: write-without-response
        Flags: notify
```

Figure 11: BLE characteristics

Among other things, we can find this information in the relay module documentation:

Command	Byte sequence
Close the first relay	C5 04 XX XX XX XX XX XX XX XX AA
Open the first relay	C5 07 XX XX XX XX XX XX XX XX AA
Close the second relay	C5 05 XX XX XX XX XX XX XX XX AA
Open the second relay	C5 06 XX XX XX XX XX XX XX XX AA
Where 8 XX bytes is a password. Default password is “12345678” (in ASCII).	

Table 3: Relay control commands

You must contact the relay module vendor to get the full documentation. Advantech does not provide it.

We will also use *bluetoothctl* to control relays in the final example. Put the next script to /var/scripts/relays

```
#/bin/sh

case "$1" in
  ON)
    R1=0x04
    ;;
  OFF)
    R1=0x06
    ;;
  *)
    echo "The first relay state is invalid"
    exit 1
    ;;
esac

case "$2" in
  ON)
    R2=0x05
    ;;
  OFF)
    R2=0x07
    ;;
  *)
    echo "The second relay state is invalid"
    exit 1
    ;;
esac

bluetoothctl connect 18:93:D7:00:4D:79
if [ $? -ne 0 ]; then
  logger -t relays "Faild to connect to relays"
fi

echo -e \
"menu gatt\n" \
"select-attribute 0000ffe1-0000-1000-8000-00805f9b34fb\n" \
"write \"0xC5 $R1 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0xAA\"\n" \
"write \"0xC5 $R2 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0xAA\"\n" \
| bluetoothctl

bluetoothctl disconnect 18:93:D7:00:4D:79
```

and the following as */var/scripts/crontab*

```
00 02 * * * root /var/scripts/relays ON OFF
10 02 * * * root /var/scripts/relays ON ON
00 03 * * * root /var/scripts/relays OFF OFF
```

Finally, run the cron daemon

```
  /etc/init.d/cron start
```

Now it should switch on the first relay at 2 am, at the second at 10 minutes later and switch off both relays at 3 am every day.

Note: The */var/scripts* folder is erased on reboot so you must renew the script every time, e.g. copy it with the system Startup script or with the *init* script in your own Router App.

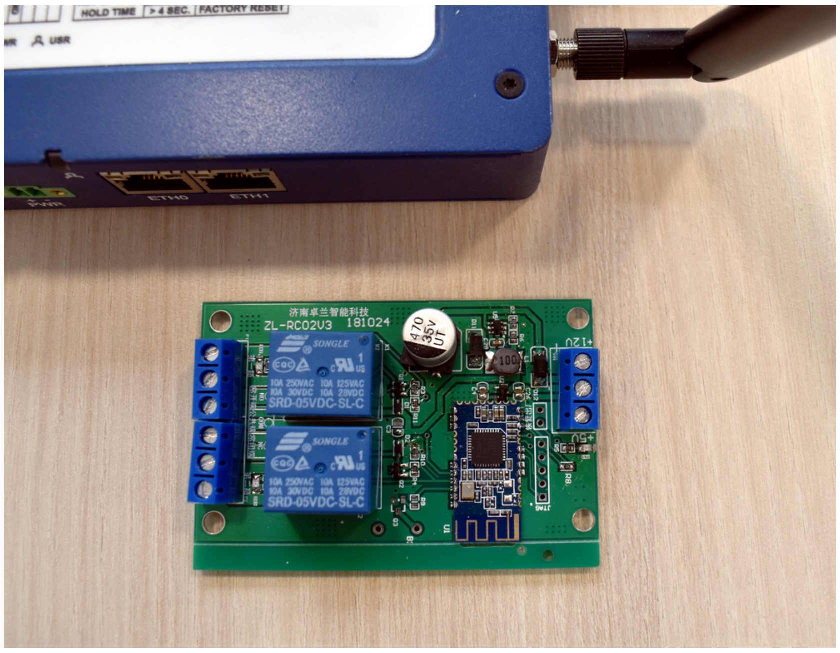
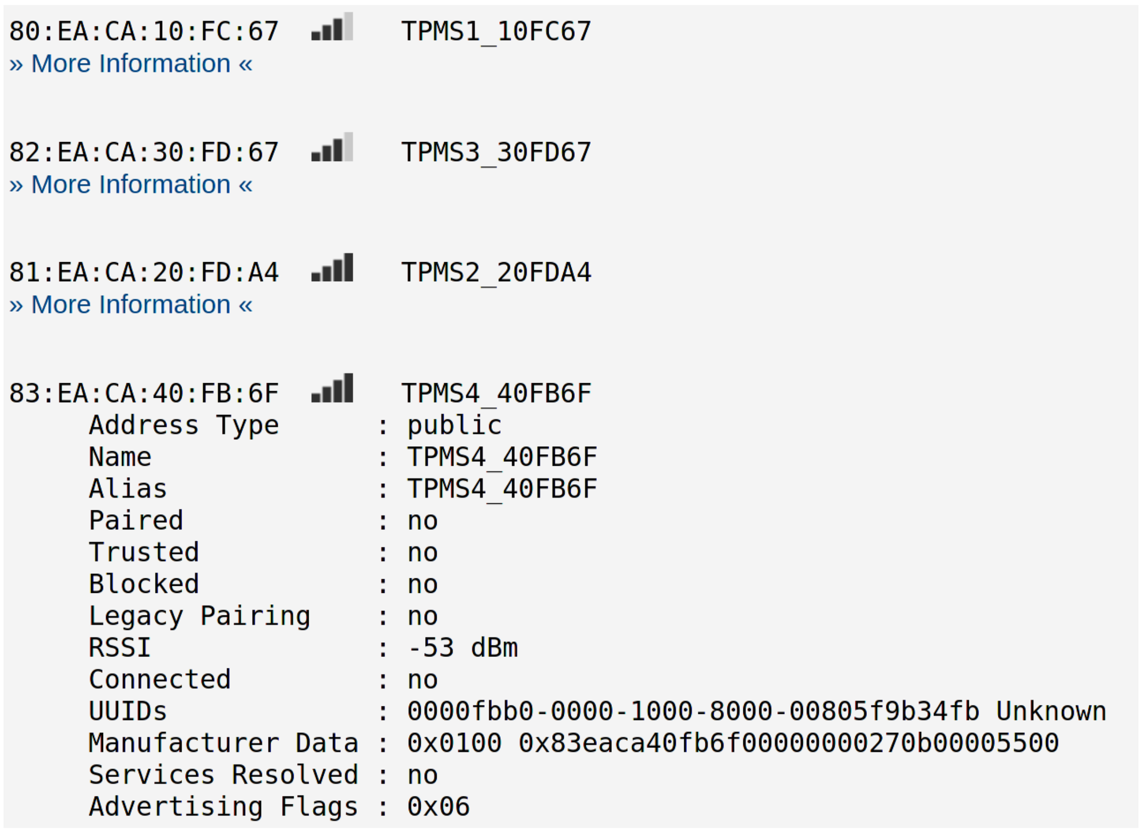


Figure 12: Jollan relay

8.3 Reading a BLE sensor in C

This example demonstrates how to read data from BLE TPMS (Tyre Pressure Monitoring Sensor) in C language.



```
80:EA:CA:10:FC:67        TPMS1_10FC67
» More Information «


82:EA:CA:30:FD:67        TPMS3_30FD67
» More Information «


81:EA:CA:20:FD:A4        TPMS2_20FDA4
» More Information «


83:EA:CA:40:FB:6F        TPMS4_40FB6F
     Address Type      : public
     Name              : TPMS4_40FB6F
     Alias             : TPMS4_40FB6F
     Paired            : no
     Trusted           : no
     Blocked           : no
     Legacy Pairing    : no
     RSSI              : -53 dBm
     Connected         : no
     UUIDs             : 0000fbb0-0000-1000-8000-00805f9b34fb Unknown
     Manufacturer Data : 0x0100 0x83eaca40fb6f00000000270b00005500
     Services Resolved : no
     Advertising Flags : 0x06
```

Figure 13: Sensor data

Used sensors propagates data via manufacturer data advertising item. Note, the sensors broadcast at very long interval and when they detect any pressure change to safe a battery. So you a need permanently running application – daemon for the real application. The example is written to run in front and show results for demonstration. You can dismount/mount sensor to the tyre to force it to send data.

We have two options with BlueZ – HCI and D-Bus. We will use HCI API in this example. As it is complicated for advanced topics, BlueZ authors recommend a newer D-Bus API. See the next example.

Note: You need Linux environment for next work. Ubuntu is sugested.

This example is a bit complex as we need a cross compiler and several dependencies. We will solve the first prerequisite with our Router App SDK. Please pull the following from the public git:

```
git clone https://marek_cernocky@bitbucket.org/bbsmartworx/modulessdk.git
git clone https://marek_cernocky@bitbucket.org/bbsmartworx/toolchains.git
```

Rename the first repository to *ModulesSDK*. Install deb or rpm packages from the second repository. You can find more details about how to build the Router App in the DevZone section on `www.advantech.cz`.

Makefile solves dependencies by downloading their source codes from Internet and building them right way. Note, it tries to built only the necessary parts.

Now switch to ModulesSDK/modules and copy the template to our new project tyres. You can change the content of *tyres/merge/etc/name* and *tyres/merge/etc/version* (its not essential for our example).

Then remove all content of the *tyres/source* folder and place the following two files to it:

Makefile

```
include ../../../Rules.mk


DEPSDIR = deps.$(PLATFORM)
INSTDIR = $(CURDIR)/$(DEPSDIR)/usr

DEPENDS += $(DEPSDIR)

CPPFLAGS += -I$(INSTDIR)/include -Wno-missing-braces
LDFLAGS += -L$(INSTDIR)/lib -pthread
LDLIBS  += -lbluetooth -lncurses

TYRES_EXE = tyres
TYRES_SRC = tyres.c

$(eval $(call build-program, $(TYRES_EXE), $(TYRES_SRC)))

$(DEPSDIR):
@mkdir $(DEPSDIR)
@echo "Downloading␣dependencies␣sources"
@wget -P $@ https://github.com/libexpat/libexpat/releases/download/R_2_2_10/expat-2.2.10.tar.
    bz2
@wget -P $@ https://dbus.freedesktop.org/releases/dbus/dbus-1.13.18.tar.xz
@wget -P $@ ftp://ftp.cwru.edu/pub/bash/readline-8.1.tar.gz
@wget -P $@ ftp://ftp.invisible-island.net/ncurses/ncurses-6.2.tar.gz
@wget -P $@ ftp://sourceware.org/pub/libffi/libffi-3.3.tar.gz
@wget -P $@ https://zlib.net/zlib-1.2.11.tar.xz
@wget -P $@ https://download.gnome.org/sources/glib/2.56/glib-2.56.4.tar.xz
@wget -P $@ http://www.kernel.org/pub/linux/bluetooth/bluez-5.55.tar.xz
@echo "Extractings␣dependencies"
@tar -x -C $@ -f $@/expat-2.2.10.tar.bz2; mv $@/expat-2.2.10 $@/expat
@tar -x -C $@ -f $@/dbus-1.13.18.tar.xz; mv $@/dbus-1.13.18 $@/dbus
@tar -x -C $@ -f $@/readline-8.1.tar.gz; mv $@/readline-8.1 $@/readline
@tar -x -C $@ -f $@/ncurses-6.2.tar.gz; mv $@/ncurses-6.2 $@/ncurses
@tar -x -C $@ -f $@/libffi-3.3.tar.gz; mv $@/libffi-3.3 $@/libffi
@tar -x -C $@ -f $@/zlib-1.2.11.tar.xz; mv $@/zlib-1.2.11 $@/zlib
@tar -x -C $@ -f $@/glib-2.56.4.tar.xz; mv $@/glib-2.56.4 $@/glib
@tar -x -C $@ -f $@/bluez-5.55.tar.xz; mv $@/bluez-5.55 $@/bluez
@echo "Building␣dependencies"
@cd $@/expat; ./configure --host=$(HOST) --prefix=$(INSTDIR) --disable-shared CC="$(CC)" CFLAG
    ="$(CFLAGS)"; make; make install
@cd $@/dbus; ./configure --host=$(HOST) --prefix=$(INSTDIR) --disable-shared --enable-static
    --disable-systemd --disable-selinux --disable-tests CC="$(CC)" CPPFLAGS="-I$(INSTDIR)/
    include" CFLAG="$(CFLAGS)" LDFLAGS="-L$(INSTDIR)/lib"; make; make install
@cd $@/readline; ./configure --host=$(HOST) --prefix=$(INSTDIR) --disable-shared --enable-
    static --disable-install-examples CC="$(CC)" CFLAG="$(CFLAGS)"; make; make install
@cd $@/ncurses; ./configure --host=$(HOST) --prefix=$(INSTDIR) --with-shared=no --without-
    progs --without-tests --without-manpages --disable-database --with-fallbacks=xterm-256color
     CC="$(CC)" CFLAG="$(CFLAGS)"; make; make install
@cd $@/libffi; ./configure --host=$(HOST) --prefix=$(INSTDIR) --disable-shared --disable-
    builddir CC="$(CC)" CFLAG="$(CFLAGS)"; make; make install
@cd $@/zlib; ./configure --prefix=$(INSTDIR) --static; make CC="$(CC)" CFLAG="$(CFLAGS)"; make
     install
@cd $@/glib; ./configure --host=$(HOST) --prefix=$(INSTDIR) --disable-shared --enable-static
```

```
      --with-pcre=internal --disable-fam --disable-libelf --disable-libmount --disable-selinux --
      disable-man --disable-debug glib_cv_stack_grows=no glib_cv_uscore=no CC="$(CC)" CFLAG="$(
      CFLAGS)" LIBFFI_CFLAGS="-I$(INSTDIR)/include" LIBFFI_LIBS="-L$(INSTDIR)/lib␣-lffi"
      ZLIB_CFLAGS="-I$(INSTDIR)/include" ZLIB_LIBS="-L$(INSTDIR)/lib␣-lz"; make; make install
  @cd $@/bluez; ./configure --host=$(HOST) --prefix=$(INSTDIR) --enable-library --disable-shared
       --enable-static --disable-tools --disable-datafiles --disable-client --disable-obex --
      disable-monitor --disable-mesh --disable-cups --disable-systemd CC="$(CC)" CPPFLAGS="-I$(
      INSTDIR)/include" CFLAG="$(CFLAGS)" LIBS="-L$(INSTDIR)/lib␣-lncurses" GLIB_CFLAGS="-I$(
      INSTDIR)/include/glib-2.0␣-I$(INSTDIR)/lib/glib-2.0/include" GLIB_LIBS="-L$(INSTDIR)/lib␣-
      lglib-2.0␣-lgobject-2.0␣-lgio-2.0␣-lgmodule-2.0"; make; make install

  install:
  @install -d $(DESTDIR)/bin
  @install -m 755 $(OBJDIR)/$(TYRES_EXE) $(DESTDIR)/bin/

  clean:
  rm -rf $(OBJDIR) $(DEPSDIR)
```

tyres.c

```
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <syslog.h>

#include <ncurses/curses.h>

#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>

#define VALS_STR_SIZE 19

// to parse iBeacons
typedef struct {
  uint8_t       length;
  uint8_t       type;
  unsigned char data[0];
} ibeacon_rec_t;

// to store measurements
typedef struct {
  bdaddr_t      address;
  char          values[VALS_STR_SIZE];
} sensor_t;

sensor_t sensors[4] = {
  {{0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, "looking for sensor"}, // TPMS1-front left
  {{0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, "looking for sensor"}, // TPMS2-front right
  {{0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, "looking for sensor"}, // TPMS3-rear left
  {{0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, "looking for sensor"}, // TPMS4-rear right
};

// pretty output for measured data
static void draw_tractor() {
  char tractor[] =
    "123456789012345678        123456789012345678\n"
    "\n"
    "                          ##################\n"
    "   ###########          /-##################-\\\n"
    "   ###########       /---+-################## ||\n"
    "      | |            |                  |  ||\n"
    " /-----+-+-------------|    +-------+   |---/\n"
    ")| ==             == |      |      |    |\n"
    " | ==             == |      |      |    |>\n"
    ")| ==         0   == |      |      |    |\n"
    " \\-----+-+-------------|    +-------+   |---\\\n"
    "      | |            |                  |  ||\n"
    "   ###########       \\---+-################## ||\n"
    "   ###########          \\-##################-/\n"
    "                          ##################\n"
    "\n"
    "123456789012345678        123456789012345678";

  memcpy(tractor    , sensors[1].values, VALS_STR_SIZE - 1);
  memcpy(tractor + 29 , sensors[3].values, VALS_STR_SIZE - 1);
  memcpy(tractor + 689, sensors[0].values, VALS_STR_SIZE - 1);
  memcpy(tractor + 718, sensors[2].values, VALS_STR_SIZE - 1);

  mvaddstr(0, 0, tractor);
  refresh();
}

int main() {
  int                 hci_id;                      // adapter id
  int                 hci_sock = 0;                // socket to work with hci
  struct              hci_filter nf;               // filter for scanner
  unsigned char       buffer[HCI_MAX_EVENT_SIZE]; // buffer to read packet in
  int                 len;                         // read data length
  evt_le_meta_event   *meta;                       // pointer to event
  le_advertising_info *info;                       // pointer to ble advertisement
  ibeacon_rec_t       *rec;                        // for iBeacon parsing
  unsigned int        i;                           // for general iterations
  unsigned int        report;                      // for event reports iteration
  int                 pos;                         // curren position in data
  unsigned int        sensor;                      // selected sensor
  uint32_t            pressure;                    // pressure value
  uint16_t            temperature;                 // temperature value
  fd_set              fds;                         // set of descriptors for select()
  int                 exit_code = EXIT_FAILURE;    // status returned to terminal

  // get the first adapter identifier
  // if you dont connect an external adapter then it should be 0 every time
  hci_id = hci_get_route(NULL);
  if (hci_id < 0) {
    syslog(LOG_WARNING, "Bluetooth adapter is off\n");
    goto clean_finish;
  }

  // connect to the first adapter
  hci_sock = hci_open_dev(hci_id);
  if (hci_sock < 0) {
    syslog(LOG_ERR, "Connecting to the adapter failed: %s\n", strerror(errno));
    goto clean_finish;
```

```
}

// we want only advertisements
hci_filter_clear(&nf);
hci_filter_set_ptype(HCI_EVENT_PKT, &nf);
hci_filter_set_event(EVT_LE_META_EVENT, &nf);

if (setsockopt(hci_sock, SOL_HCI, HCI_FILTER, &nf, sizeof(nf)) < 0) {
  syslog(LOG_ERR, "Set HCI filter failed: %s\n", strerror(errno));
  goto clean_finish;
}

// set BLE scanning parameters; at first is necessary call stop scanning for
// case scanning running, otherwise set parameters failes
if (hci_le_set_scan_enable(hci_sock, 0x00, 0, 1000) < 0 ||
    hci_le_set_scan_parameters(hci_sock, 0x01, htobs(0x0010), htobs(0x0010), 0x00, 0x00, 1000) <
         0) {
  syslog(LOG_ERR, "Setting BLE scan parameters failed: %s\n", strerror(errno));
  goto clean_finish;
}

// start BLE scanning without filtering duplicity
if (hci_le_set_scan_enable(hci_sock, 0x01, 0, 1000) < 0) {
  syslog(LOG_ERR, "BLE scanning start failed: %s\n", strerror(errno));
  goto clean_finish;
}

// initialize screen
initscr();
curs_set(0);
draw_tractor();

// infinite loop
while (1) {
  // infinite wait for hci data ready or signal interrupt
  FD_ZERO(&fds);
  FD_SET(hci_sock, &fds);
  if (select(hci_sock + 1, &fds, NULL, NULL, NULL) < 0) {
    if (errno == EINTR)
      // it was signal
      exit_code = EXIT_SUCCESS;
    else
      syslog(LOG_ERR, "Waiting for data error: %s\n", strerror(errno));
    goto clean_finish;
  }

  // read data from hci socket
  len = read(hci_sock, buffer, sizeof(buffer));
  if (len == 0) {
    goto clean_finish;
  }

  // the first skipped byte is packet type
  meta = (evt_le_meta_event*) (buffer + (1 + HCI_EVENT_HDR_SIZE));
  // skipped data[0] is number of reports
  info = (le_advertising_info*) (meta->data + 1);
```

```
// cycle through reports
for (report = 1; report <= meta->data[0]; report++) {

  // walk through beacon data
  pos = 0;

  // we may not go out of records part and out of the read block
  while (pos < info->length - 1 && info->data + pos + info->data[pos] < buffer + len) {
    rec = (ibeacon_rec_t*)(info->data + pos);

    // we interested in type 0x09 (complete name) na 0xff (manuf. data) only
    switch (rec->type) {

      // the complete name
      case 0x09:
        // if advertised complete name is "TMPSx" where x is from 1 to 4
        if (rec->length == 13 && memcmp(&rec->data, "TPMS", 4) == 0 &&
            rec->data[4] >= '1' && rec->data[4] <= '4') {
          // get index to sensors array by sensor number
          sensor = rec->data[4] - '1';
          // store mac address
          memcpy(&sensors[sensor].address, &info->bdaddr, sizeof(bdaddr_t));
        }
        break;

      // the manufacturer data
      case 0xff:
        // check if it is one of sensors we intersted in
        for (i = 0; i < sizeof(sensors); i++) {
          if (memcmp(&sensors[i].address, &info->bdaddr, sizeof(bdaddr_t)) == 0) {
            // it is our sensor, let's go parse data
            // last byte: 0x00 = regular measurement, 0x01 immeasurable pressure
            if (rec->data[17] == 0x01) {
              memcpy(sensors[i].values, "not mesurable ", VALS_STR_SIZE - 1);
            } else {
              // bytes from 10 to 12 are bigendian pressure
              pressure = rec->data[8] + (rec->data[9] << 8) + (rec->data[10] << 16);
              // bytes 14 and 15 are bigendian temperature
              temperature = rec->data[12] + (rec->data[13] << 8);
              // store human readable values
              snprintf(sensors[i].values, VALS_STR_SIZE, "%5u kPa, %3u C ",
                       pressure/1000, temperature/100);
            }
            draw_tractor();
            break;
          }
        }
    }
    pos += 1 + rec->length;
  }

  // move to next report
  info = (le_advertising_info*) (((char*)info) + sizeof(le_advertising_info) + info->length +
      1);
}
```

```
  }

clean_finish:
  // restore screen
  endwin();

  if (hci_sock) {
    // stop BLE scanning
    hci_le_set_scan_enable(hci_sock, 0x00, 0, 1000);
    // disconnect from adapter
    close(hci_sock);
  }

  return exit_code;
}
```

Makefile is not commented it is out of scope of this documentation. All steps in C source are commented and some other notes follow.

Common steps for work with BLE sensors are:

- initialize – open device, set filter, set parameters, enable scanning)
- wait for data and read them – read() optionally with select()
- process read data with walk through structures (see bellow)
- stop – disable scanning, close device

Although you can directly work with HCI via socket only, it is a good idea to use BlueZ API with HCI structure definitions and higher level functions. We use the following structures when parsing received HCI event from scanning:

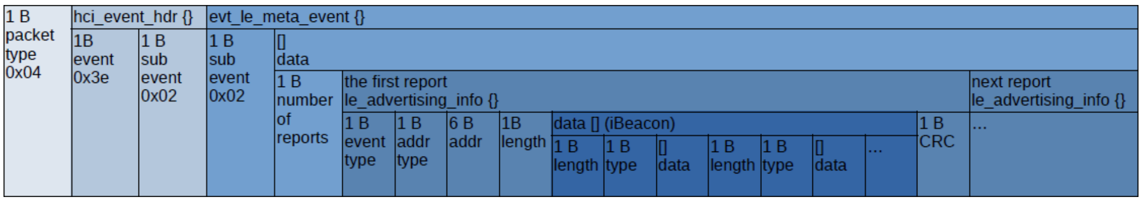
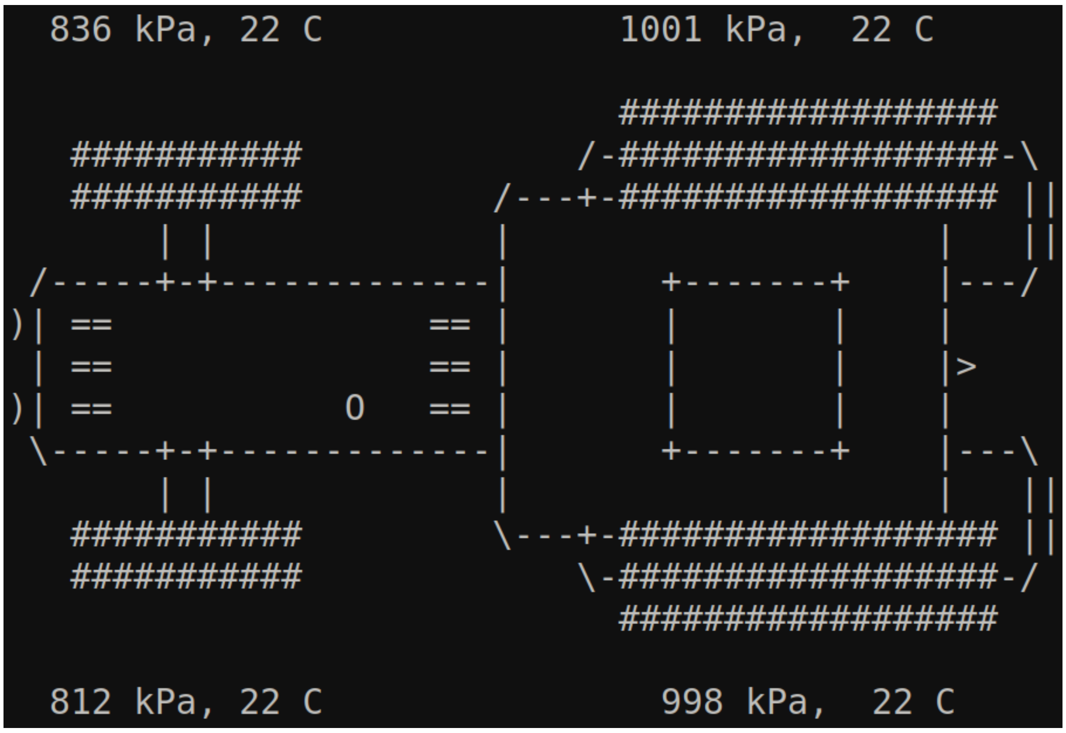


Figure 14: Structure of HCI event

Unfortunately API is missing definitions to work with beacon data as its structure is is not standardized and more proprietary variants exist (AltBeacon, iBeacon, URIBeacon. . .). Used sensors send iBeacons.

We need to know the manufacturer data means in order to get the pressure and temperature. The information necessary for this example you can be found in the code comments. Advantech does not provide the third party sensor documentation. You must ask the sensor vendor.

When you have everything prepared, run *make PLATFORM=v3*. You can find result binary *tyres* you can find in *obj.v3* subfolder. Copy it to the router (e.g. with SFTP) and run it from the terminal. Of course you can also install built Router App *tyres.v3.tgz* and then you find the binary in */opt/tyres/bin*. After execution you should see a similar screen:



```
  836 kPa, 22 C              1001 kPa,  22 C

                             ##################
   ###########             /-##################-\
   ###########         /---+-################## ||
       | |             |                    |   ||
 /-----+-+-------------|       +-------+    |---/
)| ==              == |       |       |    |
 | ==              == |       |       |    |>
)| ==          0   == |       |       |    |
 \-----+-+-------------|       +-------+    |---\
       | |             |                    |   ||
   ###########         \---+-################## ||
   ###########             \-##################-/
                             ##################

  812 kPa, 22 C                998 kPa,  22 C
```

Figure 15: Example 3 output

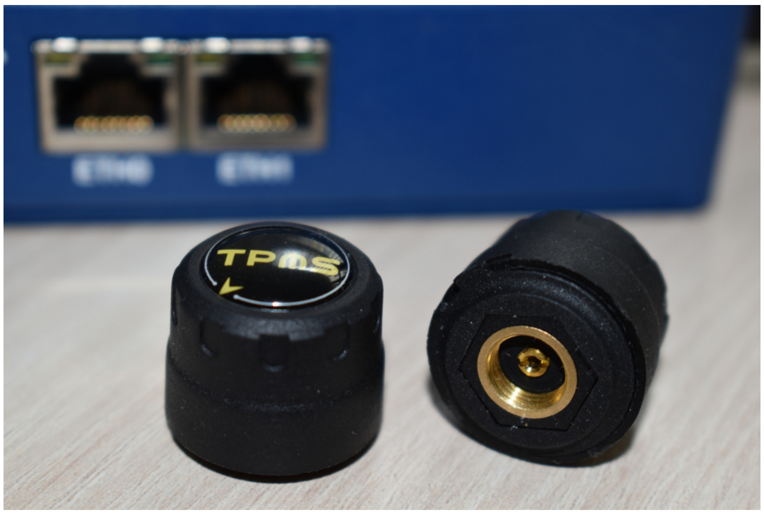


Figure 16: Tyre pressure sensor

9. Related Documents

You can obtain product-related documents on *Engineering Portal* at *icr.advantech.cz* address.

To get your router's *Quick Start Guide*, *User Manual*, *Configuration Manual*, or *Firmware* go to the *Router Models* page, find the required model, and switch to the *Manuals* or *Firmware* tab, respectively.

The *Router Apps* installation packages and manuals are available on the *Router Apps* page.

For the *Development Documents*, go to the *DevZone* page.