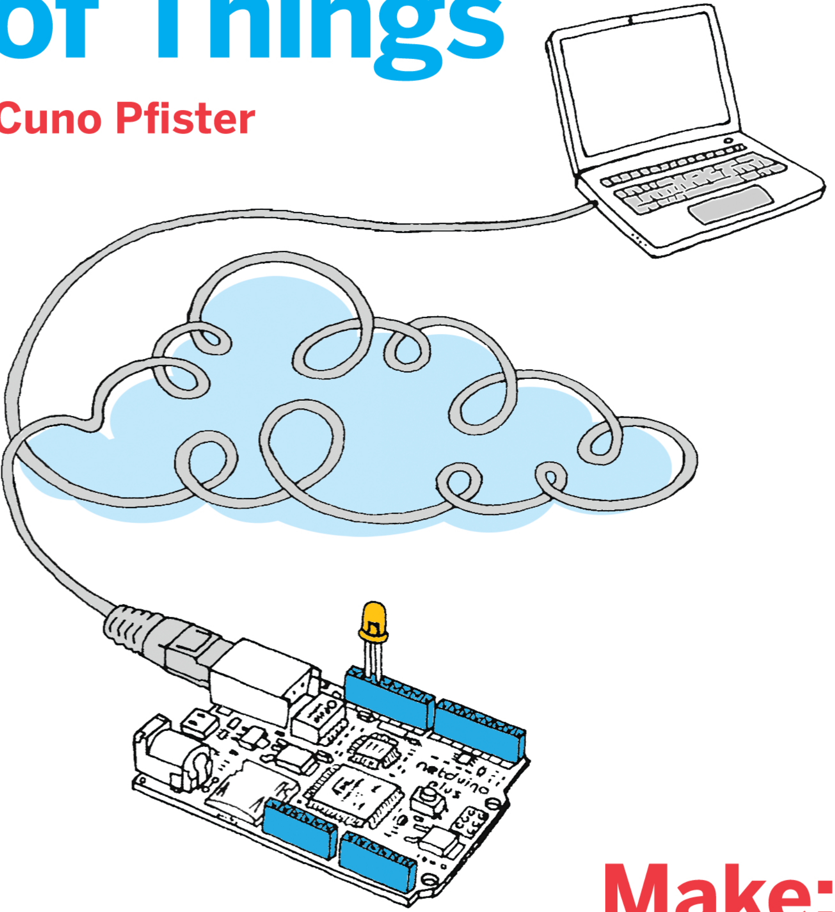
Make: PROJECTS

CONNECTING SENSORS AND MICRO-CONTROLLERS TO THE CLOUD

Getting Started with the Internet of Things

Cuno Pfister



O'REILLY®

Make:
makezine.com

Getting Started with the Internet of Things

Cuno Pfister

O'REILLY®

BEIJING • CAMBRIDGE • FARNHAM • KÖLN • SEBASTOPOL • TOKYO

Getting Started with the Internet of Things

by Cuno Pfister

Published by O’Reilly Media, Inc.
1005 Gravenstein Highway North, Sebastopol, CA 95472

Print History: May 2011: First Edition.

Editor: Brian Jepson
Production Editor: Jasmine Perez
Copyeditor: Marlowe Shaeffer
Proofreader: Emily Quill
Compositor: Nancy Wolfe Kotary
Indexer: Angela Howard
Illustrations: Marc de Vinck
Cover Designer: Marc de Vinck

ISBN: 978-1-4493-9357-1

[LSI]

Contents

Preface

One of the most fascinating trends today is the emergence of low-cost *microcontrollers* that are sufficiently powerful to connect to the Internet. They are the key to the *Internet of Things*, where all kinds of devices become the Internet's interface to the physical world.

Traditionally, programming such tiny *embedded* devices required completely different platforms and tools than those most programmers were used to. Fortunately, some microcontrollers are now capable of supporting modern software platforms like .NET, or at least useful subsets of .NET. This allows you to use the same programming language (C#) and the same development environment (Visual Studio) when creating programs for small embedded devices, smartphones, PCs, enterprise servers, and even cloud services.

So what should you know in order to get started? This book gives one possible answer to this question. It is a *Getting Started* book, so it is neither an extensive collection of recipes (or design patterns for that matter), nor a reference manual, nor a textbook that compares different approaches, use cases, etc. Instead, its approach is "less is more," helping you to start writing Internet of Things applications with minimal hassle.

The Platforms

The *.NET Micro Framework* (NETMF) provides Internet connectivity, is simple and open source (Apache license), has hardware available from several vendors, and benefits from the huge .NET ecosystem and available know-how. Also, you can choose between Visual Studio (including the free Express Edition) on Windows, and the open source Mono toolchain on Linux and Mac OS X.

There is an active community for NETMF at *http://www.netmf.com/Home.aspx*. The project itself is hosted at *http://netmf.codeplex.com/*.

Netduino Plus (*http://www.netduino.com/netduinoplus*) is an inexpensive NETMF board from *Secret Labs* (*http://www.secretlabs.com*). This board makes Ethernet networking available with a price tag of less than $60. It has the following characteristics:

» A 48 MHz Atmel SAM7 microcontroller with 128 KB RAM and 512 KB Flash memory

» USB, Ethernet, and 20 digital I/O pins (six of which can be configured optionally for analog input)

» Micro SD card support

» Onboard LED and pushbutton

» Form factor of the Arduino (*http://www.arduino.cc/*); many Arduino *shields* (add-on boards) can be used

» .NET Micro Framework preprogrammed into Flash memory

» All software and hardware is open source

There is an active community for the Netduino Plus (and NETMF) at *http://forums.netduino.com/*. All the examples in this book use the Netduino Plus.

How This Book Is Organized

The book consists of three parts:

» Part I, Introduction

The first part tells you how to set up the development environment and write and run a "Hello World" program. It shows how to write to output ports (for triggering so-called *actuators* such as LED lights or motors) and how to read from input ports (for *sensors*). It then introduces the most essential concepts of the Internet of Things: HTTP and the division of labor between clients and servers. In the Internet of Things, devices are programmed as clients if you want them to push sensor data to some service; they are programmed as servers if you want to enable remote control of the device over the Web.

» Part II, Device as HTTP Client

The second part focuses on examples that send HTTP requests to some services—e.g., to push new sensor measurements to the Pachube service (*http://www.pachube.com*) for storage and presentation.

» Part III, Device as HTTP Server

The third part focuses on examples that handle incoming HTTP requests. Such a request may return a fresh measurement from a sensor, or may trigger an actuator. A suitable server-side library is provided in order to make it easier than ever to program a small device as a server.

» Appendix A, Test Server

This contains a simple test server that comes in handy for testing and debugging client programs.

» Appendix B, .NET Classes Used in the Examples

This shows the .NET classes that are needed to implement all examples, and the namespaces and assemblies that contain them.

» Appendix C, Gsiot.Server Library

This summarizes the interface of the helper library `Gsiot.Server` that we use in Part III.

Who This Book Is For

This book is intended for anyone with at least basic programming skills in an object-oriented language, as well as an interest in sensors, micro-controllers, and web technologies. The book's target audience consists of the following groups:

» Artists and designers

You need a prototyping platform that supports Internet connectivity, either to create applications made up of multiple communicating devices, or to integrate the World Wide Web into a project in some way. You want to turn your ideas into reality quickly, and you value tools that help you get the job done. Perhaps you have experience with the popular 8-bit Arduino platform (*http://www.arduino.cc/*), and might even be able to reuse some of your add-on hardware (such as shields and *breakout boards*) originally designed for Arduino.

» Students and hobbyists

  You want your programs to interact with the physical world, using mainstream tools. You are interested in development boards, such as the Netduino Plus, that do not cost an arm and a leg.

» Software developers or their managers

  You need to integrate embedded devices with web services and want to learn the basics quickly. You want to build up an intuition that ranges from overall system architecture to real code. Depending on your prior platform investments, you may be able to use the examples in this book as a starting point for feasibility studies, prototyping, or product development. If you already know .NET, C#, and Visual Studio, you can use the same programming language and tools that you are already familiar with, including the Visual Studio debugger.

  To remain flexible, you want to choose between different boards from different vendors, allowing you to move from inexpensive prototypes to final products without having to change the software platform. To further increase vendor independence, you probably want to use open source platforms, both for hardware and software. To minimize costs, you are interested in a platform that does not require the payment of target royalties, i.e., per-device license costs.

If your background is in the programming of PCs or even more powerful computers, a fair warning: embedded programming for low-cost devices means working with very limited resources. This is in shocking contrast with the World Wide Web, where technologies usually seem to be created with utmost inefficiency as a goal. Embedded programming requires more careful consideration of how resources are used than what is needed for PCs or servers. Embedded platforms only provide small subsets of the functionality of their larger cousins, which may require some inventiveness and work where a desired feature is not available directly. This can be painful if you feel at home with "the more, the better," but it will be fun and rewarding if you see the allure of "small is beautiful."

What You Need to Get Started

This book focuses on the interaction between embedded devices and other computers on the Internet, using standard web protocols. Its examples mostly use basic sensors and actuators, so it is unnecessary to buy much additional hardware besides an inexpensive computer board. Here is a list of things you need to run all the examples in this book:

» A Netduino Plus board (*http://www.netduino.com/netduinoplus*)

» A micro USB cable (normal male USB-A plug on PC side, male micro USB-B plug on Netduino Plus side), to be used during development and for supplying power

» An Ethernet router with one Ethernet port available for your Netduino Plus

» An Internet connection to your Ethernet router

» An Ethernet cable for the communication between Netduino Plus and the Ethernet router

» A potentiometer with a resistance of about 100 kilohm and through-hole connectors

» A Windows XP/Vista/7 PC, 32 bit or 64 bit, for the free Visual Studio Express 2010 development environment (alternatively, you may use Windows in a virtual machine on Mac OS X or Linux, or you may use the Mono toolchain on Linux or Mac OS X)

NOTE: There are several sources where you can buy the hardware components mentioned above, assuming you already have a router with an Internet connection:

» Maker SHED (*http://www.makershed.com/*)

  » Netduino Plus, part number MKND02
  » Potentiometer, part number JM2118791

» SparkFun (*http://www.sparkfun.com/*)

  » Netduino Plus, part number DEV-10186

- Micro USB cable, part number CAB-10215 (included with Netduinos for a limited time)
- Ethernet cable, part number CAB-08916
- Potentiometer, part number COM-09806

For more sources in the U.S. and in other world regions, please see *http://www.netduino.com/buy/?pn=netduinoplus*.

It is also possible to add further sensors and actuators.

Conventions Used in This Book

The following typographical conventions are used in this book:

- *Italic*

  Indicates new terms, URLs, email addresses, filenames, and file extensions.

- `Constant width`

  Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, data types, statements, and keywords.

- ***`Constant width bold`***

  Shows commands or other text that should be typed literally by the user.

- *`Constant width italic`*

  Shows text that should be replaced with user-supplied values or by values determined by context.

NOTE: This style signifies a tip, suggestion, or general note.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Getting Started with the Internet of Things*, by Cuno Pfister. Copyright 2011 Cuno Pfister, 978-1-4493-9357-1."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://oreilly.com/catalog/0636920013037

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

http://oreilly.com

Acknowledgments

My thanks go to Brian Jepson, Mike Loukides, and Jon Udell, who made it possible to develop this mere idea into an O'Reilly book. It was courageous of them to take on a book that uses a little-known software platform, bets on a hardware platform not in existence at that time, and addresses a field that is only now emerging. Brian not only edited and contributed to the text, he also tried out all examples and worked hard on making it possible to use Mac OS X and Linux as development platforms.

I would like to thank my colleagues at Oberon microsystems for their support during the gestation of this book. Marc Frei and Thomas Amberg particularly deserve credit for helping me with many discussions, feed-back, and useful code snippets. Their experience was invaluable, and I greatly enjoyed learning from them. Marc's deep understanding of REST architecture principles and its implementation for small devices was crucial to me, as was Thomas's insistence on "keeping it simple" and his enthusiasm for maker communities like those of Arduino and Netduino. Both showed amazing patience whenever I misused them as sounding boards and guinea pigs. I could always rely on Beat Heeb for hardware and firmware questions, thanks to his incredible engineering know-how, including his experience porting the .NET Micro Framework to several different processor architectures.

Corey Kosak's feedback made me change the book's structure massively when most of it was already out as a Rough Cut. This was painful, but the book's quality benefited greatly as a result.

I have profited from additional feedback by the following people: Chris Walker, Ben Pirt, Clemens Szyperski, Colin Miller, and Szymon Kobalczyk. I am profoundly grateful because their suggestions definitely improved the book.

The book wouldn't have been possible without the Netduino Plus, and Chris Walker's help in the early days when there were only a handful of prototype boards. Whenever I had a problem, he responded quickly, competently, and constructively. I have no idea when he finds time to sleep.

Last but not least, many thanks go to the team at Microsoft—in particular Lorenzo Tessiore and Colin Miller—for creating the .NET Micro Framework in the first place. Their sheer tenacity to carry on over the years is admirable, especially that they succeeded in turning the platform into a true open source product with no strings attached.

4/The Internet of Things

Now that you have seen how to work with simple sensors and actuators, it is time to take the next step toward an Internet of Things application. In this chapter, I will briefly introduce the Internet of Things, and the related *Web of Things*.

The Internet of Things is a global network of computers, sensors, and actuators connected through Internet protocols.

A most basic example is a PC that communicates over the Internet with a small device, where the device has a sensor attached (e.g., a temperature sensor), as shown in Figure 4-1.

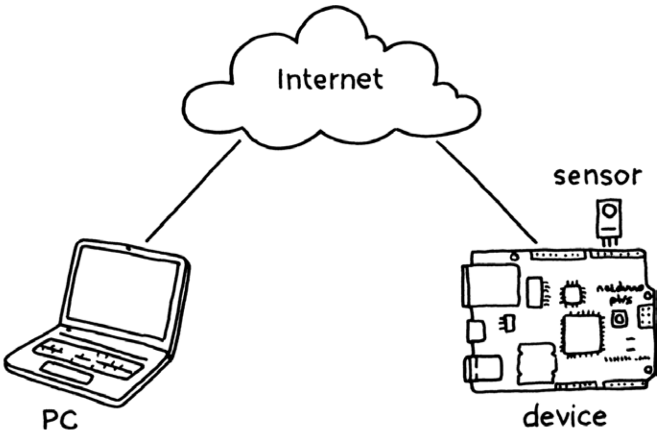


Figure 4-1. A PC and a device connected through the Internet

The *TCP/IP* protocol is the key Internet protocol for such communication scenarios. It enables the transfer of byte streams between two computers in either direction. For example, using the TCP/IP protocol, the device in Figure 4-1 may periodically deliver temperature measurements to a program running on the PC.

HTTP

While it is possible to run any kind of proprietary protocol on top of TCP/IP, there are a few popular and widely supported standard protocols. If you use a standard protocol to deliver your sensor data, you'll be able to work with many more devices and applications than if you developed your own proprietary protocol.

The most important standard protocol by far is the *Hypertext Transfer Protocol* (HTTP), the protocol of the World Wide Web. HTTP describes how a client interacts with a server, by sending *request messages* and receiving *response messages* over TCP/IP, as diagrammed in Figure 4-2.

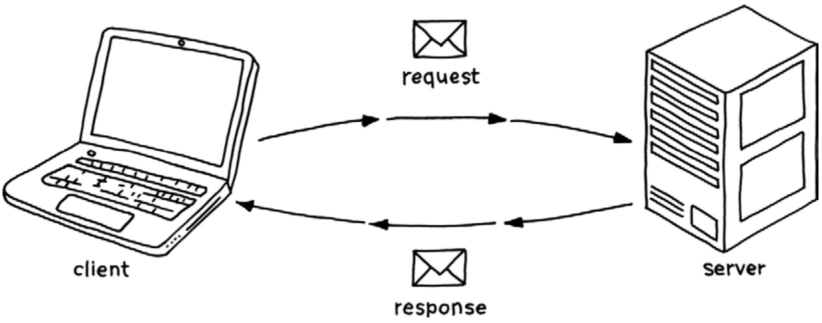


Figure 4-2. Client sends request message, server answers with response message

Web browsers are the most popular HTTP clients, but you can easily write your own clients—and your own servers. If you use a web browser to access a device, the device has the role of a web server, providing a *web service* over the Internet.

A server contains *resources*, which can be anything of interest, e.g., a document (typically an HTML web page), the most current measurement of a sensor, or the configuration of a device. When you design a web service, you need to decide which resources it should expose to the world.

HTTP uses *Uniform Resource Identifiers* (URIs) to tell the server which resource the client wants to read, write, create, or delete. You know URIs from web browsing; they look something like these:[1]

```
http://www.example.com/index.html
http://www.example.com/temperatures
http://www.example.com/temperatures/actual
http://www.example.com:50000/temperatures/actual
http://www.example.com/temperatures?alarm=none
http://www.example.com/temperatures?alarm=high
http://www.example.com/temperatures?alarm=low
http://www.example.com/valve/target
```

A URI indicates the *scheme* (e.g., `http`), the *host* (e.g., `www.example.com`), optionally the *port* (e.g., `50000`), and the *path* (e.g., `/temperatures/actual`) to the resource owned and managed by this host, as shown in Figure 4-3. Optionally, a URI may also contain a *query* (e.g., `alarm=high`) after a `?` character that follows the path.

For the HTTP protocol, port 80 is used by default unless another port is chosen explicitly, perhaps for testing purposes. The path is called *request URI* in HTTP; it denotes the target resource of an HTTP request.

NOTE: URIs that start with a scheme are *absolute URIs*. URIs without a scheme are *relative URIs*. A request URI is a relative URI that starts with `/`. Sometimes you will have to work with absolute URIs and other times with relative URIs, as you will see in the examples.

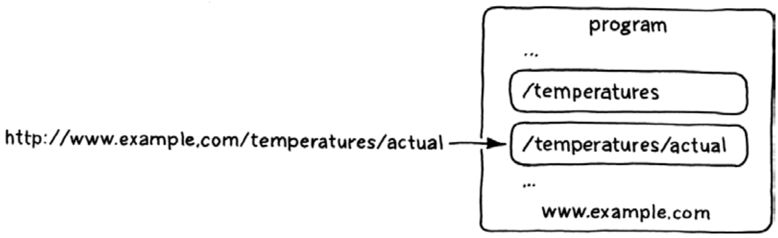


Figure 4-3. URI that addresses a resource managed by a host

[1] These URIs are URLs (*Uniform Resource Locators*) as well. A URL is a URI that also indicates a specific location of a resource, in addition to its identity. I will use the more general term URI throughout this book.

There are several kinds of HTTP requests that a client can send, but the most popular are *GET* for reading a resource, *PUT* for writing to a resource, *POST* for creating a resource, and *DELETE* for deleting a resource. Web browsers mostly issue GET requests, which make up the vast majority of HTTP requests. In a Web of Things application, a GET request to a URI, such as:

```
http://www.example.com/temperatures/actual
```

may return the most recent measurement of a temperature sensor, while a PUT to a URI, such as:

```
http://www.example.com/valve/target
```

may change the setting of an actuator—in this case, a valve. POST requests add sub-resources to a resource, which is similar to putting a file into a directory. For example, a POST of a measurement to the following resource:

```
http://www.example.com/temperatures
```

may create a new resource:

```
http://www.example.com/temperatures(42135)
```

A DELETE request removes a resource—e.g., it may remove the `/temperatures` resource:

```
http://www.example.com/temperatures
```

from the server. (Of course, this would not physically remove the temperature sensor from the hardware.)

PUT requests, POST requests, and GET responses carry *representations* of the addressed resource. The best-known representation is the *Hypertext Markup Language*, better known as HTML. A web browser is an HTTP client that knows how to render HTML pages on the screen. There are other popular representations: PDF, JPEG, XML-based data formats, etc. A web service may support *one or several representations* for a single resource. For example, a temperature measurement may be represented in a plain-text representation, like this:

```
23.5 deg
```

or in an XML representation, like this:

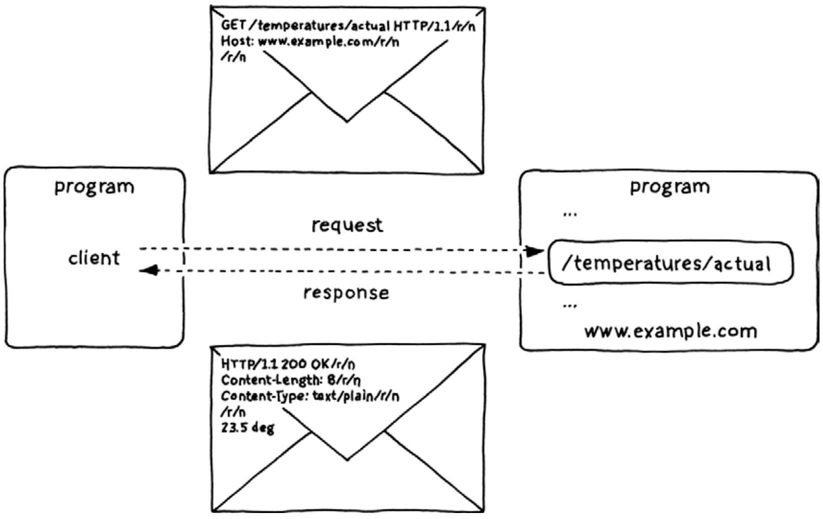
```
<sample>
    <value>23.5</value>
    <unit>deg</unit>
</sample>
```

Some representations are standardized, like HTML, but you may also define your own representations, like those above. Some representations are self-contained documents; others support *links* to other resources. You know the hypertext links from HTML, which use URIs to address other resources. By clicking on a link, you cause the browser to send a GET request to obtain a representation of that resource. This request is sent to the host contained in the link's URI.

Let's look at a complete example of an HTTP request/response interaction (Figure 4-4):

1. This diagram shows a GET request, as it may be sent by a web browser or your own client program. The client requests a representation of the resource's "actual temperature as measured by the temperature sensor," whose URI consists of the host `www.example.com` and the request URI `/temperatures/actual`.

2. The service at host `www.example.com` receives the request, measures the temperature, and returns a response message. In this example, the response indicates success (`200 OK`) and a plain-text representation that is 8 bytes long. The representation is `23.5 deg`.



```
GET /temperatures/actual HTTP/1.1/r/n
Host: www.example.com/r/n
/r/n
```

```
HTTP/1.1 200 OK/r/n
Content-Length: 8/r/n
Content-Type: text/plain/r/n
/r/n
23.5 deg
```

Figure 4-4. HTTP request and response

Even the most complex web interactions consist of such message exchanges. The Web includes several hundred million clients and several hundred thousand servers with their resources, and it produces a torrent of messages that carry resource representations. The technical term for this architecture is *representational state transfer*, or *REST*. For more information on REST, see *RESTful Web Services* by Leonard Richardson and Sam Ruby (O'Reilly).

The focus of *Getting Started with the Internet of Things* is to show how REST and common web standards can be used as the preferred way of creating Internet of Things applications. Such applications are sometimes called Web of Things applications, to emphasize the use of web standards on top of the basic Internet protocols.

The Web of Things consists of RESTful web services that measure or manipulate physical properties.

Thus, the term Web of Things focuses on the application layer and the real-world "things" that are measured or manipulated. The term Internet of Things focuses on the underlying network layers and the technical means for measuring and manipulating the physical environment—i.e., sensors and actuators.

Push Versus Pull

There are four basic ways in which your device may communicate with another computer on the Web:

1. Device is the *client*, pushing data *to* a server

2. Device is the *client*, pulling data *from* a server

3. Device is the *server*, providing data *to* clients

4. Device is the *server*, accepting data *from* clients

These patterns can be visualized as shown in Figure 4-5. A black arrow indicates the direction of a request message and a dotted arrow indicates the direction in which data flows, i.e., in which direction a resource representation is sent.

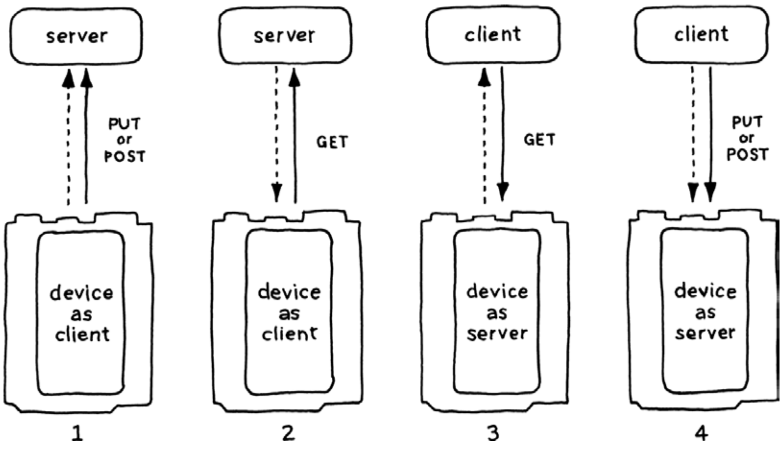


Figure 4-5. Four basic web interaction patterns

In monitoring applications, a device *produces* data, i.e., measurements from its attached sensors. For such applications, the interaction patterns 1 and 3 are suitable: data flows from the device *to* another computer; the device is either client (1) or server (3).

In control applications, a device *consumes* data, i.e., commands from a web browser or other client. For such applications, the interaction patterns 2 and 4 are suitable: data flows to the device *from* another computer; the device is either client (2) or server (4).

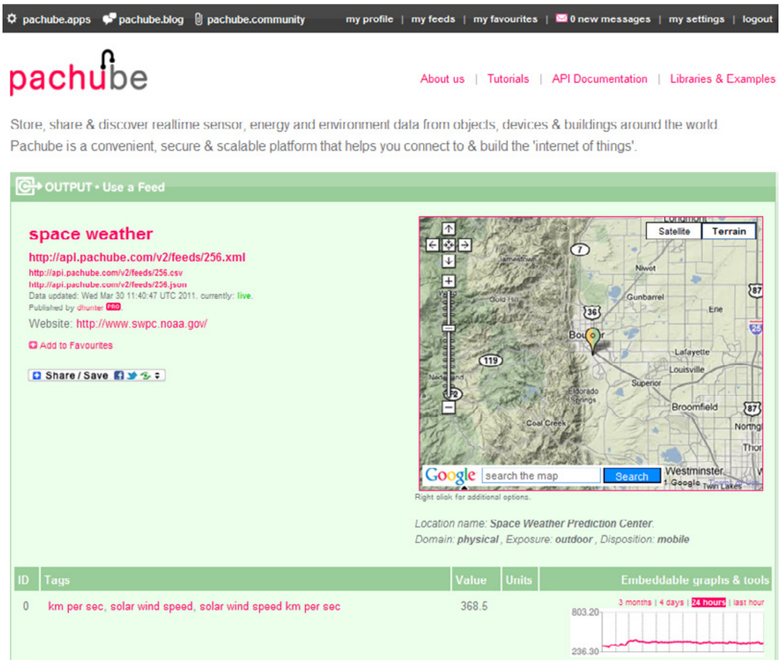
NOTE: A web browser is a client that mainly pulls data from web servers by sending GET requests to them. So you are probably most familiar with interaction pattern 2 because this is the way web browsers work.

In Part II, I will focus on the device as client (i.e., on scenarios 1 and 2). Since in general, a device cannot know in advance when you want to send it a command (e.g., to set up an actuator or to reconfigure a sensor), it makes sense to support devices as servers as well. Therefore, I will discuss scenarios 3 and 4 in Part III. I believe that the potential of the Internet of Things will only be realized if devices can become clients, servers, or both.

5/Pachube

Imagine that your Netduino Plus uses a sensor to take measurements periodically. After each measurement, the Netduino Plus immediately sends the sample to a server for storage and later retrieval. This server effectively provides a *feed* resource to which you publish your data samples. You may already know the concept of feeds from RSS feed readers. A feed entry can be anything of interest, from political news to blog entries to measurements, as in the case of your Netduino Plus. In a way, a feed that contains measurements can be thought of as a news source about the physical world.

For such an example, you need a suitable web service to which your device can send its measurements. Conveniently, there's a free service, Pachube, which does exactly this. It provides web-based interfaces for storing and for accessing feeds, as shown in Figure 5-1.


pachube.apps
pachube.blog
pachube.community
my profile | my feeds | my favourites | 0 new messages | my settings | logout
pachube
About us | Tutorials | API Documentation | Libraries & Examples
OUTPUT • Use a Feed
Add to Favourites
Share / Save
Satellite
Terrain
Google
search the map
Search

Store, share & discover realtime sensor, energy and environment data from objects, devices & buildings around the world
Pachube is a convenient, secure & scalable platform that helps you connect to & build the 'internet of things'.

space weather

http://api.pachube.com/v2/feeds/256.xml
http://api.pachube.com/v2/feeds/256.csv
http://api.pachube.com/v2/feeds/256.json
Data updated: Wed Mar 30 11:40:47 UTC 2011. currently: live.
Published by dhunter PRO.
Website: http://www.swpc.noaa.gov/

Right click for additional options.

Location name: Space Weather Prediction Center.
Domain: physical , Exposure: outdoor , Disposition: mobile

ID	Tags	Value	Units	Embeddable graphs & tools
0	km per sec, solar wind speed, solar wind speed km per sec	368.5		3 months | 4 days | 24 hours | last hour; 803.20; 236.30

Figure 5-1. Example of a Pachube feed

NOTE: The example in Figure 5-1 is a NASA feed. It is atypical insofar as the source of its data is a multimillion dollar space probe—not exactly a low-cost device. Nevertheless, you can use Pachube just as well with your $60 Netduino Plus.

To use Pachube, you need a free account and a feed to which you can send your own data. Follow these steps to create both the account and a first feed:

1. Sign up for a free account at *http://www.pachube.com/signup.*
2. On the "my settings" page (*http://www.pachube.com/users/<your account name>/settings*), you will find the private master API key that you will need later on in your Pachube client programs.

NOTE: Your Netduino Plus programs will send the API key along with every HTTP request to Pachube. The API key tells Pachube that your client program is authorized to add new measurements to your feeds. You'll see how to use this in Chapter 6.

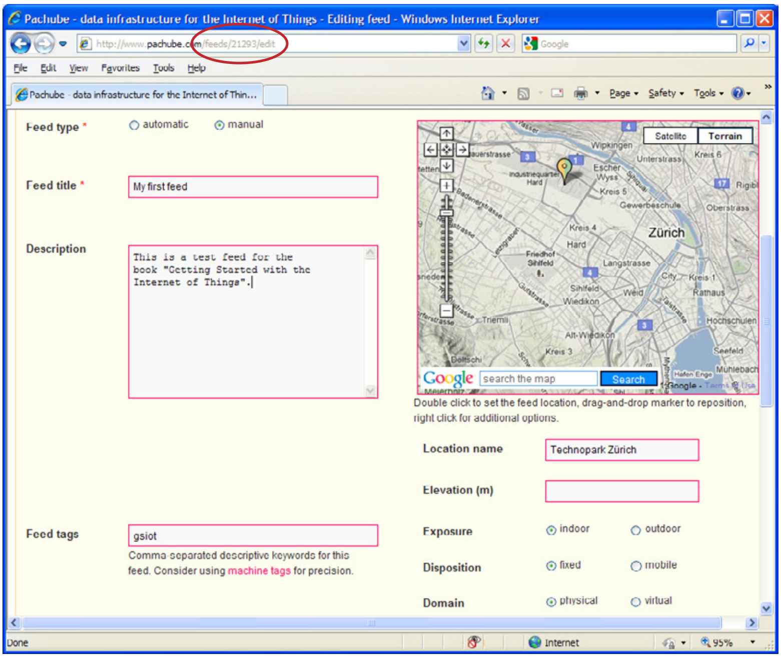
Pachube also supports more advanced *secure sharing keys* as a more secure and fine-grained mechanism where you can, for example, use keys specifically for particular applications, limit the actions possible with these keys, control how long they remain valid, etc.

3. Set up your first feed at *http://www.pachube.com/feeds/new.*
4. For the Feed type, click on "manual".
5. For the Feed title, type in a suitable name, such as "My first feed".
6. For the Feed tags, you could type in "gsiot" so that other readers of this book can find it.
7. For the Exposure, click on "indoor".
8. For the Disposition, click on "fixed".

9. For the Domain, click on "physical".

10. You may enter other information if you want, such as a location name and the location itself (click on the Google map to define the location). If you choose to provide a location, I suggest you pick a well-known public point of interest near you rather than your actual home address.

11. Note the ID of this feed. It is part of the web page URI (circled in Figure 5-2).


Pachube - data infrastructure for the Internet of Things - Editing feed - Windows Internet Explorer
File Edit View Favorites Tools Help
Pachube - data infrastructure for the Internet of Thin...
Page Safety Tools
Satellite Terrain
Zürich
Google search the map Search
Done
Internet
95%

http://www.pachube.com/feeds/21293/edit

Feed type *	automatic	[✓ manual]
Feed title *	My first feed	
Description	This is a test feed for the book "Getting Started with the Internet of Things".	
Feed tags	gsiot	

Comma-separated descriptive keywords for this feed. Consider using machine tags for precision.

Double click to set the feed location, drag-and-drop marker to reposition, right click for additional options.

Location name	Technopark Zürich	
Elevation (m)		
Exposure	[✓ indoor]	outdoor
Disposition	[✓ fixed]	mobile
Domain	[✓ physical]	virtual

Figure 5-2. Editing the properties of a Pachube feed

NOTE: A Pachube feed contains one or several *data streams*; for example, a feed may contain one data stream for every sensor in a building. In the simplest case, a feed has only one data stream—for the measurements of one sensor. In our examples, we will use two data streams: one for voltage values, the other for simple integer numbers.

---

12. Click on "+ Add a new datastream". Enter "voltage" as the ID, enter "Volt" in the Units field, and enter V in the Symbol field. In Type, select "derived SI", which means that this is a unit derived from some other physical units that are considered more basic.

13. Click on "+ Add a new datastream" again. Enter "number" as the ID and leave all other properties as they are.

14. Click on Save Feed.

15. Given your Pachube feed ID, look at the feed's home page by typing in its URI. For example, for the feed `256`, use the URI `http://www.pachube.com/feeds/256`.

Pachube supports a number of URIs for accessing a given feed or data stream. Table 5-1 shows the most important URIs, using the feed ID `256` and the data stream ID `0` as examples.

Table 5-1. Most important URIs for accessing Pachube feeds

Pachube URI	**Description**
http://www.pachube.com/feeds/256	HTML home page of feed `256`.
http://api.pachube.com/v2/feeds/256.json	JSON (*http://www.json.org*) representation of feed `256`, providing maximum, minimum, and current measurement values, plus some metadata that describes the feed. It is also possible to request the data in XML or CSV formats by using the `.xml` or `.csv` suffixes respectively, instead of `.json`.
http://api.pachube.com/v2/feeds/256/datastreams/0.csv?duration=24hours&interval=900	History of measurements in data stream `0` of feed `256`, represented as comma-separated values. Can be imported directly into a spreadsheet. All measurements of the last 24 hours are given, in 15-minute intervals. You can vary the arguments to adjust the time period and the minimum interval between the points.
http://api.pachube.com/v2/feeds/256/datastreams/0.png?duration=24hours&interval=900	Same data as in the above example, but represented as a diagram.

In Chapter 6, you will learn how to send data to your Pachube feed from a program that runs on your Netduino Plus.

JSON

JSON, which stands for *JavaScript Object Notation*, is a textual format for representing arbitrary data. In this respect, it is similar to the often-used XML representation. JSON is popular for web applications since its text is simpler and usually less verbose than equivalent XML text. While JSON is part of the JavaScript language, it is supported by libraries for practically all programming languages today, and has thereby gained "a life of its own." Here is an example of JSON text:

```
{
    "recorded_at" : "2011-03-23T13:29:37Z",
    "max_value" : 25.5,
    "min_value" : 0.0,
    "value" : 1.6
}
```

About the Author

Dr. Cuno Pfister studied computer science at the Swiss Federal Institute of Technology in Zürich (ETH Zürich). His PhD thesis supervisor was Prof. Niklaus Wirth, the designer of the Pascal, Modula-2, and Oberon programming languages. Dr. Pfister is the Managing Director of Oberon microsystems, Inc., which has worked on various projects related to the Internet of Things, from mobile solutions to a large hydropower-plant monitoring system with 10,000 sensors.

Colophon

The cover, heading, and body font is BentonSans, and the code font is Bitstreams Vera Sans Mono.