



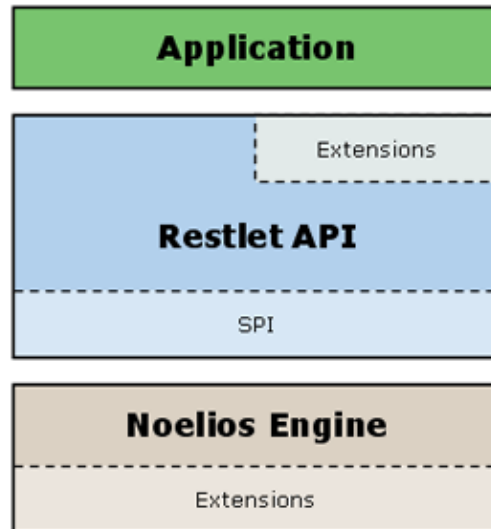
Tutorial

Table of contents

1. [Registering an implementation](#)
2. [Retrieving the content of a Web page](#)
3. [Listening to Web browsers](#)
4. [Overview of a REST architecture](#)
5. [Containers and Virtual Hosts](#)
6. [Serving context resources](#)
7. [Access logging](#)
8. [Displaying error pages](#)
9. [Guarding access to sensitive resources](#)
10. [URI rewriting and redirection](#)
11. [Routers and hierarchical URIs](#)
12. [Conclusion](#)

1. Registering an implementation

The Restlet framework is composed of two main parts. First, there is the "[Restlet API](#)", a neutral API supporting the concepts of REST and a mechanism called Restlet facilitating the handling of REST uniform calls. This API must be supported by a "Restlet Implementation" before it can effectively be used. Multiple implementations could be provided (open source projects or commercial products).



This separation between the API and the implementation is similar to the one between the Servlet API and Web containers like Jetty or Tomcat, or between the JDBC API and actual JDBC drivers. Currently, the "[Noelios Restlet Engine](#)" (NRE) is available and acts as the reference implementation. When you download the Restlet framework, the API and the NRE come bundled together, ready to be used. If you need to use a different implementation just add the implementation JAR file to the classpath and remove the NRE JAR file named `com.noelios.restlet.jar` by default.

The registration is done automatically. See the [JAR specification](#) for details. When an implementation is loaded, it automatically calls

back the `org.restlet.Factory.setInstance()` method.

2. Retrieving the content of a Web page

As we mentioned in the [introduction paper](#), the Restlet framework is at the same time a client and a server framework. For example, NRE can easily work with remote resources using its HTTP client connector. A connector in REST is a software element that enables the communication between components, typically by implementing one side of a network protocol. Here we will get the representation of an existing resource and output it in the JVM console:

```
// Outputting the content of a Web page
Client client = new Client(Protocol.HTTP);
client.get("http://www.restlet.org").getEntity().write(System.out);
```

Note that the example above uses a simplified way to issue calls via the generic `Client` class. A more flexible way is to create a new `Request` object and to ask the `Client` to handle it. The example below illustrates how to set some preferences in your call, like a referrer URI or the languages and media types you prefer to receive as a response:

```
// Prepare the request
Request request = new Request(Method.GET, "http://www.restlet.org");
request.setReferrerRef("http://www.mysite.org");

// Handle it using an HTTP client connector
Client client = new Client(Protocol.HTTP);
Response response = client.handle(request);

// Write the response entity on the console
Representation output = response.getEntity();
output.write(System.out);
```

3. Listening to Web browsers

Now, we want to see how the Restlet framework can listen to client requests and reply to them. We will use one of the NRE HTTP server connectors (supported by Mortbay's Jetty HTTP listener) and return a simple string representation "Hello World!" as plain text. Note that in a more realistic application, we would probably create a separate class extending the Restlet instead of relying on an anonymous inner class. Also, if you need to handle only a specific method, you should use `Handler` as your base class.

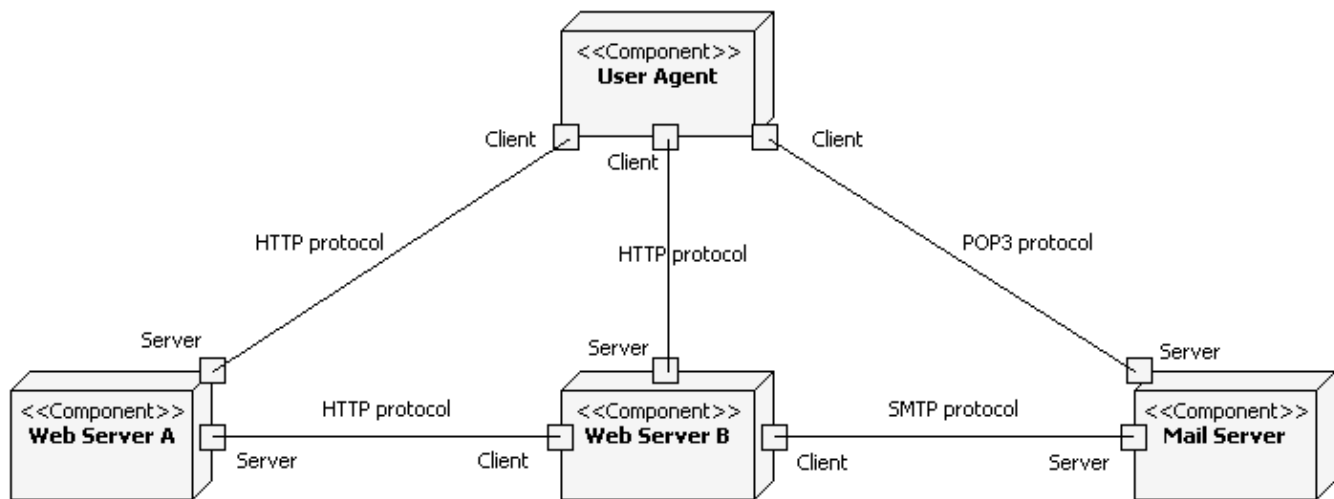
```
// Creating a minimal Restlet returning "Hello World"
Restlet restlet = new Restlet()
{
    public void handle(Request request, Response response)
    {
        response.setEntity("Hello World!", MediaType.TEXT_PLAIN);
    }
};

// Create the HTTP server and listen on port 8182
new Server(Protocol.HTTP, 8182, restlet).start();
```

If you run this code and launch your server, you can open a Web browser and hit the <http://localhost:8182>. Actually, any URI will work, try also <http://localhost:8182/test/tutorial>. Note that if you test your server from a different machine, you need to replace "localhost" by either the IP address of your server or its domain name. The handler that we created is very primitive. We will see later how to take more advantage of the Restlet framework.

4. Overview of a REST architecture

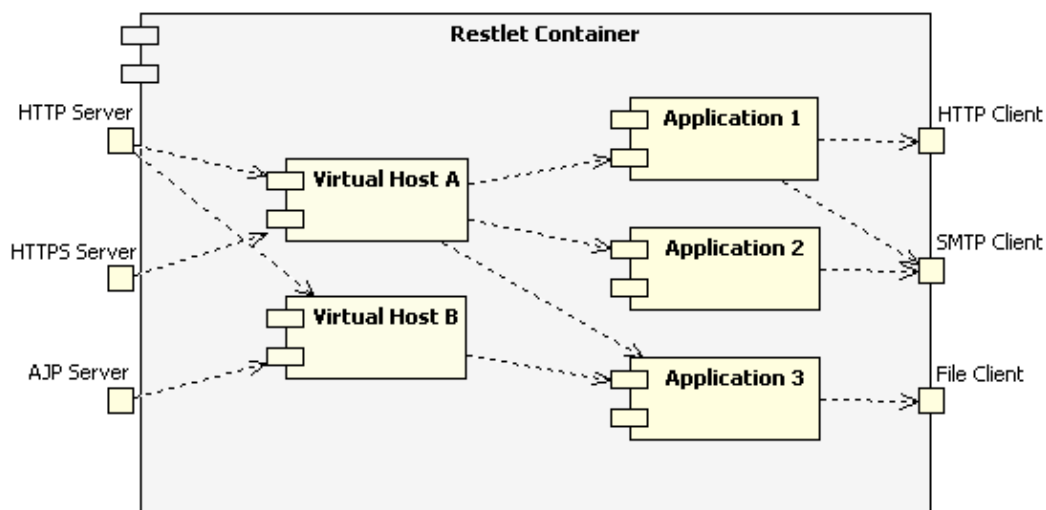
Let's step back a little bit and consider typical web architectures from a REST point of view. In the diagram below, ports represent the connector that enables the communication between components which are represented by the larger boxes. The links represent the particular protocol (HTTP, SMTP, etc.) used for the actual communication.



Note that the same component can have any number of client and server connectors attached to it. Web Server B, for example, has both a server connector to respond to requests from the User Agent component, and client connectors to send requests to the Database Server, the Mail Server and the Web Server A.

5. Containers and Virtual Hosts

In addition to supporting the standard REST software architecture elements as presented before, the Restlet framework also provides a set of classes that greatly simplify the hosting of complex applications within a single JVM. The goal is to provide a RESTful, portable and more flexible alternative to existing Servlet API. In the diagram below, we can see the three types of Restlets that are provided in order to manage these complex cases. Containers are Components managing several Virtual Hosts and Applications. Virtual Hosts support flexible configuration where, for example, the same IP address is shared by several domain names, or where the same domain name is load-balanced across several IP addresses. Finally, we use Applications to manage a set of related Restlets, Resources and Representations. In addition, Applications are ensured to be portable and reconfigurable over different Restlet implementations and with different virtual hosts, and they provide important services such as access logging and status page setting.



In order to illustrate these classes, let's examine a simple example. He we create a Container, then add an HTTP server to it, listening on port 8182. Then we create a simple trace Restlet and attach it to the default VirtualHost of the Container. This default host is catching any request that wasn't already routed to a declared VirtualHost (see the "hosts" property on Container for details). In a later example, we will also introduce the usage of the Application class. Note that for now you don't see any access log displayed in the console.

```
// Create a new Restlet container and add a HTTP server connector to it
```

```

Container container = new Container();
container.getServers().add(Protocol.HTTP, 8182);

// Create a new Restlet that will display some path information.
Restlet restlet = new Restlet()
{
    public void handle(Request request, Response response)
    {
        // Print the requested URI path
        String message = "Resource URI:  " + request.getResourceRef() + '\n' +
                        "Base URI:      " + request.getBaseRef() + '\n' +
                        "Relative path: " + request.getRelativePart() + '\n' +
                        "Query string:  " + request.getResourceRef().getQuery();
        response.setEntity(message, MediaType.TEXT_PLAIN);
    }
};

// Then attach it to the local host
container.getDefaultHost().attach("/trace", restlet);

// Now, let's start the container!
// Note that the HTTP server connector is also automatically started.
container.start();

```

Now let's test it by entering <http://localhost:8182/trace/abc/def?param=123> in a Web browser. Here is the result that you will get:

```

Resource URI:  http://localhost:8182/trace/abc/def?param=123
Base URI:      http://localhost:8182/trace
Relative path: /abc/def
Query string:  param=123

```

6. Serving context resources

Do you have a part of your web application that serves static pages like Javadocs? Well, no need to setup an Apache server just for that, the Noelios Restlet Engine provides a dedicated Restlet called `DirectoryRestlet`. See how simple it is to use it:

```

// Create a container
Container container = new Container();
container.getServers().add(Protocol.HTTP, 8182);
container.getClients().add(Protocol.FILE);

// Create an application
Application application = new Application(container)
{
    public Restlet createRoot()
    {
        return new DirectoryHandler(getContext(), ROOT_URI, "index.html");
    }
};

// Attach the application to the container and start it
container.getDefaultHost().attach("", application);
container.start();

```

Note that no external configuration file is needed. If you want to customize the mapping between file extensions and metadata (media type, language or encoding), you can use the Application's "localService" property.

7. Access logging

Being able to properly log the activity of a Web application is a common requirement. Restlet Applications and Containers have built-in support for logging that knows how to generate Apache-like logs. By taking advantage of the logging facility built in the JDK, the logger can be configured like any standard JDK log to filter messages, reformat them and specify where to send them. Rotation of logs is also supported; see the [java.util.logging](#) package for details.

Note that you can customize the logger name given to the java.util.logging framework by modifying the Application's "logService" property. In order to fully configure the logging, you need to declare a configuration file by setting a system property like:

```
System.setProperty("java.util.logging.config.file", "/your/path/logging.config");
```

For details on the configuration file format, please check the [JDK's LogManager](#) class.

8. Displaying error pages

Another common requirement is the ability to customize the status pages returned when something didn't go as expected during the call handling. Maybe a resource was not found or an acceptable representation isn't available? In this case, or when any unhandled exception must be intercepted, the Application or the Container will automatically provide a default status page for you. This service is associated to the org.restlet.util.LogService class, which is accessible as an Application and Container property called "logService".

In order to customize the default messages, you will simply need to create a subclass of LogService and override the getRepresentation (Status, Request, Response) method. Then just set an instance of you custom service to the appropriate "logService" property.

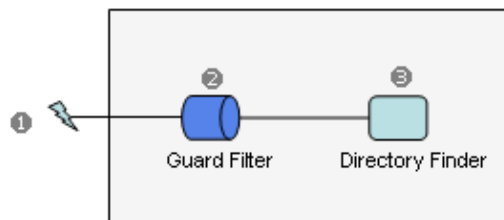
9. Guarding access to sensitive resources

When you need to secure the access to some Restlets, several options are available. The most common way is to rely on cookies to identify clients (or client sessions) and to check a given user ID or session ID against your application state to determine if access should be granted. Restlets natively support cookies via the [Cookie](#) and [CookieSetting](#) objects accessible from a [Call](#).

Also, there is another way based on the standard HTTP authentication mechanism. The Restlet framework currently supports the basic HTTP authentication scheme via the GuardFilter. Filters are specialized Restlets that filter calls passed to them, without modifying the requested resource URI. If you are familiar with the Servlet API, the concept is similar to the [Filter](#) interface. See below how we would modify the previous example to secure the access to the DirectoryHandler:

```
// Create a guard Filter
GuardFilter guard = new GuardFilter(getContext(), true, ChallengeScheme.HTTP_BASIC ,
    "Tutorial", true);
guard.getAuthorizations().put("scott", "tiger");

// Create a directory Restlet able to return a deep hierarchy of Web files
DirectoryHandler directory = new DirectoryHandler(getContext(), ROOT_URI, "index.
html");
guard.setNext(directory);
return guard;
```



Note that the authorization decision is fully customizable via the authorize method. Any mechanism can be used to check whether the given credentials are valid. Here we simply hard-coded the only valid user and password. In order to test this authentication mechanism, let's use the client-side Restlet API:

```

// Prepare the request
Request request = new Request(Method.GET, "http://localhost:8182/");

// Add the client authentication to the call
ChallengeResponse authentication = new ChallengeResponse(ChallengeScheme.HTTP_BASIC,
"scott", "tiger");
request.setChallengeResponse(authentication);

// Ask to the HTTP client connector to handle the call
Client client = new Client(Protocol.HTTP);
Response response = client.handle(request);

if(response.getStatus().isSuccess())
{
    // Output the response entity on the JVM console
    response.getEntity().write(System.out);
}
else if(response.getStatus().equals(Status.CLIENT_ERROR_UNAUTHORIZED))
{
    // Unauthorized access
    System.out.println("Your access was not authorized by the server, check your
credentials");
}
else
{
    // Unexpected status
    System.out.println("An unexpected status was returned: " + response.getStatus());
}

```

You can change the user ID or password sent by this test client in order to check the response returned by the server. Remember to launch the previous Restlet server before starting your client. Note that if you test your server from a different machine, you need to replace "localhost" by either the IP address of your server or its domain name when typing the URI in the browser. The server won't need any adjustment due to the usage of the HostRouter which accepts all types of URI by default.

10. URI rewriting and redirection

One of the advantages of the Restlet framework is the built-in support for [cool URIs](#). Another good description of the importance of proper URI design is given by Jacob Nielsen in his [AlertBox](#). The first tool available is the RedirectRestlet, which allows the rewriting of a cool URI to another URI, followed by an automatic redirection. Several types of redirection are supported, the external redirection via the client/browser, the internal redirection at the container level and the connector redirection for proxy-like behavior. In the example below, we will define a search service for our web site (named "mysite.org") based on Google. The "/search" relative URI identifies the search service, accepting some keywords via the "query" parameter. :

```

// Create a redirect Restlet then attach it to the container
String target = "http://www.google.com/search?q=site:mysite.org+${query('query')}";
return new RedirectRestlet(getContext(), target, RedirectRestlet.
MODE_CLIENT_TEMPORARY);

```

Note that the RedirectRestlet needs two parameters only. The first one defines how the URI rewriting should be done, based on a URI template. This template will be processed by the [StringTemplate](#) class and the URI data model provided by the [RestletCallModel](#) class. The second parameter defines the type of redirection; here we chose the client redirection, for simplicity purpose.

11. Routers and hierarchical URIs

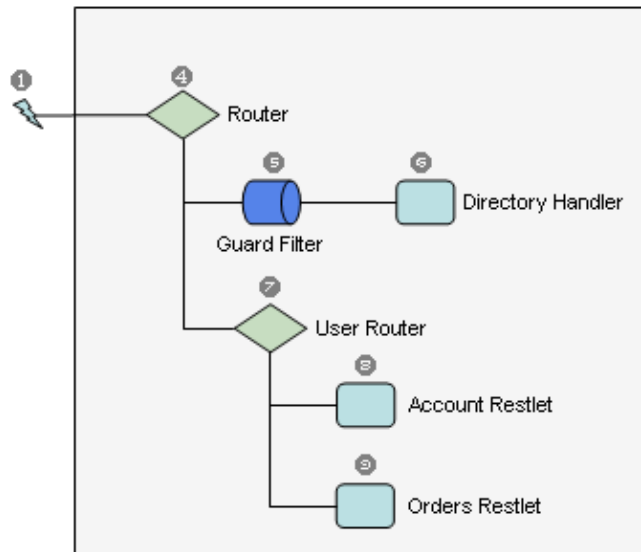
In addition to the RedirectRestlet, we have another tool to manage cool URIs: Routers. They are specialized Restlets that can have other handlers (Restlets, Routers or Filters) attached to them and that can automatically delegate calls based on a URI pattern (a regular expression). Note that Restlet containers are also Routers, so you could directly attach Restlets to "myContainer" using an adequate URI pattern. In general, it is easier to attach Filters followed by a HostRouter to the container. Then you can attach your application Restlets or Routers to this root HostRouter.

Here we want to explain how to handle the following URI patterns:

1. /docs/ to display static files
2. /users/[a-z]+ to display a user account
3. /users/[a-z]+/orders to display the orders of a particular user

The fact that these URIs contain some dynamic parts and that no file extension is used makes it harder to handle them in a typical Web container. Here we will simply rely on a tree of Routers. In the processing flow below, you can see that when a call is received by the container (1), it is first filtered by two Filters (2) and (3), and then it reaches the root Host Router. If the URI starts with "/docs/" then the GuardFilter will get the call (5) and pass it to the DirectoryRestlet (6) if the user credentials are authorized. Instead, if the URI starts with "/users/[a-z]+" (where "[a-z]+" is representing a valid user name) then it is the User Router that gets the call, otherwise an error is returned to the client (404, not found). Please, note that the User Router is directly attached to its parent Host Router and not to the top level container!

When a call reaches the User Router (7) we try to match the remaining part of the URI against the "\$" pattern, meaning the the remaining resource path should be empty. If a match is found, then the Account Restlet is invoked (8). Otherwise, if the "/orders\$" pattern matches, the Orders Restlet is invoked (9), where we can still get the login name from the list of URI segments. Note that again we use the '\$' character at the end of the pattern in order to make sure that we match "/users/foo/orders" but not "/users/foo/orders123".



See the implementation code below. In a real application, you will probably want to create separate subclasses instead of the anonymous ones we use here:

```
// Create a container
Container container = new Container();
container.getServers().add(Protocol.HTTP, 8182);
container.getClients().add(Protocol.FILE);

// Create an application
Application application = new Application(container)
{
    public Restlet createRoot()
    {
        // Create a root Router
        Router router = new Router(getContext());

        // Attach a guard Filter to secure access to the chained directory handler
        GuardFilter guard = new GuardFilter(getContext(), true, ChallengeScheme.
```

```

HTTP_BASIC , "Restlet tutorial", true);
guard.getAuthorizations().put("scott", "tiger");
router.getScorers().add("/docs/", guard);

// Create a directory handler able to return a deep hierarchy of Web files
DirectoryHandler directory = new DirectoryHandler(getContext(), ROOT_URI,
"index.html");
guard.setNext(directory);

// Create the user router
Router user = new Router(getContext());
router.attach("/users/[a-z]+", user);

// Create the account Restlet
Restlet account = new Restlet()
{
    public void handle(Request request, Response response)
    {
        // Print the requested URI path
        String message = "Account of user named: " + request.getBaseRef().
getLastSegment();
        response.setEntity(message, MediaType.TEXT_PLAIN);
    }
};
user.attach("$", account);

// Create the orders Restlet
Restlet orders = new Restlet(getContext())
{
    public void handle(Request request, Response response)
    {
        // Print the user name of the requested orders
        List segments = request.getBaseRef().getSegments();
        String message = "Orders of user named: " + segments.get(segments.size
() - 2);
        response.setEntity(message, MediaType.TEXT_PLAIN);
    }
};
user.attach("/orders$", orders);

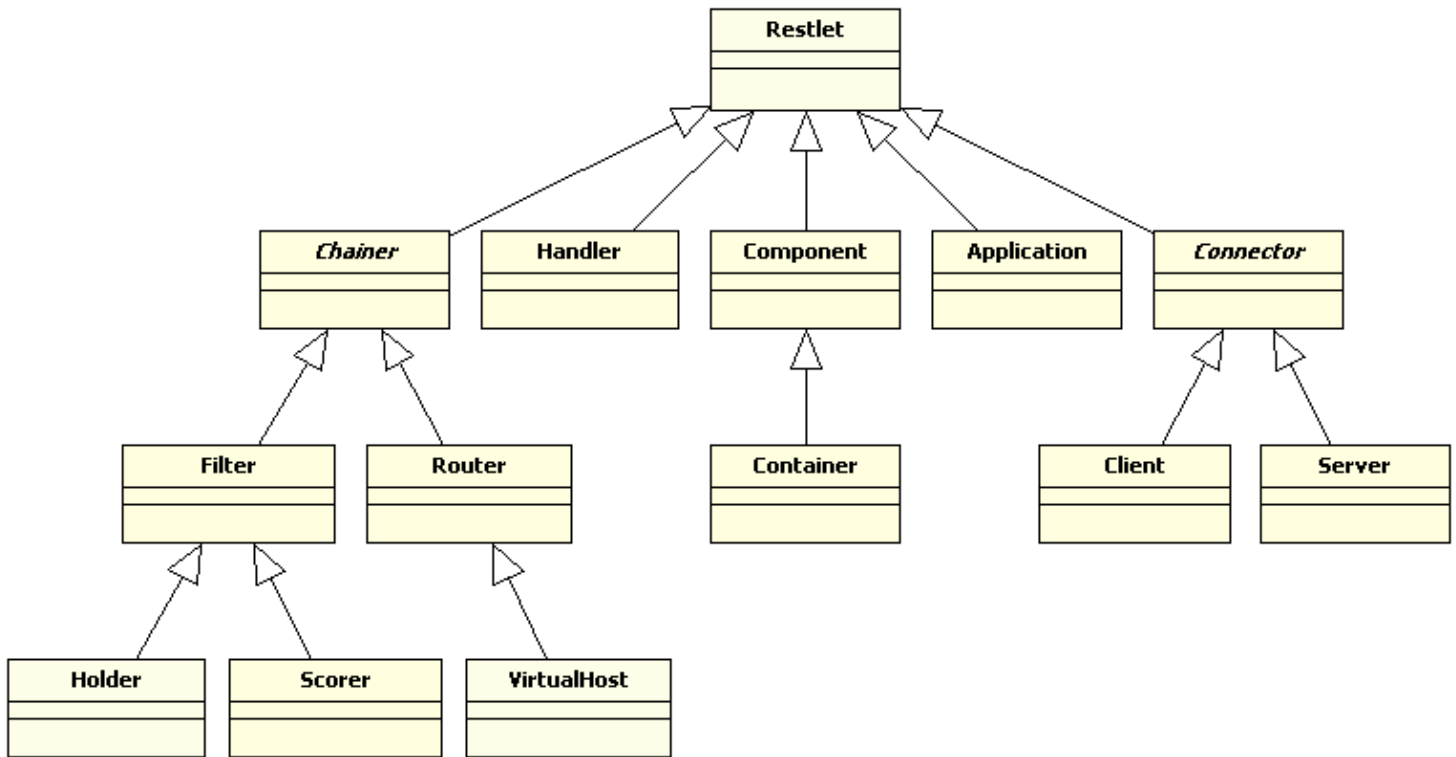
// Return the root router
return router;
}
};

// Attach the application to the container and start it
container.getDefaultHost().attach("", application);
container.start();

```

Conclusion

We already covered many aspects of the framework. Before you move on by yourself, let's take a step back and look at this hierarchy diagram showing the main concepts covered in this tutorial and their relationships.



Beside this tutorial, your best source of information will be the Javadocs available for both the [API](#) and the [NRE](#). You can also post your questions and help others in our [discussion list](#).

[Jérôme Louvel](#) ([blog](#))

Notes

- I encourage you to run the examples. The full source code is available in the latest release.
- Thanks to [Jean-Paul Figer](#), Christian Jensen, Jim Ancona, Roger Munday, John D. Mitchell, Jérôme Bernard and Dave Pawson for the feed-back on this tutorial.

Last modified on 2006-10-29. Copyright © 2005-2006 [Jérôme Louvel](#). Restlet is a registered trademark of [Noelios Consulting](#).