



1 Introduction

In this document, we look at REST, a technology for web services, and see how Rails automatically generates interfaces that are RESTful.

The document assumes the reader has followed the instructions that are given in the documents *Rails HOW-TO: A simple Rails apps: phones* and *Rails HOW-TO: Apache and Basic Authentication*. These documents are available at <http://www.oucs.ox.ac.uk/rails/howtos>

2 The HTTP Protocol

When you submit a request from a web browser to a web server using a URL like:

```
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones
```

there is a communication between your computer and the computer running the web server. This communication uses the *HTTP protocol*.

The HTTP protocol defines many different methods for communicating including GET, PUT, POST and DELETE. If you click a link on a web page, the GET method is used. If a web page provides a form, then, if you click on the form's submit button, the method used to transmit the data of the form to the web server is usually POST (although the form can be configured to use GET). The PUT and DELETE methods are not always implemented by web clients. And some firewalls block their use.

3 REST

In 2000, Roy Fielding submitted a Ph.D. dissertation entitled *Architectural Styles and the Design of Network-based Software Architectures*. He introduced the term *REpresentational State Transfer* (i.e., *REST*), a style for the design of software for distributed systems.

These ideas have been used with the HTTP protocol as a way for programs to communicate across the web with web servers. The URL can be used to indicate the resource you want to manipulate and the HTTP method can be used to indicate what you want to do with that resource.

The HTTP methods can be used as follows:

HTTP method	purpose
GET	get the details of a resource
PUT	overwrite the details of a resource
POST	add a new resource
DELETE	delete a resource

Examples are:

URL	HTTP method	operation	details of the operation
.../phones	GET	<i>index</i>	produce a list of the entries in the phone book
.../phones	POST	<i>create</i>	add a new entry to the phone book
.../phones	DELETE	<i>destroy</i>	destroy the phone book
.../phones/1	GET	<i>show</i>	show entry 1 of the phone book
.../phones/1	PUT	<i>update</i>	overwrite entry 1 of the phone book
.../phones/1	DELETE	<i>destroy</i>	destroy entry 1 of the phone book

Aside: for many people, REST comes as a welcome simplification of web services: it is a relief to the complexity of performing web services using SOAP.

4 Ruby and REST

The URLs and HTTP methods that have been used in previous documents to communicate with a Rails application are essentially RESTful. For example, if we click on the submit button of a form for a new entry of the phone book, it communicates the data using POST to the URL:

```
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones
```

There is just one difference: because of the problems with HTTP's PUT and DELETE methods (mentioned above), Rails permits these communications to be transmitted using POST provided the data has a field called `_method` that is set to either `put` or `delete`.

5 Testing Ruby's RESTfulness

The main idea about providing a web application that supports REST is to make it easy for programs to communicate with that web application. Such programs can be written in any suitable programming language. Examples include: C#, Java, JavaScript, Perl, Python and Ruby.

In order to demonstrate Ruby's RESTfulness, I will use the command `curl` from the command line. Here is an example. It provides a list (in XML) of all the items of the phone book.

```
curl \
-k \
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones.xml
```

The examples in this document assume that the Rails application is accessible through Apache and has been configured to require HTTP Basic Authentication for all operations except for *index*. Details about how to do this are given in the document *Rails HOW-TO: Apache and Basic Authentication*. This document is available at <http://www.oucs.ox.ac.uk/rails/howtos>

Here are some aspects about the `curl` command that are needed to understand the examples.

- If `curl` is used to communicate using `https`, we can use `curl` with a `-k` option in order to advise `curl` not to check the certificate offered by the web server.
- If the web server is using HTTP Basic Authentication, we can use `curl` with a `-u` option to pass the username and password.
- By default, `curl` will use GET. If we use `curl` with a `-d` option, it will use POST.
- A `-d` option imitates the filling in of a box of a web form. So the `-d` is followed by both the name of the box and the value that is to be typed into the box. The `curl` command accepts more than one `-d` option.

And here are some quirks about the use of `curl` with a Rails application:

- If a `-d` option has the name `_method`, Ruby uses the value as the HTTP method that was really wanted. So for PUT use `-d "_method=put"` and for DELETE use `-d "_method=delete"`
- If the URL ends in `.xml`, a Rails application will return XML; otherwise HTML will be returned. Instead of adding `.xml` to the end of the URL, you can also ask for XML by using the option

```
--header="Accept: text/xml"
```

Part of the documentation for Rails says: 'Protecting controller actions from Cross-Site Request Forgery attacks by ensuring that all forms are coming from the current web application, not a forged link from another site, is done by embedding a token based on the session (which an attacker wouldn't know) in all forms and Ajax requests generated by Rails and then verifying the authenticity of that token in the controller.' The token is passed through a hidden field of the form called `authenticity_token`. I found it difficult to give this field the right value when `curl` is being used. The following edit turns off this checking. In practice, it may be wise not to do this.

```
cd /var/apps/contacts
ed app/controllers/phones_controller.rb <<'%'
/class PhonesController/a
  skip_before_filter :verify_authenticity_token
.
w
q
%
```

We will now look at some examples of `curl` being used to query the phone book. Here again is the command that provides a list of all the items of the phone book.

```
curl \  
-k \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones.xml
```

Here is a command that just shows the item at row 2 of the phone book.

```
curl \  
-k \  
-u UN4login:PW4login \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones/2.xml
```

Here are two commands: one that deletes the item at row 2 of the phone book and another that lists the items of the new phone book.

```
curl \  
-k \  
-d '_method=delete' \  
-u UN4login:PW4login \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones/2.xml  
curl \  
-k \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones.xml
```

Here are two commands: one that adds a new item to the phone book and another that lists the items of the new phone book.

```
curl \  
-k \  
-d "phone[name]=lisa" \  
-d "phone[number]=23456" \  
-u UN4login:PW4login \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones.xml  
curl \  
-k \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones.xml
```

Finally, here is a command that alters item 3 of the phone book. As before, it is followed by a command that lists the items of the new phone book.

```
curl \  
-k \  
-d '_method=put' \  
-d 'phone[name]=dave' \  
-d 'phone[number]=12345' \  
-u UN4login:PW4login \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones/3.xml  
curl \  
-k \  
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones.xml
```