Have you ever wondered what really goes on under the hood of your car? Do you wish you could peek inside the engine-management system and read values from it? Are you annoyed that your dashboard displays a cryptic “check engine” light but gives absolutely no explanation what the problem might be?

You don’t need a $10,000 specialist diagnostic console or even a laptop computer to get access to useful data from your car. This project shows you how to connect an Arduino to the engine-management system in your car and log data to a CSV file on a USB memory stick in real time as you drive around. By reading and storing vehicle data and combining it with GPS values, you can create your very own “black box” flight data recorder for your car to record a complete snapshot of all engine-management parameters at the moment a fault occurs, and even generate graphs of vehicle performance.

Because this project stores everything in a standard CSV file on a memory stick formatted with a normal FAT filesystem, it’s really easy to access the data on your computer and manipulate it in any way you like. When you get home from a trip, you can pull out the memory stick, plug it into a computer, and open it in a spreadsheet or convert it into other formats.

Included in this project is a simple script that converts the raw data into KML, the Google Earth data format, allowing you to create an interactive 3D “fly-around” view of a trip. The screenshot in Figure 15-1 was generated using Google Earth and shows the vehicle speed plotted as the height of the line. You can clearly see the speed of the car varying as it goes around corners and through intersections.

Figure 15-1. Vehicle speed and location plotted in Google Earth
You can also process the data to generate graphs like the one in Figure 15-2 that shows the coolant temperature gradually rising, the car accelerating and decelerating during the trip, and finally coming to a halt at its destination. The engine RPM at each part of the trip is also plotted and you can see how it interacts with vehicle speed.

![Figure 15-2. Coolant temperature, vehicle speed, and engine RPM data recorded using an Arduino](image)

Extracting data from a vehicle’s engine-management system while you drive along might seem like magic, but in recent years it has become much easier thanks to a standard called OBD-II, or On-Board Diagnostics version 2. All cars and light trucks sold in the U.S. since 1996 have been required by law to provide an OBD-II interface that provides access to a variety of real-time and historical operational data about the vehicle. Europe followed in 2001 for petrol vehicles and in 2003/2004 for diesels with the EOBD (European OBD) standard, which is basically just OBD-II with a different name. Other parts of the world also have related legislation, such as ADR79/01 in Australia, which is derived from the OBD-II and EOBD standards.

Because car manufacturers try to standardize their production lines, OBD-II vehicles also found their way into many markets outside the U.S. in 1996. Most Japanese car manufacturers in particular deployed OBD-II in other markets very rapidly, even when not legally required to do so. The result is that when you take your car to an auto mechanic, the first thing they usually do is plug in either a dedicated diagnostic console or a laptop using a special adapter, then run software that interrogates the engine-management system to retrieve stored data, such as a list of faults it has detected since the last service.

In this project, we combine an Arduino Mega with an OBD-II adapter, a GPS module, a USB mass-storage module, an LCD module, and control buttons to create a flexible platform for extracting, logging, and reporting data from your car. You won’t get Formula One–level telemetry with thousands of data points per second, but you’ll certainly get more information than an annoyingly vague “check engine” light!

If you don’t want to build such an ambitious system, you might find that much of the project is still useful because it’s broken down into a number of subsystems as shown in Figure 15-5 that can be used in your own designs. Even if you don’t have an OBD-II–compatible car, you can combine elements, such as the GPS module and memory stick driver, with mechanisms to retrieve data via other means.

Building this project the way we designed it requires an Arduino Mega because we need the extra hardware serial ports provided by the four hardware USART (Universal Synchronous/Asynchronous Receiver/Transmitter) channels in the ATMega1280 CPU. Serial communications can either be implemented in software using careful timing (often called “bit-banging”) to send and receive the data stream at the correct rate, or it can be implemented in hardware using a device that takes care of the low-level data transmission on your behalf. A USART is a hardware device specifically designed to do that job.