November 24, 2014

Raspberry Pi + GPIOs with DS18B20 + Azure + C# = Internet Thermometer! (part 1/2)

In my previous post I explained how you could run .NET code on your Raspberry Pi with the help of the Mono open source project. In the long-run I'd like to use my Raspberry Pi as a web gateway, in between my Arduino sensors and the Azure cloud. So the next step in this journey is to connect the Pi to that Azure cloud. Instead of posting dummy "hello world" data to the cloud, I decided to try to get some data from a sensor, directly connected to the Raspberry Pi. You may be wondering how you can connect a dumb sensor to the Pi, well the answer is **General Purpose Input/Output pins** (or GPIO's for short). Every Raspberry Pi has a bunch of them, exposed as pins on the edge of your board. You can think of the GPIO's as:

- switches which the Pi can turn on/off programmatically (**output**)
- switches which can be turned on/off externally and the corresponding state can be read by the Pi (**input**)

As you know, if the Pi can send signals and receive signals, it can communicate with the outside world, which is exactly the goal of the GPIO's. In this example that outside world will be a temperature sensor, the famous **DS18B20**: a fairly cheap, but very accurate sensor, which can be placed in a bus so you can chain multiple of them together.

In this sample I'll send the output of this sensor (the temperature), to an **Azure Mobile Service**. In real life, when you have many, many sensors posting their data to the cloud, **Azure Event Hubs** is probably a better choice. But for simplicity I'll stick to a Mobile Service for now and we'll come back to Event Hubs in a later post. If you haven't set up your Raspberry Pi so you it can run .NET applications, follow step 1 and 2 from my previous post. If you don't have a DS18B20 sensor, or you just want to learn how use Azure Mobile Services on your Raspberry Pi, you can skip ahead to part 2 of this post (of course you won't have real temperature data to post).

Step 1: connecting the DS18B20 sensor

At this point in time you probably want to pay close attention because if you mess up you could, and probably will, damage your Raspberry Pi (**disclaimer: don't blame me if it happens, I warned you**). The DS18B20 sensor has 3 contacts (or legs), when you hold the sensor in front of you (with the flat side facing towards you, and its legs pointing down, like in the picture above) they are identified as follows:

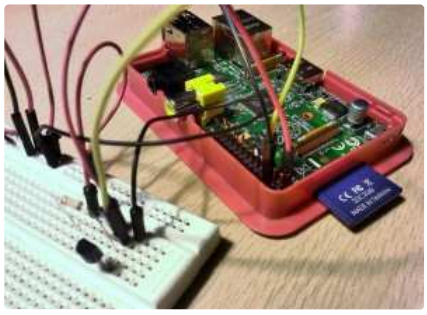
- left leg (pin 1) = negative supply (aka ground)
- right leg (pin 3) = positive supply (aka voltage)
- middle leg (pin 2) = data (aka DQ)

Each of the DS18B20 pins need to be connected to a GPIO of the Raspberry Pi as follows: (check how the GPIO's are numbered on the official Raspberry Pi site)

- DS18B20 pin 1 to Raspberry Pi pin 6 (ground)
- DS18B20 pin 2 to Raspberry Pi pin 7 (data)

- DS18B20 pin 3 to Raspberry Pi pin 1 (voltage)

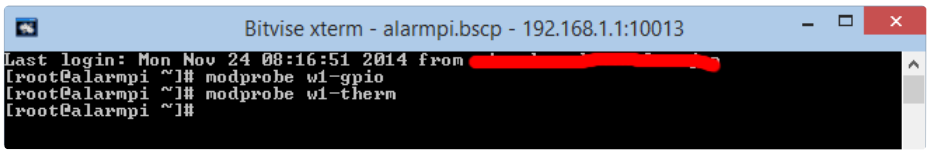
On top of that, we need to have a so called pull-up resistor of 4.7KOhm between the DS18B20 pin 2 and 3. If you want to avoid soldering everything together (which I highly recommend in your test/prototype phase), you can use a so-called breadboard and some jumper wires. The picture on the right shows how I connected my Raspberry Pi to the DS18B20 using such a breadboard. When you make these connections, make sure to power off your Raspberry Pi. When you are ready, give your Pi power and pay close attention to what happens: e.g. if your DS18B20 gets very, very hot, you probably make a wiring mistake and you need to remove power very quickly or you will fry your sensor, the Pi (or both).



Step 2: reading the DS18B20 on your Raspberry Pi

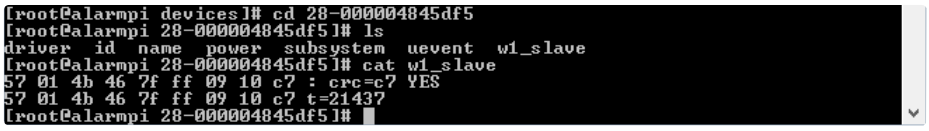
There are a couple of ways you can use to access the GPIO's of the Raspberry Pi: via the memory access, the file system, an HTTP interface etc. There are .NET libraries wrapping the these methods like RaspberryPi.NET and Raspberry.IO. But once again, I'll be using the easiest way to access the DS18B20, using the file system.

To get going, just open a SSH terminal session to your Raspberry Pi (see my previous post for more details), and type the command **modprobe w1-gpio**, this will load the kernel module to access the GPIO's. Then type the command **modprobe w1-therm**, which will add the kernel module to access devices via the One Wire protocol, used by DS18B20 sensor but also others.



Next you can navigate to the directory **/sys/bus/w1/devices** by executing the command **cd /sys/bus/w1/devices**. This directory corresponds with all the devices found on the One Wire bus. If everything is OK you will find one directory with the name **w1_bus_master1** (corresponding with the One Wire bus itself), and a directory with the name **28- and some more numbers**. This last directory will correspond with your DS18B20 sensor's unique serial number. If you have very good eyes you can find it printed on the sensor itself. If you have more sensors on the bus, you'll find more 28-* directories over there.

When you navigate into the 28-* directory, you'll find a file with the name **w1_slave**. When accessing this file (it's just a text file), the actual sensor will be probed to fetch the latest data. With the **cat w1_slave** command, you can display the contents of that file.



The interesting part is the value behind **t=**. That value is the temperature in millidegrees Celcius, so if you device it by 1000, you'll get the temperature in degrees. So in my living room, it's 21.437 °C at the time of writing.

Step 3: creating a .NET application to read the temperature

Now we know where to find the sensor data on the file system, writing a .NET **C# Console Application** to access it is pretty easy.

The following code snippet will loop over all directories in **/sys/bus/w1/devices** which are prefixed with **28** (so which correspond with a sensor). For every directory found, the **w1_slave** text file is opened, and the value after **t=** is converted into a double. Finally the Console Application will output a line of text displaying the device ID and the measured temperature.

```
using System;
using System.IO;
using System.Linq;
using System.Text;

namespace testds
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo devicesDir = new DirectoryInfo("/sys/bus/w1/devices");

            foreach (var deviceDir in devicesDir.EnumerateDirectories("28*"))
            {
                var w1slavetext =
                    deviceDir.GetFiles("w1_slave").FirstOrDefault().OpenText().ReadToEnd();
                string temptext =
                    w1slavetext.Split(new string[] { "t=" }, StringSplitOptions.RemoveEmptyEntries)[1];

                double temp = double.Parse(temptext) / 1000;

                Console.WriteLine(string.Format("Device {0} reported temperature {1}C",
                    deviceDir.Name, temp));
            }

        }
    }
}
```

Build to code and copy the assembly (.EXE) to your Raspberry Pi (once again, previous post for details). When you execute the Console Application with the **mono *yourname*.exe** command, you'll see the output. A sample output from my living room is:

```
[root@alarmpi dotnet]# cd /dotnet
[root@alarmpi dotnet]# mono testds.exe
Device 28-000004845df5 reported temperature 21.437C
[root@alarmpi dotnet]#
```

In the second part of this post, we'll finish up this example by building an Azure Mobile Service which will be used by the Raspberry Pi to send data to. So stay tuned for Part 2, and in the meanwhile if you have any questions, remarks, suggestions ... drop a comment, or let me know via Twitter.

Azure, IoT | 6 Comments | Jan

‹ Running ASP.NET on a Raspberry Pi with Mono and OWIN

Raspberry Pi Internet Thermometer with .NET & Azure, part 2/2 ›

6 thoughts on "Raspberry Pi + GPIOs with DS18B20 + Azure + C# = Internet Thermometer! (part 1/2)"

November 25, 2014 at 10:14 am

Ian

Excellent example, I'm going to try recreating it for the code club I help run in school.

The Pi has internal pull up/down resistors which can be set programmatically 😃

Reply

November 25, 2014 at 4:13 pm

Jan

Thanks! And good tip, did not know about the internal resistor.

Reply

November 27, 2014 at 3:01 pm

Raspberry Pi Internet Thermometer with .NET & Azure, part 2/2 / Jan Tielens' Spot

[...] sure to check post 1 of 2, which describes how to connect your Raspberry Pi to a DS18B20 temperature sensor and read the [...]

Reply

January 2, 2015 at 12:32 pm

promikhail

i try to run this code in cycle? but in second iteration throws error:
Sharing violation on path /sys/bus/w1/devices/22-000000217af0/w1_slave

can’t determine the cause, please help

Reply

January 2, 2015 at 12:35 pm

promikhail

localized to this line of code:
var w1slavetext=deviceDir.GetFiles(“w1_slave”).FirstOrDefault().OpenText().ReadToEnd();

but still don’t know cause

Reply

January 2, 2015 at 2:26 pm

promikhail

i found the reason!

with this request is not closed stream. and therefore in the next iteration, the previous stream blocks the file.

you need to force close the stream, something like:

```
var w1slavefiles =
                                device.GetFiles ("w1_slave");
                            var w1slavefirst =
                                w1slavefiles.FirstOrDefault ();
                            var w1slaveopentext =
                                w1slavefirst.OpenText ();
                            var w1slavetext =
                                w1slaveopentext.ReadToEnd ();
                            w1slaveopentext.Close ();
```

Reply

Leave a Reply

Comment

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <em> <i> <q cite=""> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">`

Post Comment

more of Jan

Twitter: @jantielens

LinkedIn: Jan Tielens

recent posts

- Tweet by @nicktrog
- Raspberry Pi Internet Thermometer with .NET & Azure, part 2/2
- Raspberry Pi + GPIOs with DS18B20 + Azure + C# = Internet Thermometer! (part 1/2)
- Running ASP.NET on a Raspberry Pi with Mono and OWIN
- Tweet by @therealbanksy
- Tweet by @clemensv
- Hi there, Autumn
- Tweet by @jantielens
- Getting Started with Arduino, as a .NET Developer
- SharePoint Saturday Belgium: SharePoint & Windows 8 Apps

recent comments

- promikhail on Raspberry Pi + GPIOs with DS18B20 + Azure + C# = Internet Thermometer! (part 1/2)
- promikhail on Raspberry Pi + GPIOs with DS18B20 + Azure + C# = Internet Thermometer! (part 1/2)
- promikhail on Raspberry Pi + GPIOs with DS18B20 + Azure + C# = Internet Thermometer! (part 1/2)
- Hossein on Running ASP.NET on a Raspberry Pi with Mono and OWIN
- Jan on Running ASP.NET on a Raspberry Pi with Mono and OWIN

archives

- January 2015
- November 2014
- October 2014
- April 2014
- March 2014

proudly hosted on Azure