Restlet IN ACTION

Jérôme Louvel
Thierry Templier
Thierry Boileau

Foreword by Brian Sletten
brief contents

PART 1 GETTING STARTED ........................................................1

1 ■ Introducing the Restlet Framework  3
2 ■ Beginning a Restlet application  13
3 ■ Deploying a Restlet application  46

PART 2 GETTING READY TO ROLL OUT .............................................79

4 ■ Producing and consuming Restlet representations  81
5 ■ Securing a Restlet application  121
6 ■ Documenting and versioning a Restlet application  151
7 ■ Enhancing a Restlet application with recipes and best practices  165

PART 3 FURTHER USE POSSIBILITIES ......................................201

8 ■ Using Restlet with cloud platforms  203
9 ■ Using Restlet in browsers and mobile devices  242
10 ■ Embracing hypermedia and the Semantic Web  274
11 ■ The future of Restlet  294
Let’s say you headed a team that built a new kind of email service. It was a web application written in Java that made heavy use of servlets, and although initially it worked fine, it didn’t scale well to larger loads when launched in the cloud. Your team had trouble responding to requests to expose the service to clients other than a browser using a SOAP-based web services stack. As a result, your service lost ground to more scalable and interoperable systems; your team was disbanded; and you were let go.

You’re okay—you found a similar position at a better company with a shorter commute. Only one problem: the first thing they want you to do is head a team building...a new kind of email service! You don’t want to repeat the same mistakes this time around.

What were those mistakes, anyway? You’re pretty sure it wasn’t the choice of programming language. You needed the type-safety, the rich built-in library, and the
access to powerful open source libraries that Java developers enjoy. Besides, Java is your core expertise, and it was easy to find competent Java programmers for your team. It wasn’t that servlets didn’t perform as promised, and it wasn’t due to bad programming practices. In fact, it’s hard to pinpoint any specific mistakes.

In the time between jobs, you read about REST and web API designs (see the Resources section near the end of the book, for more information about these). At first you think it’s a step backward, working directly with HTTP, which seems like a lower-level layer, and having no easy access to anything like the session state you were used to with servlets. You’re also understandably leery of buzzwords that can cause religious wars between purists. As you face this new design challenge, a few things about REST begin to click for you:

- _It’s not fair to think of HTTP as a lower-level layer_—HTTP is a mature application-level protocol with support for advanced features like caching and conditional requests, compression, and negotiation between multiple languages and media types; the web depends on it directly for its existence. Although adding these features to your old service would have been prohibitively expensive, a RESTful design, done correctly, would be able to take advantage of them for free.

- _It’s not fair to complain about the absence of session state in HTTP_—One of the things hampering the scalability of your old service was the need to share session state between servers in a cluster, which caused high latency as users’ application states were shunted from server to server. Had the design been more RESTful, application state would have been encoded into the URLs and maintained by the client, leaving the servers dedicated to server-side state.

Maybe you’re not entirely convinced, but at least you’re willing to look further. You want to try a framework for building RESTful services and clients in Java that doesn’t shut out other non-Java clients. You want to be able to deploy these services and clients in a variety of environments, including lightweight ones that don’t involve Java EE. Even if your story isn’t exactly like this (and we hope it isn’t), the punch line is the same: you want Restlet.

An open source project for Java developers, Restlet Framework makes it as easy as possible for you to take advantage of REST and the web in Java. It provides an extensive set of classes and routines that you can call or extend, saving you from writing a great deal of code you would otherwise need to write and allowing you to focus instead on your domain requirements. Restlet helps you use the rich set of HTTP features such as content negotiation, caching, conditional processing, and secure authentication. It can open the doors of the web to you in all its forms—from the classic web to the semantic web, from web services to rich web clients and websites, from the mobile web to cloud computing.

In this chapter we show you some sample Restlet code based on the traditional “Hello World” on both the client and server sides. Then, we explain what the Restlet Framework is and why it’s valuable. By the end of this opening chapter, you’ll have a good overview of the framework, its design, and its capabilities.
**1.1 “Hello World” with Restlet**

In this section coding a “Hello World” program in Restlet will show you how resources are materialized in Restlet, on both the client and server side. We don’t explain all the details right now. Our goal is to show you how basic HTTP clients and servers are developed with Restlet.

At this point it’s important to note that resources is in italics in this chapter to clarify that we are talking about REST resources, the domain concepts identified by URIs and manipulated through HTTP standard method and representations such as HTML documents and JSON data. If you don’t feel familiar enough with REST concepts, we encourage you to read appendix C, which briefly introduces the architectural style of the web.

**1.1.1 Coding a ServerResource subclass**

First, how does Restlet enable you to implement your own resources on the server side? In a few lines of code, listing 1.1 implements one that returns the “hello, world” string. It creates a subclass of the org.restlet.resource.ServerResource class.

Next, you add a represent() Java method, then use a Restlet-specific @Get annotation imported from the same org.restlet.resource package to expose it as a GET method defined in HTTP.

```
import org.restlet.resource.Get;
import org.restlet.resource.ServerResource;
```
public class HelloServerResource extends ServerResource {

    @Get
    public String represent() {
        return "hello, world";
    }
}

In this case the @Get annotation means “Here is the code to be run when a client wants to retrieve a representation of a resource implemented by the HelloServerResource class.” The name of the annotated Java method is flexible for your convenience. We used represent() here, but you can choose any other name as long as it doesn’t override a method of ServerResource or one of its supertypes, such as get().

Note that the signature of an annotated Java method is important and requires you to understand the meaning of the associated HTTP method. An application receiving an HTTP GET request is required to provide a representation of the resource identified in the request by the URI. This is translated into the Java world by a call to a method that returns a value. Here the representation is a simple string, but you’ll see in chapter 4 how to return more complex representations such as XML or JSON documents.

Later in the book, we look at the entire set of existing annotations, how to extend it, and alternative ways to define behavior in ServerResource subclasses that don’t rely on annotations but on overriding regular Java methods.

1.1.2 Running the server

Once written, your Hello resource needs to be exposed and served. The main() method in listing 1.2 creates an HTTP server connector, which is an instance of the Server class available in the org.restlet package. It’s configured with the protocol, the listening socket port, and the target Restlet that will handle the requests—in this case, the HelloServerResource class.

After launching this Java program, you can point your web browser to the http://localhost:8111 URI and get this line of text: “hello, world.”

Listing 1.2 Serving the Hello server resource

```java
import org.restlet.Server;
import org.restlet.data.Protocol;

public class HelloServer {
    public static void main(String[] args) throws Exception {
        Server helloServer = new Server(Protocol.HTTP, 8111,
            HelloServerResource.class);
        helloServer.start();
    }
}
```

Passing the HelloServerResource class to the Server constructor might suggest that a kind of factory is used. That’s true. Each incoming request is handled by a new instance of HelloServerResource. It lets you focus on the behavior and ensure that
each resource instance starts with a clean slate in a way that mirrors the statelessness of REST. As an additional benefit, it simplifies concurrent programming.

By design, a Server instance can’t run without a Protocol. The listening port is optional, because each protocol defines a default port (for example, port number 80 for HTTP, 443 for HTTPS). Creating and starting a Server object is a way to associate a listening port to an object that will handle incoming requests. Such objects can be ServerResource subclasses, as illustrated earlier, but can also be instances of the org.restlet.Restlet class that I present in the next section. Focus your attention on the fact that a Server instance has a set of methods dedicated to its lifecycle—a start() and a stop() method.

Even though this program works fine, many features are lacking to make it a fully featured web application. All URIs, as an example, will be mapped to the same resource class, which isn’t what you would do in a real application. You’ll see in chapter 3 how you can map more specific URIs and how Restlet supports routing and filtering in complex situations such as virtual hosting.

At this point it seems important to clarify what we mean by a web application. In this terminology, as illustrated in figure 1.1, it covers web services in the sense of programmatic interactions over HTTP, static and dynamic websites, and even web clients in the sense of browser-hosted or standalone programs.

It’s common for a web application to mix those types—something can be a web service and a web client at the same time. Let’s now switch to the client side.

### 1.1.3 Using the ClientResource class

As mentioned earlier, Restlet not only is a server-side framework, but also gives you the ability to write clients that consume and manipulate resources exposed on the web. Let’s illustrate this by accessing a sample server. If you run the following code, either in your IDE or your shell, the “hello, world” text will be displayed in the console.

```java
import org.restlet.resource.ClientResource;

public class HelloClient {
    public static void main(String[] args) throws Exception {
        ClientResource helloClientResource =
            new ClientResource("http://localhost:8111/");
    }
}
```

![Figure 1.1](image.png) We use the term web applications to refer to web services, websites, and web clients.
The `ClientResource` class is analogous to a `ServerResource` but located on the client side. This aspect is enforced by the fact that instances of this class are created with the URI of the `resource`. You can consider a `ClientResource` as a local proxy of the remote `resource`. The communication aspects are handled in an intuitive way, with Java methods that are simple transpositions of HTTP methods such as `get()` for GET and `delete()` for DELETE.

Note that the `ClientResource` class allows a series of methods to be invoked on the same target `resource`, automatically follows redirections, and can retry idempotent requests (requests that produce the same result when executed several times) when network errors occur.

Figure 1.2 illustrates the decomposition of `resources` between client side and server side. This separation is based on the uniform interface defined by REST and concretized by HTTP (see appendix C for details), which means the client side can interact with any server-side `resource` on the basis of predefined rules.

### 1.2 Overview of the Restlet Framework

The Restlet Framework, the main topic of this book, has been available since its launch in 2005 as an open source project at www.restlet.org. It’s free, meaning you can use it without charge for in-house, commercial, and open source projects, subject to the relevant license agreement. It’s also mature, meaning it has been under active development since its creation. It’s well supported thanks to an extremely active community; questions asked on the mailing list are usually answered quickly.

In this section we briefly describe this framework, its main features and benefits, its overall design, and the target environments supported.
12.1 **Main benefits of the Restlet API**

The main feature of the Restlet Framework is its Restlet API, a compact and portable Java API located in an `org.restlet` package that embodies major REST concepts such as:

- **Uniform interface**—Standard way to interact with resources via requests and responses
- **Components**—Logical containers for Restlet applications
- **Connectors**—Enable communication between REST components using a protocol
- **Representations**—Expose the state of a REST resource

The Restlet API also abstracts the main features of the HTTP application protocol without requiring deep knowledge of HTTP methods, headers, and statuses:

- **Content negotiation**—Selects the best representation variant based on format (media type), language, or encoding, taking into account client capabilities as well as server preferences
- **Automatic compression**—Reduces the size of representations sent and expands compressed representations received using built-in HTTP capabilities
- **Partial representations**—Retrieves only the part you need via ranged requests, for download resumption or partial-update scenarios
- **Conditional processing**—Executes an HTTP method only if a given condition is met, such as a change in the signature (E-Tag) of the representation you want to retrieve
- **Caching**—Gives hints to clients about the way they should cache and update retrieved representations

The Restlet API is also thread-safe and designed for high concurrency and scalable deployment environments.

As illustrated in figure 1.3, this API has built-in support for routing that is both comprehensive and dynamic (in contrast to the Servlet API, which relies on static XML configuration or annotations for those aspects). It even supports virtual hosting in a way comparable to Apache httpd server but even more flexible. In addition, it comes with a complete web server for both static and dynamic web pages, blurring the distinction between web services and websites which merge into web applications exposing web APIs.

In addition, the Restlet API offers comprehensive support for security based on HTTP features including authentication, authorization (both coarse- and fine-grained), confidentiality (HTTPS, TLS/SSL, Google SDC), and access logging. This built-in feature reduces the complexity of your web projects, removing the need to select and learn a third-party security API for common security needs. But it’s easy to integrate this API with other libraries using standard extensions for JAAS, OAuth, and OpenID, and there is support for authentication schemes such as Azure Shared Key and Amazon Web Services.

In conjunction with a large set of extensions, this single Java API lets you use several protocols such as POP3, SMTP, and pseudoprotocols such as FILE and WAR in a
REST-like way. Numerous representation types are supported for XML, JSON, and many more media types.

Finally, its unifying Java API is unique from many points of view in being usable both for server-side and client-side interactions—or even both at the same time, as with mash-up scenarios. This usability reduces the learning curve and increases productivity.

### 1.2.2 Overall design of the Restlet Framework

The Restlet Framework is composed of a lightweight core and a growing set of extensions. The core is distributed as a single `org.restlet.jar` file with a size of around 500 KB and which contains both the Restlet API and the Restlet Engine. Users typically write programs against the API, indirectly using the engine. But it’s also possible to directly use the engine or extend its behavior by registering new plug-ins, such as connectors for new protocols or helpers for new authentication schemes.

As illustrated in figure 1.4, Restlet user projects can use a growing number of extensions adding standards support (like Atom, JAX-RS, JSON, RDF, and WADL), pluggable connectors (like POP3, SMTP, and FTP), or third-party integrations (like Apache FileUpload, FreeMarker, Jackson, JAXB, Spring, Velocity, and XStream).

As a result, the Restlet Framework offers a comprehensive solution thanks to its numerous extensions while keeping a simple and compact core. For additional details on those extensions, see appendix A, sections A.1–A.3, as well as the project Javadocs. We’ll now describe the target environments supported by this framework.

---

**Figure 1.3  Example of one use of the comprehensive and modular Restlet architecture**
1.2.3 Available editions and deployment targets

The Restlet Framework is Java-based software supporting REST and the all-embracing web. In addition to being usable on both client and server sides, with the HTTP protocol and others like POP3, SMTP, and FTP, the framework is available and supported on several Java-based platforms.

As shown in figure 1.5, those platforms are Java Standard Edition (Java SE) version 5.0 and above, Java Enterprise Edition (Java EE) version 5.0 and above, OSGi version 4.2 and above, Google App Engine (GAE), Android version 1.0 and above, and Google Web Toolkit (GWT) version 1.7 and above. You may not be familiar with all these platforms, so we’ll introduce them next.

Java SE is the classic Java edition that you use when you install a JDK or a JRE. It’s composed of a set of runtime libraries on which Java applications can be built and run. Java
EE builds on top of Java SE and adds a standard set of APIs such as the Servlet, JavaMail, JPA, and EJB APIs. We cover Restlet editions for Java SE and Java EE in parts 1 (chapters 1-3) and 2 (chapters 4-7) of this book. OSGi is a dynamic module system for Java made popular by its support in the Eclipse IDE, by Apache Felix, and by its use as a kernel of application servers such as JBoss.

GAE is a cloud computing service that lets you deploy your application on Google’s own infrastructure, with virtually unlimited CPU power, network bandwidth, and storage capacities. We cover the Restlet edition for GAE in chapter 8.

GWT is an open source technology that lets you develop Rich Internet Applications for web browsers with no additional plug-in. You write and debug using the Java language, but a special compiler produces optimized JavaScript code for execution in a web browser. We cover the Restlet edition for GWT in chapter 9.

Finally there is Android, an open source OS based on Linux and Dalvik, a special virtual machine that can run Java-based programs on mobile devices. It’s supported by Google and a large consortium called the Open Handset Alliance. We cover the Restlet edition for Android in chapter 9 as well.

As promised, this was a brief overview. If you want to get more details at this stage, we encourage you to read sections A.4 and A.5 of appendix A, which presents the various editions of the Restlet Framework, including the matrix of available extensions and the logical versioning scheme of the project.

Now that you’ve had a glance at Restlet programming, you’re ready to move forward to more realistic Restlet application development in chapter 2.

1.3 Summary

In the context of the growing success of RESTful web APIs, the goal of the Restlet Framework is simple: making it as easy as possible for you to take advantage of REST and the web in Java. It provides an object-oriented framework that helps you use the rich set of HTTP features such as content negotiation, caching, conditional processing, and authentication.

This chapter gave you an overview of the Restlet API, a Java API abstracting REST and HTTP concepts and features and enriched by a growing number of Restlet extensions.

The Restlet Framework helps you build first-class RESTful systems, because it was designed from the start to support the REST style. You can use it to create RESTful web services, websites, or client programs, and deploy on a number of platforms including Java SE, Java EE, OSGi, GAE, GWT, and Android, thanks to its multiple editions.

Let’s now continue our exploration of the Restlet Framework with chapter 2 and begin creating Restlet applications.
In a RESTful architecture any component can act, if needed, as both client and server—this is flexible and powerful, but tricky to implement. The Restlet project is a reference implementation with a Java-based API and everything you need to build servers and web clients that integrate with most web and enterprise technologies.

Restlet in Action introduces the Restlet Framework and RESTful web APIs. You’ll see how to easily create and deploy your own web API while learning to consume other web APIs effectively. You’ll learn about designing, securing, versioning, documentation, optimizing, and more on both the server and client side, as well as about cloud computing, mobile Android devices, and Semantic Web applications.

What’s Inside

- Written by the creators of Restlet!
- How to create your own web API
- How to deploy on cloud and mobile platforms
- Focus on Android, Google App Engine, Google Web Toolkit, and OSGi technologies

The book requires a basic knowledge of Java and the web, but no prior exposure to REST or Restlet.

The authors are founders and technical leads of Restlet Inc. and Restlet SAS, Jérôme Louvel is the creator of the Restlet Framework.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/RestletinAction

“Broad, deep, and example-driven.”
—From the Foreword by Brian Sletten Bosatsu Consulting

“Accurate, informative, and extremely useful.”
—Dustin Jenkins, National Research Council of Canada

“A must-have for RESTful web services Java developers.”
—Fabián Mandelbaum NeoDoc SARL

“A broad reach for a perfectly minimalist framework.”
—Tal Liron, Three Crickets

“Thoroughly recommended ... helps the reader come to grips with REST.”
—Dave Pawson Pawson Software Services, Ltd

$49.99 / Can $52.99 [INCLUDING eBook]