LINUX VOICE TUTORIAL

CODE NINJA: RESTFUL APIS

BEN EVERARD

Access resources from around the web and integrate them into your own site in a simple, uniform manner.

WHY DO THIS?

- Understand how huge databases handle billions of transactions.
- Gain the flexibility of not having a schema.
- Use the trendiest database in town.

The web is a great way of sharing data. Anyone, anywhere in the world can upload information and make it instantly available to almost everyone in the world. However, as wonderful as these web pages are, they do have a problem: it's hard for computers to understand them. Web browsers can obviously render web pages, but they can't easily extract information. Take, for example, a web page of a weather forecast. The browser knows what text to put where, and which images should be displayed, but it's hopeless at understanding the forecast, and can't easily pull out the data so it can be displayed in a different forecast.

When a person or organisation wants to make their information available for computers to understand it, they need to create an API (Application Program Interface). This is a method for programs to extract the information they need in a computer-readable format. The most popular method for doing this on the internet is through REpresentational State Transfer (REST) APIs.

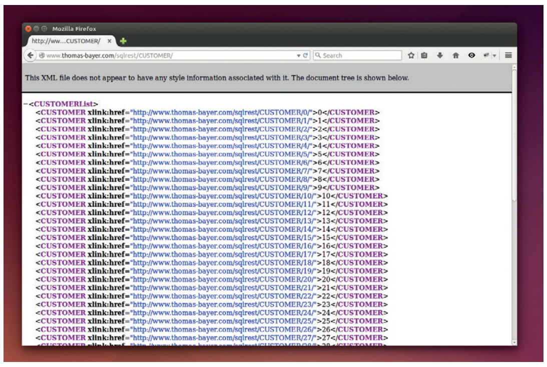
A RESTful API must be client–server; stateless; cacheable; layered; code on demand (optional); and have a uniform interface.

Let's look at these properties by analysing the largest RESTful system, the world-wide web. This is a series of HTML documents that are sent and received using Hyper Text Transfer Protocol (HTTP). The web is client–server. This means there's a separation between the software that browses the web (such as *Firefox*) and the software that serves the web pages (such as *Apache*).

The web (or rather, web servers) is stateless. This means that if you make the same request, you should get the same result. It doesn't matter what requests you've sent before this. This still allows for things like authentication, because this session data can be sent in the HTTP request along with the URL.

Web resources can be cacheable. To statisfy the RESTful requirements, not everything has to be cached, but information needs to be included about what can and can't be cached. This is taken care of in HTTP headers.

When you connect to a website, you could be connecting directly to the server, or through some proxy. This is all handled transparently because HTTP is layered. Code on demand is the only optional aspect of the REST principals. However, the web does allow it. This is when the server sends some code to be executed on the client. In terms of the web, this is usually JavaScript or a Java applet.



```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

-<CUSTOMERList>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/0/">0</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/1/">1</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/2/">2</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/3/">3</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/4/">4</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/5/">5</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/6/">6</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/7/">7</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/8/">8</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/9/">9</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/10/">10</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/11/">11</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/12/">12</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/13/">13</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/14/">14</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/15/">15</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/16/">16</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/17/">17</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/18/">18</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/19/">19</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/20/">20</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/21/">21</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/22/">22</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/23/">23</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/24/">24</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/25/">25</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/26/">26</CUSTOMER>
   <CUSTOMER xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/27/">27</CUSTOMER>
```

The XML pages are viewable in a web browser, but the links aren't clickable, so you'll have to copy and paste them into the URL bar.

URLs are in a uniform structure, and there's a uniform series of four HTTP requests that you can perform on them (GET, PUT, POST and DELETE) . Although four are all available in theory, in practice almost all websites only use GET and POST, the latter of which is used when you submit a form.

There's an example API that links an SQL database to a RESTful service at **www.thomas-bayer.com/sqlrest**. If you point your browser there, you'll see an XML document describing all the resources available.

You should notice that they're all resources one level above **/sqlrest/**. This is part of the uniform architectural constraint. If you point your browser to one of the listed resources – **www.thomas-bayer.com/sqlrest/CUSTOMER/** – you should then see a list of the customers. Again, each listed resource is one level above **/sqlrest/CUSTOMER/**.

This is a public API that allows anyone to change and delete information without authentication, so it's possible that the exact examples we use now won't exist any longer in the database. If they don't, you'll have to substitute in other values.

You can view one of the customers by viewing a link in this list, for example, **www.thomas-bayer.com/sqlrest/CUSTOMER/16**. Unlike the other queries we've done, this doesn't just return a list of resources, but the actual customer information as an XML file.

We can see that the format of the URLs for this database is: **www.thomas-bayer.com/sqlrest/<table>/<id>/**. Using this, we can extract any information from the database. We can also manipulate the database using other HTTP methods. POST, PUT and DELETE map to amending a record, adding a record and deleting a record respectively. However, you can only generate GET requests using the URL bar of most web browsers, so we'll need another tool to send these requests. There's a web-based HTTP tool at **http://thomas-bayer.com/restgate/** that will do what we need.

To add a new item to the database, we use the PUT method on the URL for the table. So, for example, to add a new person at ID 0, you'll need to go to **http://thomas-bayer.com/restgate/**, then enter the URL **http://www.thomas-bayer.com/sqlrest/CUSTOMER**, select the method PUT, then in the content text box (that will appear when you select PUT), enter the following XML:

```
<CUSTOMER>
<ID>0</ID>
<FIRSTNAME>Ben</FIRSTNAME>
<LASTNAME>Everard</LASTNAME>
<STREET>xxx</STREET>
<CITY>Brizzle</CITY>
</CUSTOMER>
```

Note that you'll have to use a different ID when you do it as this one's taken. At the time of writing, this request returned an error, however, the actual record was updated. You can see for yourself at **http://www.thomas-bayer.com/sqlrest/CUSTOMER/0/**.

Using the same web form, you can amend a customer's details. This time, enter the URL **http://www.thomas-bayer.com/sqlrest/CUSTOMER/0/** (again, you'll need to change the ID to match one you've created), the method POST and the content:

```
<STREET>Colston Av</STREET>
```

Finally, you can use DELETE to remove customers.

So far, what you've seen is a slightly awkward way of accessing a database through a browser. This isn't at all the point of a RESTful API. The point is to make it easy for other software to interact with our service.

Doing it in Python

In Python, we can use the **requests** module to interact with this RESTful API. This is easy to use, so we'll use an Python interactive session, which you can start by typing **Python2** at the command line. Then you need to import the **requests** module with:

```
>>> import requests
```

A GET request is then simply done with:

```
>>> r = requests.get("http://www.thomas-bayer.com/sqlrest/
CUSTOMER/0/")
>>> print r.content
```

This will print out the raw XML. If you were doing this in a real program, you could either manipulate the XML as strings, or you could use one of the XML parsing modules to do it for you.

The requests module can also issue POST, PUT and DELETE requests.

```
>>> mike_data="'<CUSTOMER>
...     <ID>897324</ID>
...     <FIRSTNAME>Mike</FIRSTNAME>
...     <LASTNAME>Saunders</LASTNAME>
...     <STREET>xxx</STREET>
...     <CITY>Munich</CITY>
... </CUSTOMER>"'
>>> r = requests.put("http://www.thomas-bayer.com/sqlrest/
CUSTOMER/", data=mike_data, headers={'Content-
Type':'application/xml'})
>>> r = requests.get("http://www.thomas-bayer.com/sqlrest/
CUSTOMER/897324/")
>>> print r.content
```

This should display the new record we've just created for Mike Saunders. We can now update it to his nicknames with:

```
>>> r = requests.post("http://www.thomas-bayer.com/sqlrest/
CUSTOMER/897324/", data="<FIRSTNAME>Mikeyboy</
FIRSTNAME>", headers={'Content-Type':'application/xml'})
>>> r = requests.get("http://www.thomas-bayer.com/sqlrest/
CUSTOMER/897324/")
>>> print r.content
```

Then delete it with:

```
>>> r = requests.delete("http://www.thomas-bayer.com/sqlrest/
CUSTOMER/897324/")
```

In this case, we've been updating a database, but RESTful APIs exist for all sorts of data source. You should be able to use the same methods we've used here to access the vast majority of them which use HTTP. In many cases, you'll need to authenticate yourself before you can perform any action (especially those that modify data), but this should be fully documented as it differs between APIs.

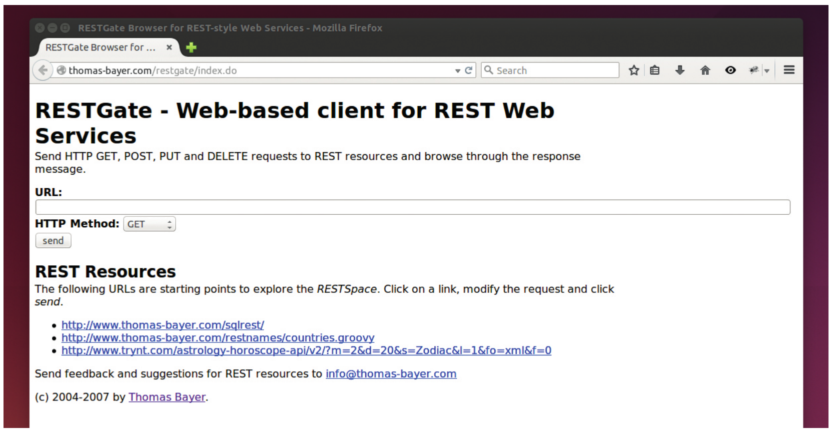
To go back to the first problem, how could our program get a weather forecast:

```
>>> r = requests.get("http://api.openweathermap.org/data/2.5/
weather?q=Bristol&mode=xml"
)

>>> print r.content
```

For more information on this weather API, go to **http://openweathermap.org/current**.

We've looked at Python here, but you should find equivalent libraries in most languages that make it just as simple. Once you've mastered sending HTTP requests, the process is quite straightforward. It's this simple approach that makes HTTP-based RESTful APIs so simple and ubiquitous.


RESTGate Browser for REST-style Web Services - Mozilla Firefox
RESTGate Browser for ...
thomas-bayer.com/restgate/index.do
Search

RESTGate - Web-based client for REST Web Services

Send HTTP GET, POST, PUT and DELETE requests to REST resources and browse through the response message.

URL:

HTTP Method: GET

send

REST Resources

The following URLs are starting points to explore the *RESTSpace*. Click on a link, modify the request and click *send*.

- http://www.thomas-bayer.com/sqlrest/
- http://www.thomas-bayer.com/restnames/countries.groovy
- http://www.trynt.com/astrology-horoscope-api/v2/?m=2&d=20&s=Zodiac&l=1&fo=xml&f=0

Send feedback and suggestions for REST resources to info@thomas-bayer.com

(c) 2004-2007 by Thomas Bayer.

The RESTgate webpage can be used to interact with almost any HTTP RESTful API, not just the ones we've used here.