

## Module 1 sample

# Browsing with HttpClient

1.1 A taste of HttpClient: downloading a web page.....	1
1.2 Reviewing the HttpClient RFCs .....	3
1.3 The HttpClient API.....	14
1.4 HttpClient in action.....	19
1.5 Summary .....	40
Index .....	41

Ah! HTTP.

I can hear the sighs of relief and the shakes of head in an affirmative action, as you contemplate the title of this module. Yes, it's about HTTP—something, as developers, most of us are intimately aware of: the Hypertext Transfer Protocol that drives the thing we lovingly call the Internet. Specifically, it's about the Commons component HttpClient. When you think about it, this is the best possible name for the component; it exactly demonstrates what it does. *HTTP + Client* = HttpClient—a component to help deal with the client side of HTTP.

But wait a minute. I can hear you ask, “Isn't my browser an HTTP client?” Of course, it is. In fact, with the help of HttpClient, you can build your own browser, or embed it into your distributed application, or write a web services client. With the help of HttpClient, you can do anything that you want to do over HTTP (on the client side, anyway).

To start this module, we'll look at the simplest case of using HttpClient, by making it act as a browser and letting it download a web page. We'll then take a step back and take a brief look at the basics of HTTP: specifically, the Requests for Comments (RFCs) that HttpClient implements.<sup>1</sup> This will help you understand the fundamentals of HTTP and better prepare you for learning about HttpClient. We'll then take a whirlwind tour of the HttpClient API, to see its various packages and how they relate to each other. This will get you ready for the examples of using HttpClient that will follow.

## 1.1 A taste of HttpClient: downloading a web page

In this section, we'll look at how to use HttpClient in the simplest possible case of downloading a static web page. This is just a teaser section. More detailed examples will follow after we've discussed the HttpClient API.

Listing 1.1 shows the code required to download a web page using HttpClient. In this listing, HttpClient acts as a browser by requesting Google's home page.

### Listing 1.1 A taste of HttpClient: Downloading Google's home page

```
package com.manning.common.chapter01;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.GetMethod;
```

<sup>1</sup> See module 2, section 2.1.1, for a definition of an RFC.

```
public class HttpClientTeaser {

    public static void main(String args[]) throws Exception {
        HttpClient client = new HttpClient();
        GetMethod method = new GetMethod("http://www.google.com");
        int returnCode = client.executeMethod(method);
        System.err.println(method.getResponseBodyAsString());
        method.releaseConnection();
    }
}
```

HttpClient relies on three other Commons components for it to work. Before you run listing 1.1, make sure that you have the following libraries in your CLASSPATH, in addition to commons-httpclient.jar: commons-lang.jar (discussed in module 7, “Enhancing Java core libraries with collections”), commons-logging.jar, and commons-codec.jar (discussed in module 10, “Codec: encoders and decoders”).

If you run this Java application on a command line, the HTML for Google’s default page will be printed (provided you’re connected to the Internet). A browser would interpret the tags in this HTML and format its display accordingly, but here we were only interested in getting the HTML. We could just as easily have asked for a text document, image, or results of scripts—anything available on the Internet via HTTP—and HttpClient would have fetched it for us.

Using HttpClient starts with creating an instance of the client, as in the listing. In this case, the client that is created uses default settings. These settings include the connection manager used to create underlying connections, the attributes of these connections, the timeouts for giving up delayed connections, and so on.

Next, the GetMethod class is used to specify the document to retrieve using the instance of HttpClient that was just created. The GetMethod class implements the HTTP GET method for retrieving documents of the Internet. (Not surprisingly, you would use the PostMethod class for the HTTP POST method.)

The document to retrieve is specified as the sole argument to the GetMethod constructor. Notice that in this case, the argument specifies an absolute web address, as opposed to a relative web address, to Google’s default page (the argument must be interpreted as `http://www.google.com/index.html` or `http://www.google.com/index.htm` or whatever Google uses as its default page).

The HttpClient instance is used next to execute the method request to the default page of Google. HttpClient’s executeMethod expects an HttpMethod interface implementation as its argument and returns an integer, which represents an HTTP response code for the execution of this request. The GetMethod class extends the abstract class HttpMethodBase, which implements the HttpMethod interface.

Finally, the response from Google is printed on the command line by using the method `getResponseBodyAsString` on the GetMethod instance created earlier. This method evaluates the response received from the execution of a request and returns it by converting the bytes received into their String representation based on the character set specified in the response’s Content-Type header. You could choose to receive this response as a stream by using the `getResponseBodyAsStream` method, or as raw bytes by using the method `getResponseBody`.

As you can see, you need only four lines of code to retrieve a web page from the Internet using HttpClient. However, there is much more to HttpClient than simply retrieving web pages. Before we get into these extras, let’s take a brief look at the RFCs that HttpClient implements. These RFCs define HTTP, and you need to understand them before you can begin to understand HttpClient itself.