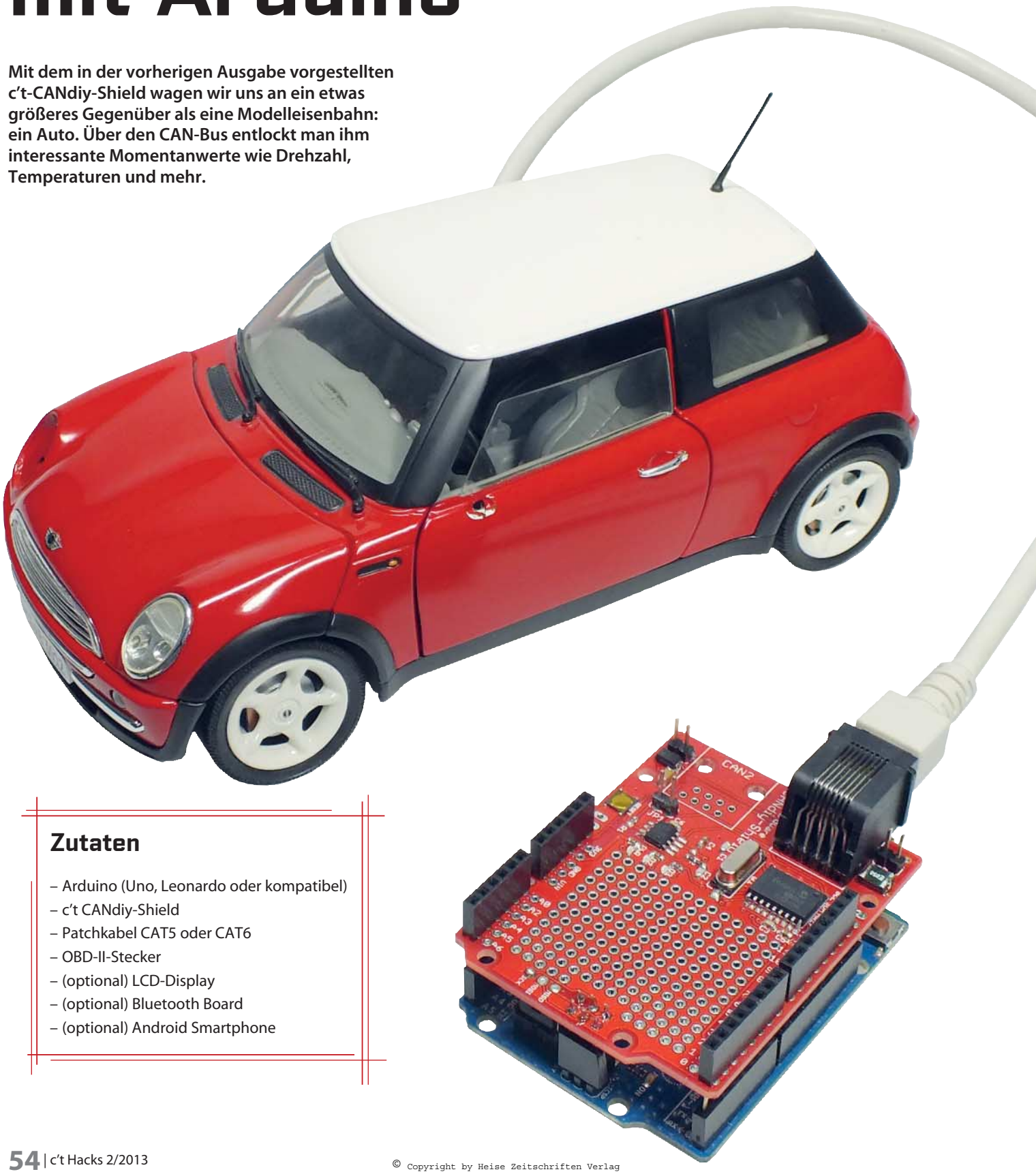
Jörg Pleumann

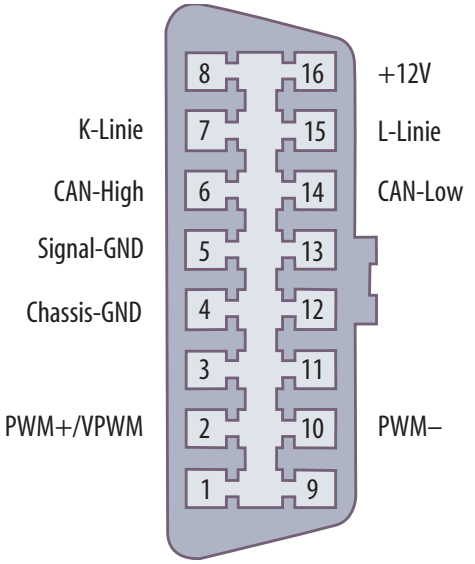
Fahrzeugdiagnose mit Arduino

Mit dem in der vorherigen Ausgabe vorgestellten c't-CANdiy-Shield wagen wir uns an ein etwas größeres Gegenüber als eine Modelleisenbahn: ein Auto. Über den CAN-Bus entlockt man ihm interessante Momentanwerte wie Drehzahl, Temperaturen und mehr.



Zutaten

- Arduino (Uno, Leonardo oder kompatibel)
- c't CANdiy-Shield
- Patchkabel CAT5 oder CAT6
- OBD-II-Stecker
- (optional) LCD-Display
- (optional) Bluetooth Board
- (optional) Android Smartphone



Die OBD-Schnittstelle und ihre Belegung im Auto: Oft kann man schon an den Kontakten sehen, welche Pins wirklich beschaltet sind.

Seit seiner Entwicklung durch Bosch in den frühen 1980er Jahren hat sich das Controller Area Network, kurz CAN, für die Vernetzung der vielfältigen Bordelektronik in Kraftfahrzeugen durchgesetzt. Davon gibt es weit mehr, als man gemeinhin denkt: In einem aktuellen Mittelklassewagen stecken mehrere Dutzend sogenannte Steuergeräte, also Subsysteme, die mit Mikrocontrollern und/oder Sensoren bestückt sind.

Dabei ist es nur logisch, dass der CAN-Bus nicht nur innerhalb des Fahrzeugs eingesetzt wird, sondern auch als Schnittstelle zur Außenwelt dient. Praktisch alle modernen Fahrzeuge verfügen über eine standardisierte Schnittstelle zur On-Board Diagnose (OBD, aktuell in der Version 2, üblicherweise als OBD-II bezeichnet), die den Werkstätten Zugriff auf Daten des Fahrzeugs bietet. Dabei handelt es sich sowohl um Abgas- und Fehlerdaten als auch um Live-Daten des aktuellen Zustands wie Geschwindigkeit, Umdrehungszahl und Benzinstand.

Kam es anfangs noch zu einem ziemlichen Wildwuchs an Protokollen, die je nach Hersteller auf der Schnittstelle unterstützt wurden oder nicht, ist für Neufahrzeuge seit 2008 der CAN-Bus Pflicht. Auch viele ältere Fahrzeuge unterstützen ihn bereits. Die

Kurzinfo

Zeitaufwand:
2 Stunden

Kosten:
ab 20 Euro

Programmieren:
Arduino SDK, optional Java/Android SDK

Löten:
Platine teilbestücken, Kabel konfektionieren

Elektronik:
Grundkenntnisse

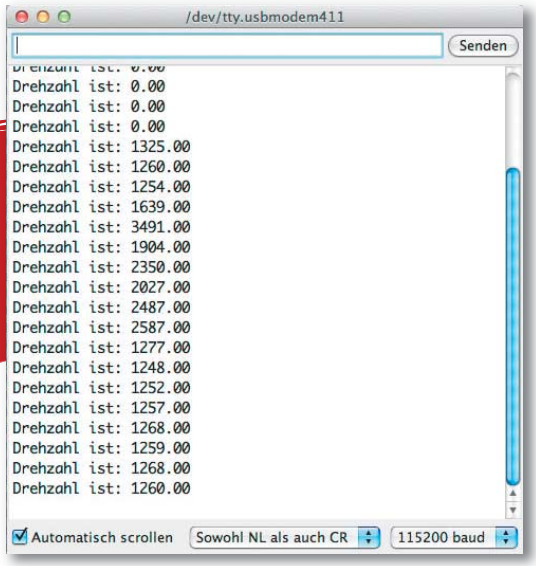
Schwierigkeitsgrad

leicht — schwer

Signal	RJ45	ODB	Farbe
CAN_H	1	6	Orange-Weiß
CAN_L	2	14	Orange
GND	7	5	Braun-Weiß
12 V	8	16	Braun

Vier Pins reichen aus, um über den CAN-Bus im Auto kommunizieren zu können.

Wenn die Farben der Adern im RJ45-Stecker genauso aussehen, dann gilt die Belegung wie in der Tabelle oben.



Der Arduino fragt die Motordrehzahl ab und gibt sie über die serielle Schnittstelle aus.

Listing 1

```
#include <Mechanic.h>
ObdInterface obd;
ObdMessage msg;
void setup() {
  Serial.begin(115200);
  while (!Serial);

  obd.setSlow(false);
  obd.setExtended(false);
  obd.setDebug(false);
  obd.begin();
}
```

Listing 2

```
void loop() {
  boolean supported;

  if (obd.isPidSupported(0x0c, supported)) {
    if (supported) {
      float value;

      if (obd.getPidAsFloat(0x0c, 0.0f, 16383.75f, value)) {
        Serial.print("Drehzahl ist: ");
        Serial.println(value);
      } else {
        Serial.print("Abfrage fehlgeschlagen );
      }
    } else {
      Serial.println("PID wird nicht unterstützt );
    }
  } else {
    Serial.println("PID-Check fehlgeschlagen );
  }

  delay(1000);
}
```

TIPP

Kann es CAN? Wer vor dem Bestellen der Bauteile für den Artikel sichergehen möchte, dass das eigene Auto auch wirklich den CAN auf der OBD-Schnittstelle unterstützt, kann dies relativ einfach herausfinden: Mit einem Multimeter wird der Widerstand zwischen den Pins 6 und 14 der OBD-Buchse gemessen (Motor aus, keine Zündung). Beträgt dieser ungefähr 60 Ohm (zweimal der übliche CAN-Terminierungswiderstand von 120 Ohm parallel), dann ist ziemlich sicher ein CAN-Bus beschaltet.

OBD-Schnittstelle befindet sich üblicherweise im Innenraum des Fahrzeugs, in der Nähe des Fahrersitzes (z. B. unter oder neben der Lenksäule). Hinter einer Abdeckplatte verbirgt sich eine Buchse, die vom Aussehen her ein wenig an den guten alten SCART erinnert – sie ist nämlich recht klobig. Das passende Gegenstück gibt es für wenige Euro auf eBay oder bei den üblichen Verdächtigen unter den Elektronik-Versendern. Damit Sie sich das Zusammensuchen sparen können, hält Watterott sämtliche Komponenten inklusive des Steckers und des CANdiy-Shields vorrätig.

Basteln

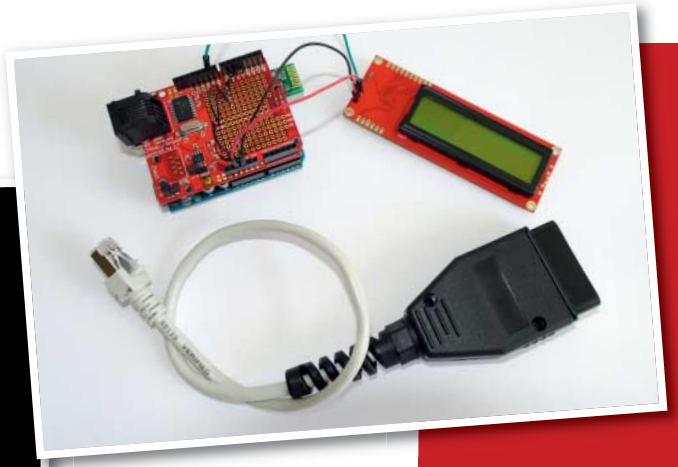
Unser selbst gebautes Diagnosegerät setzt sich zusammen aus einem Arduino mit CANdiy-Shield. Den OBD-Stecker verlöten wir mit einem CAT5- oder

NOTIZ

Nicht jedes Auto gibt alle Daten heraus. Die Öltemperatur wird beispielsweise in vielen Modellen gar nicht gemessen.

Im Auto abfragbare Daten

PID	Bedeutung	Bytes	Minimum	Maximum	Einheit
0x00	unterstützte PIDs 0x01 – 0x20	4	–	–	Bit-codiert
0x04	errechnete Motorlast	1	0	100	Prozent
0x05	Kühlmittel-Temperatur	1	–40	215	Grad C
0x0C	Umdrehungszahl	2	0	16383,75	1/min
0x0D	Geschwindigkeit	1	0	255	km/h
0x11	Gaspedalstellung	1	0	100	Prozent
0x1F	Zeit seit Zündung	2	0	65536	sek
0x20	unterstützte PIDs 0x21 – 0x40	4	–	–	Bit-codiert
0x2F	Treibstoffvorrat	1	0	100	Prozent
0x33	Umgebungsluftdruck	1	0	255	kPa
0x40	Unterstützte PIDs 0x41 – 0x60	4	–	–	Bit-codiert
0x42	Batteriespannung	2	0	65,535	V
0x46	Umgebungstemperatur	1	–40	215	Grad C
0x51	Treibstofftyp	1	1	16	–
0x5C	Öltemperatur	1	–40	215	Grad C



Arduino Uno mit CANdiy-Shield und Diagnosekabel

Mit einem LC-Display wird unser Diagnosegerät unabhängig vom PC.

CAT6-Kabel, das wir an einem Ende mit einem Seitenschneider unsanft von seinem RJ45-Stecker befreit haben. Die Belegung von Kabel und Stecker ist im Bild auf Seite 55 zu sehen. Alternativ kann auch das CAN Shield von Sparkfun mit entsprechendem Kabel verwendet werden. Auf dem CANdiy Shield werden die drei vorhandenen Jumper wie folgt konfiguriert: J3 wird geschlossen. Das sorgt für eine gemeinsame Masse von Arduino und CAN-Bus. J2 wird ebenfalls geschlossen. Der Arduino erhält damit seine Betriebsspannung (12 V) vom CAN-Bus. J1 bleibt offen: Der CAN im Fahrzeug ist bereits mit 120 Ohm terminiert. Damit ist die Hardware bereits fertig. Sie sollte im Großen und Ganzen der Abbildung rechts oben entsprechen. Gehen wir also zum Hacking über.

Software

Um dem Auto seine Diagnosedaten zu entlocken, nutzen wir eine kleine Bibliothek namens „Mechanic“, die ihre Verwandtschaft mit der Railuino-Bibliothek aus dem letzten Heft nicht verleugnen kann: Eine Klasse `ObdMessage` repräsentiert eine Diagnosenachricht, eine weitere Klasse `ObdInterface` implementiert die Kommunikation mit der Diagnoseschnitt-

Das virtuelle Armaturenbrett auf dem Nexus 7 erhält seine Daten via Bluetooth.

Fertiges

Unterstützt Ihr Fahrzeug aufgrund des Alters noch keinen CAN-Bus auf der Diagnoseschnittstelle? Ist Ihnen die Bastelei zu problematisch? Es gibt auch fertige Alternativen, die der c’t-Artikel „Bordcomputer Handy“ in Ausgabe 22/12 ab Seite 138 zeigt. Dort werden verschiedene Bluetooth-Adapter und Android-Apps zur Anzeige der Daten vorgestellt. Die Preise allein für die Adapter liegen allerdings meist deutlich jenseits dessen, was die hier vorgestellte Lösung kostet – und es macht weniger Spaß.

stelle. Viele Bezeichner sind ähnlich wie im letzten Heft. Komfortmethoden decken auch hier die meisten Anwendungsfälle ab, sodass sich niemand mit Details herumschlagen muss, der sich nicht dafür interessiert. Die ZIP-Datei mit der Bibliothek wird – wie üblich – in einem eigenen Ordner unterhalb des Ordners „Libraries“ von Arduino ausgepackt. Hier bietet sich der Name „Mechanic“ an.

Die Verwendung der Bibliothek lässt sich am besten anhand von Listing 1 erläutern. Zunächst wird die Bibliothek über das entsprechende `#include` eingebunden. Innerhalb von `setup()` initialisieren wir neben der seriellen Schnittstelle (für Ausgaben und Debugging) auch den CAN-Controller. Für Letzteren können verschiedene Parameter festgelegt werden, etwa die CAN-Übertragungsgeschwindigkeit mit 250 kbps oder 500 kbps, die Länge der CAN-IDs (11 oder 29 Bit) sowie das Loggen sämtlicher Nachrichten auf der Arduino-Konsole (hilfreich zum Debugging).

Die ersten beiden Parameter sind je nach Fahrzeug unterschiedlich. Sie lassen sich aber einfach durch Ausprobieren herausfinden: Wenn keine Antworten kommen, stimmen normalerweise die Einstellungen nicht. Für alle Parameter gilt: Der Standardwert ist `false`, das heißt, wer nichts einstellt, verwendet automatisch 500 kbps, kurze IDs und kein Debugging. Durch einen Aufruf von `begin()` wird die OBD-Kommunikation gestartet.

Zum Verständnis der weiteren Schritte lohnt es sich einen Blick in den Kasten „OBD-Kommunikation über CAN“ unten zu werfen, der ein bisschen Theorie

TIPP

Wer mehr Kontrolle über die Kommunikation haben möchte, um auch über den Modus 0x01 hinaus auf die Diagnoseschnittstelle zuzugreifen, findet in den Methoden `sendMessage()`, `receiveMessage()` und `exchangeMessage()` das nötige Handwerkszeug.

OBD-Kommunikation über CAN

Über den CAN-Bus tauschen verschiedene Geräte Daten aus, wozu sie einheitliche Pakete verwendeten. Die Felder des Pakets haben unterschiedliche Funktionen. Die Adresse bezeichnet das Steuergerät, mit dem kommuniziert wird. Das CAN-Paket hat einen festen Data Length Code (DLC) von 8. Die Länge der eigentlichen ODB-Nachricht findet sich im ersten Datenbyte und gibt die Anzahl der folgenden Datenbytes an, die genutzt werden. Anfragen haben (im Rahmen dieses Artikels) normalerweise eine Länge von 2, Antworten von bis zu 6 Bytes, wobei sich in speziellen Fällen die Antworten auch auf mehrere Nachrichten erstrecken können. Das letzte CAN-Datenbyte bleibt ungenutzt.

Die Nachrichten teilen sich in 10 standardisierte Modi auf. Einzelne Hersteller fügen proprietäre Modi hinzu, die dann nur für spezielle Fahrzeuge gültig sind. Für diesen Artikel wollen wir uns auf den Standardmodus 0x01 beschränken, mit dem Momentanwerte ohne Seiteneffekte abgefragt werden können.

Innerhalb von Modus 0x01 sind verschiedene Parameter (z. B. die Drehzahl) erfragbar. Jeder Parameter wird über eine 1 Byte lange Parameter-ID (PID) adressiert und besitzt einen Wertebereich, den man kennen muss, um die bis zu 4 Datenbytes der Antwort in einen darstellbaren Wert umrechnen zu können. Nicht jedes Fahrzeug muss jeden Parameter unterstützen. Einige Parameter ergeben nur bei bestimmten Motortypen Sinn. Es lohnt sich also vorher nachzufragen, ob ein Parameter vom Fahrzeug unterstützt wird.

Teil der Bibliothek ist der Beispiel-Sketch „SupportedParameters“, der die unterstützten Parameter-IDs in einer Tabelle anzeigt. Die Ermittlung kann ein paar Sekunden dauern.

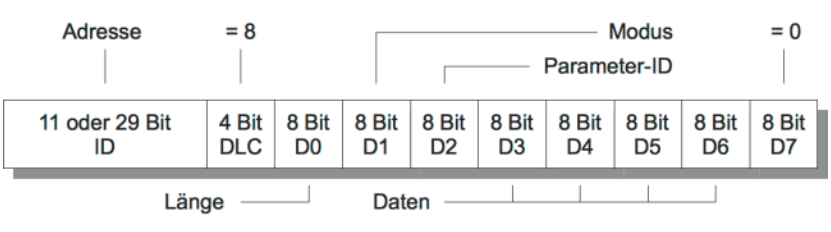
Die folgende Tabelle enthält eine Auswahl von interessanten PIDs, die für den Modus 0x01 definiert sind.

Beim Austausch einer Nachricht passiert Folgendes:

1. Das Diagnosegerät sendet eine Anfrage an die Adresse 0x7df, die als Broadcast-Adresse dient. Die Länge der Nachricht ist 2, Modus und Parameter-ID sind gesetzt.
2. Die Antwort kommt von einem Steuergerät mit einer Adresse im Bereich 0x7e8 bis 0x7ef. Sie hat (im Rahmen dieses Artikels) die Länge 3 oder 4.
3. Der Antwortmodus entspricht dem Anfragemodus plus 0x40. Die Parameter-ID bleibt gleich. Die verbleibenden Bytes enthalten den gewünschten Wert.

Wenn man einmal weiß, welches Steuergerät eine bestimmte Anfrage beantwortet, kann man die gleiche Anfrage künftig direkt an dessen Adresse minus 0x08 senden, was die Mechanic-Bibliothek aber bislang nicht tut.

Ein ODB-Paket auf dem CAN-Bus



BEISPIEL-SKETCHES

Im Sketch-Archiv (Download siehe Link) befinden sich folgende Beispiele:

- VehicleInfo
- SupportedParameters
- CheckAndPrint
- DumpToConsole
- TwoLineLcdDisplay
- AndroidViaBluetooth
- ParametersFromTable
- SelfTest
- Sniffer

zu OBD2-Kommunikation liefert (noch ergiebiger sind die entsprechenden Einträge der englischsprachigen Wikipedia). Mit diesem Wissen ausgestattet können wir in Listing 2 die Drehzahl unter der Parameter-ID 0x0c erfragen. Der Wertebereich geht von 0 bis 16 383,75. Nach dem Start des Sketches spielen wir lässig ein bisschen mit dem Gaspedal.

Sofern bis hierher alles ohne Probleme funktioniert, ist der Rest reine Fleißarbeit. Wenn wir an mehr als ein oder zwei Diagnosecodes interessiert sind und die zugehörigen Werte regelmäßig anzeigen wollen, lohnt es sich, über eine bessere Struktur des Sketches nachzudenken. Mit ein paar Arrays lässt sich die gesamte Tabelle aus Kasten „ODB-Kommunikation am CAN“ in den Quellcode integrieren und auf einfache Weise erweitern. Für jeden Diagnosecode verwalten wir außerdem den letzten bekannten Wert. Die Aufgabe von `loop()` besteht dann nur noch darin, innerhalb einer Schleife jeden Parameter zu erfragen und anzuzeigen. Der Beispiel-Sketch „ParametersFromTable“ tut genau das.

Unser Diagnosegerät soll schöner werden

Wie könnten wir unser Diagnosegerät noch ein wenig aufpeppen? Es gibt natürlich die Möglichkeit, auf das Protofeld des CANdiy-Shields zusätzliche Komponenten zu löten, zum Beispiel LED-Arrays zur Anzeige der Daten in Skalen. Wer möchte, baut ein kleines serielles LCD-Display an und stellt darin die Daten dar (siehe Beispiel-Sketch TwoLineLcdDisplay). Eine weitere Möglichkeit ist eine Aufbereitung auf dem PC. Statt der Arduino-Konsole kann auch eine beliebige andere Applikation die serielle Schnittstelle anzapfen und die Live-Daten in einem Fenster anzeigen oder mitloggen. Die einschlägige Arduino-Literatur enthält dazu Beispiele für die gängigen Programmiersprachen wie Java, Python etc.

Last, but not least: Wer ein Android Smartphone besitzt, kann – ganz wie im Vorgängerartikel zu Railuino – mit einem Bluetooth-Adapter für unter 10 Euro eine galvanisch perfekt getrennte Verbindung zwischen Arduino und Android herstellen (siehe hierzu den Artikel auf Seite 150 des Hefts). Für die Geräte wird einmalig ein Pairing durchgeführt. Die Android App sucht anschließend nach einem verbundenen Bluetooth-Gerät, welches das Communication Device Class (CDC) Profile unterstützt, und erhält dafür einen InputStream und einen OutputStream. Der geänderte Quellcode für die Arduino-Seite ist trivial – wir schreiben einfach die Werte zeilenweise und Komma-getrennt in ein SoftwareSerial-Objekt, an dem der Bluetooth-Adapter hängt. Die Android-Gegenstelle würde den Rahmen des Artikels sprengen, steht jedoch bei den Downloads (siehe Link unten) sowohl im Quellcode wie auch als lauffähige App bereit. (dab)

TIPP

Wenn die Installation der Bibliothek korrekt durchgeführt wurde, taucht diese nach einem Neustart der Arduino IDE mit ein paar Beispielen im Menü auf. Eines der Beispiele trägt den Namen „SelfTest“ und dient zur Überprüfung der Hardware im „Loopback“-Modus, also ohne angeschlossenes Kabel.

Erfolgsmeldungen

In unserem Forum zum Artikel finden Sie eine Liste von Fahrzeugen, mit denen wir den Hack getestet haben. Wenn Sie mit einem Wagen Erfolg hatten, der noch nicht Teil der Liste ist, mailen Sie uns, mit welchen Einstellungen es bei Ihnen geklappt hat!

Links und Foren
www.ct.de/ch1302054

Das Shield im Praxiseinsatz im Auto