Displaying vehicle information with Raspberry PI

Introduction of the open source project "OBD display" for the world of Internet-of-Things (IoT) including an app.

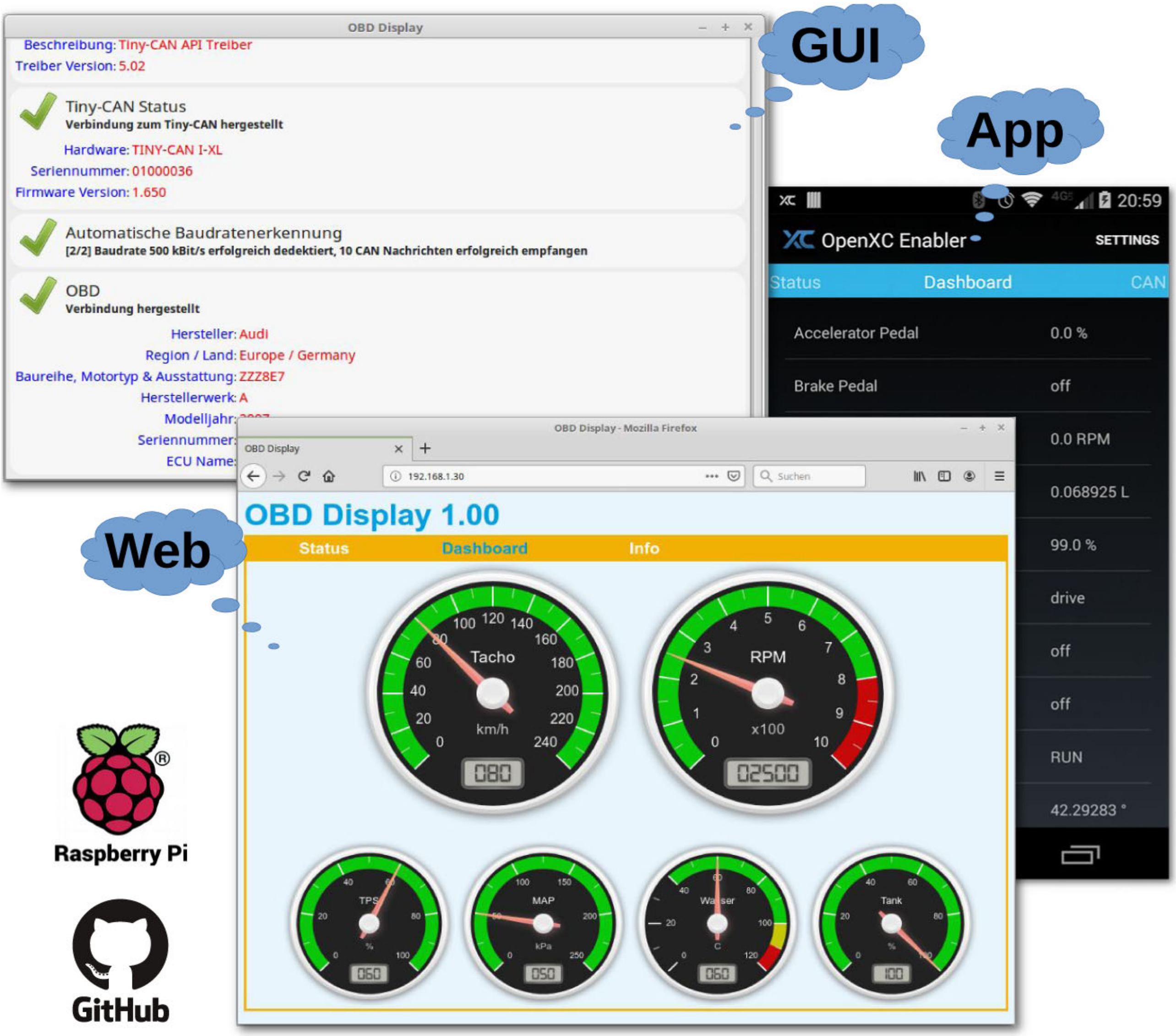

Figure 1: OBD display interfaces (Source: MHS-Elektronik)

Via the OBD-II interface, measurement data (SID 01_h), vehicle information such as chassis number/vehicle identification number (SID 09_h) and fault memory (diagnostic trouble codes, SID 03_h are queried via a CAN network. A list of all values that can be displayed is shown in the appendix. A Tiny-CAN is used as an interface adapter from the CAN network to the USB network. By using a standard USB-CAN adapter, the program can be used on any Linux PC. The software is written in C. GTK+ is used as GUI (graphical user interface). The graphic illustrates the functionality in a very simplified way.

The program flow even more detailed:

1. Load CAN API driver libmhstcan.so, query information about driver and Tiny-CAN hardware, configure and open CAN interface, forward received CAN messages and send CAN messages (can_device.c)
2. Monitor the connection and disconnection of the Tiny-CAN interface (can_dev_pnp.c)
3. Driver for the ISO-TP protocol (isotp.c), sending of single and segmented ISO-TP messages with data flow control. Receive single and segmented ISO-TP messages, including generated CAN messages for data flow control
4. Establish OBD connection, read VIN and supported PIDs, cyclically read the life data and read the error memory, errors are not deleted.

The vin_db.c module contains utility functions for breaking down the VIN in manufacturer, country, etc. The ▷

Table: List of all values that can be displayed. The prerequisite, of course, is that the vehicle also provides the data. The provided data is determined via supported PIDs

Value	Mode	PID
Supported PIDs in the range 01 - 20	01_h	00_h
Monitor status since DTCs cleared	01_h	01_h
Freeze DTC	01_h	02_h
Fuel system status	01_h	03_h
Calculated engine load	01_h	04_h
Engine coolant temperature	01_h	05_h
Short term fuel trim Bank 1	01_h	06_h
Long term fuel trim Bank 1	01_h	07_h
Short term fuel trim Bank 2	01_h	08_h
Long term fuel trim Bank 2	01_h	09_h
Fuel pressure (gauge pressure)	01_h	$0A_h$
Intake manifold absolute pressure	01_h	$0B_h$
Engine RPM	01_h	$0C_h$
Vehicle speed	01_h	$0D_h$
Timing advance	01_h	$0E_h$
Intake air temperature	01_h	$0F_h$
MAF air flow rate	01_h	10_h
Throttle position	01_h	11_h
Commanded secondary air status	01_h	12_h
Oxygen sensors present	01_h	13_h
Oxygen sensor 1	01_h	14_h
Oxygen sensor 2	01_h	15_h
Oxygen sensor 3	01_h	16_h
Oxygen sensor 4	01_h	17_h
Oxygen sensor 5	01_h	18_h
Oxygen sensor 6	01_h	19_h
Oxygen sensor 7	01_h	$1A_h$
Oxygen sensor 8	01_h	$1B_h$
OBD standards this vehicle conforms to	01_h	$1C_h$
Oxygen sensors present in 4 banks	01_h	$1D_h$
Auxiliary input status	01_h	$1E_h$
Run time since engine start	01_h	$1F_h$
Supported PIDs in the range 21 - 40	01_h	20_h
Distance traveled with malfunction indicator lamp on	01_h	21_h
Fuel rail pressure (relative to mainfold vacuum	01_h	22_h
Fuel rail gauge pressure (diesel, or gasoline direct injection)	01_h	23_h
Oxygen sensor 1	01_h	24_h
Oxygen sensor 2	01_h	25_h
Oxygen sensor 3	01_h	26_h
Oxygen sensor 4	01_h	27_h
Oxygen sensor 5	01_h	28_h
Oxygen sensor 6	01_h	29_h

Value	Mode	PID
Oxygen sensor 7	01_h	$2A_h$
Oxygen sensor 8	01_h	$2B_h$
Commanded EGR	01_h	$2C_h$
EGR error	01_h	$2D_h$
Commanded evaporative purge	01_h	$2E_h$
Fuel tank level input	01_h	$2F_h$
Warm-ups since codes cleared	01_h	30_h
Distance traveled since codes cleared	01_h	31_h
Evaporative system vapor pressure	01_h	32_h
Absolute barometric pressure	01_h	33_h
Oxygen sensor 1	01_h	34_h
Oxygen sensor 2	01_h	35_h
Oxygen sensor 3	01_h	36_h
Oxygen sensor 4	01_h	37_h
Oxygen sensor 5	01_h	38_h
Oxygen sensor 6	01_h	39_h
Oxygen sensor 7	01_h	$3A_h$
Oxygen sensor 8	01_h	$3B_h$
Catalyst temperature, bank 1, sensor 1	01_h	$3C_h$
Catalyst temperature, bank 2, sensor 1	01_h	$3D_h$
Catalyst temperature, bank 1, sensor 2	01_h	$3E_h$
Catalyst temperature, bank 2, sensor 2	01_h	$3F_h$
Supported PIDs in the range 41 - 60	01_h	40_h
Monitor status this drive cycle	01_h	41_h
Control module voltage	01_h	42_h
Absolute load value	01_h	43_h
Fuel-air commanded equivalence ratio	01_h	44_h
Relative throttle position	01_h	45_h
Ambient air temperature	01_h	46_h
Absolute throttle position B	01_h	47_h
Absolute throttle position C	01_h	48_h
Accelerator pedal position D	01_h	49_h
Accelerator pedal position E	01_h	$4A_h$
Accelerator pedal position F	01_h	$4B_h$
Commanded throttle actuator	01_h	$4C_h$
Time run with MIL on	01_h	$4D_h$
Time since trouble codes cleared	01_h	$4E_h$
Get DTCs	01_h	00_h
Supported PIDs	01_h	00_h
VIN message count	01_h	01_h
Get VIN	01_h	02_h
ECU name message count	01_h	09_h
Get ECU name	01_h	$0A_h$

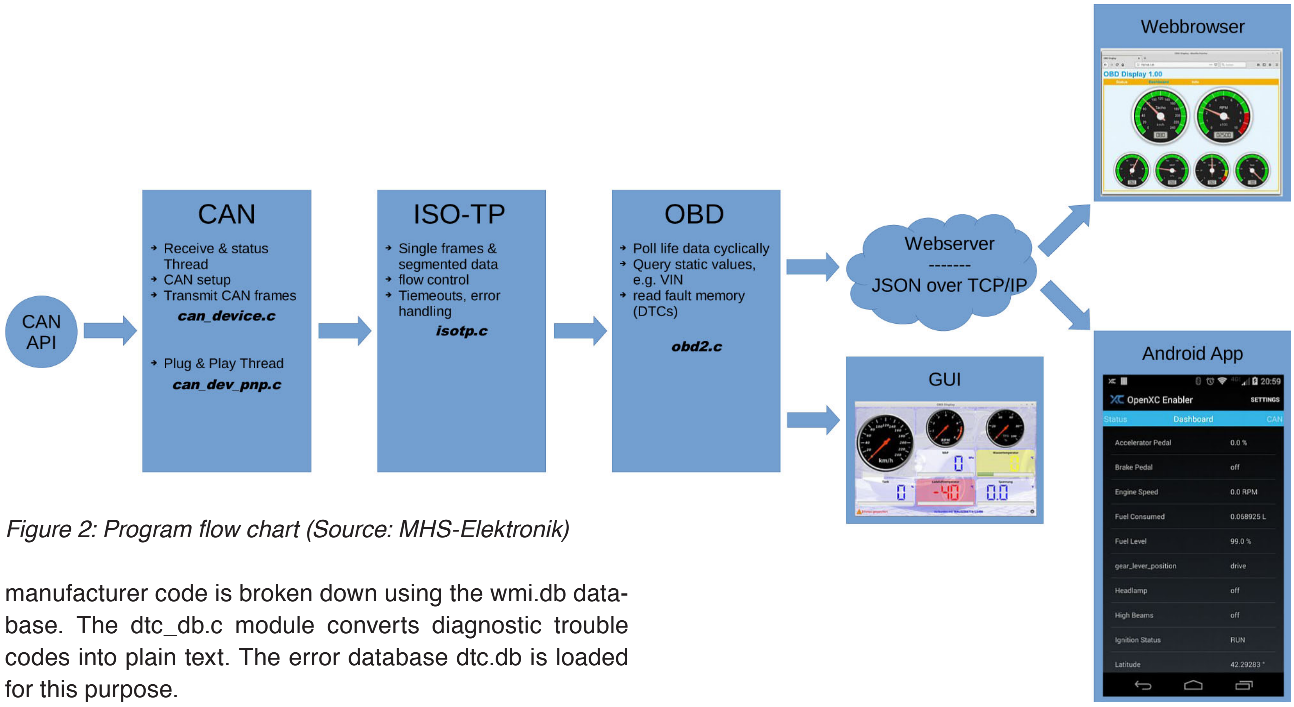

Figure 2: Program flow chart (Source: MHS-Elektronik)

manufacturer code is broken down using the wmi.db database. The dtc_db.c module converts diagnostic trouble codes into plain text. The error database dtc.db is loaded for this purpose.

Without IoT (Internet-of-Things), nothing runs today. The most important vehicle data is provided as HTML5 page via an Apache web server. A JSON over TCP/IP interface is available for apps.

The xml_database.c cyclically writes the dashboard.xml and status.xml files with the current measured values via the XMLDatabaseUpdate function. Here is an excerpt from the XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<dashboard>
<Speed> 0</Speed>
<Rpm> 0</Rpm>
....
</dashboard>
```

Since the XML files are only simple static one-dimensional structures, no XML library was used to write the files. Instead g_strdup_printf and the standard file I/O functions are used.

A Java script of the HTML page cyclically triggers a GET request, which reads the corresponding XML file according to the displayed page. The two modules sock_lib.c und open_xc.c are responsible for TCP/IP communication. The sock_lib.c module creates its own thread in which new socket connections and received data are processed. The open_xc.c module also generates an auxiliary thread that triggers the cyclic transmission of the OBD data. The used JSON message format is compatible to the open source project Open XC of Ford Bug Labs, so the Android, iOS libraries and apps of Open XC can be used. As soon as an app opens the TCP/IP socket, the OBD data is also transferred cyclically. Example of a data record:

```
{"name": "vehicle_speed", "value": 45}\0
```

A data record is completed with \0. It is also possible to send several data records in one package. Example:

```
{"name": ...}\0{"name": ...}\0
```

The app can also send commands to the software. Here is an example of a command and its response:

```
{"command":"platform","unix_time":0,"bypass":
false," bus":0,"enabled":false}\0
{"command_response": "platform", "message":
"Tiny-CAN & Pi", "status": true}\0
```

The open source project is hosted on Github and is licensed under the MIT license. The GIT project homepage describes the compilation, the required hardware, and the packages to be installed. Also the license text, numerous useful tips, e.g. how to turn off the mouse pointer, and some screenshots can be found there. The sources of the libmhstcan.so (Tiny-CAN API) are included in the Tiny-CAN software package and not part of the GIT repository. ◀

Author

Klaus Demlehner
MHS-Elektronik
info@mhs-elektronik.de
www.mhs-elektronik.de

Tuesday, March 17, 2020		
09:30 - 09:45	Holger Zeltwanger (CiA)	Conference opening
Keynote session **Chairperson: Holger Zeltwanger (CiA)**		
09:45 - 11:00	Carsten Schanze (VW)	Future of CAN from the prospective of an OEM
Session I: Physical layer **Chairperson: Carsten Schanze (VW)**		
11:00 - 11:30	Magnus-Maria Hell (Infineon)	The physical layer in the CAN XL world
11:30 - 12:00	Patrick Isensee (C&S Group)	The challenge of future 10-Mbit/s in-vehicle networks
12:00 - 12:30	Johnnie Hancock (Keysight)	Characterizing the physical layer of CAN FD
12:30 - 14:00		*Lunch break*
Session II: CAN XL data link layer **Chairperson: Reiner Zitzmann (CiA)**		
14:00 - 14:30	Florian Hartwich (Robert Bosch)	Introducing CAN XL into CAN networks
14:30 - 15:00	Dr. Arthur Mutter (Robert Bosch)	CAN XL error detection capabilities
15:00 - 15:30	Dr. Christian Senger (University of Stuttgart)	CRC error detection for CAN XL
15:30 - 16:00		*Coffee break*
Session III: CANopen testing **Chairperson: Uwe Koppe (Microcontrol)**		
16:00 - 16:30	Mark Schwager (Vector)	A new approach for simulating and testing of CANopen devices
16:30 - 17:00	Oskar Kaplun (CiA)	CANopen FD conformance testing – today and tomorrow
Session IV: CANopen FD **Chairperson: Christian Schlegel**		
17:00 - 17:30	Uwe Wilhelm (Peak), Christian Keydel (Emsa)	A simplified classic CANopen-to-CANopen FD migration path using smart bridges
17:30 - 18:00	Alexander Philipp (Emotas)	A theoretical approach for node-ID negotiation in CANopen networks
18:00 - 18:30	Yao Yao (CiA)	CANopen FD devices identification via new layer setting services (LSS)

Wednesday, March 18, 2020		
Session V: CAN FD lower layers **Chairperson: Dr. Frank Deicke (Fraunhofer IPMS)**		
09:00 - 09:30	Tony Adamson (NXP)	CAN signal improvement and designing 5-Mbit/s networks
19:30 - 10:00	Fred Rennig (ST Microelectronics)	A lightweight communication bus based on CAN FD for data exchange with small monolithic actuators and sensors
10:00 - 10:30	Kent Lennartsson (Kvaser)	Improved CAN-driver
10:30 - 11:00		*Coffee break*
Session VI: Engineering **Chairperson: Kent Lennartsson (Kvaser)**		
11:00 - 11:30	Nikos Zervas (Cast)	Designing a CAN-to-TSN Ethernet gateway
11:30 - 12:00	Dr. Heikki Saha (TKE)	Automated workflow for generation of CANopen system monitoring graphical user interface (GUI)
12:00 - 12:30	Dr. Christopher Quigley (Warwick)	Benchmarking of CAN systems using the physical layer – car, truck, and, marine case studies
12:30 - 14:00		*Lunch break*
Session VII: Security **Chairperson: Torsten Gedenk (Emotas)**		
14:00 - 14:30	Thilo Schumann (CiA)	Embedded security recap
14:30 - 15:00	Prof. Dr. Axel Sikora (Hochschule Offenburg), Georg Olma (NXP), Olaf Pfeiffer (Emsa)	Achieving multi-level CAN (FD) security by complementing available technologies
15:00 - 15:30	Vivin Richards, Allimuthu Elavarasu (Infineon)	CAN XL made secure
15:30 - 16:00		*Coffee break*
Session VIII: CAN XL higher layers **Chairperson: Dr. Arthur Mutter (Robert Bosch)**		
16:00 - 16:30	Peter Decker (Vector)	IP concepts on CAN XL
16:30 - 17:00	Holger Zeltwanger (CiA)	Multi-PDU concept for heterogeneous backbone networks

For details regarding sponsorship, please contact CiA office:
Phone: +49-911-928819-22 • email: conferences@can-cia.org

Sponsors

emotas embedded communication
GEMAC
PEAK System
VECTOR