**Innovation time January 2011**

After having read the o’Reilly book “REST in Practice” (<http://www.amazon.co.uk/REST-Practice-Hypermedia-Systems-Architecture/dp/0596805829>), I set myself the challenge of using OpenRasta (<http://trac.caffeine-it.com/openrasta>) to create a basic RESTful web service.

I decided for the first day to just concentrate on getting a basic CRUD app as outlined in chapter 4 working. This involved the ability to create, read, update and delete physical file xml representations of Artists. It is described in the book as a Level 2 application on Richardson’s maturity model (<http://martinfowler.com/articles/richardsonMaturityModel.html>), as it doesn’t make use of Hypermedia yet.

One reason why OpenRasta is such a good framework to implement a RESTful service is that it deals with “resources” and their representations (<http://www.zephyros-systems.co.uk/blog/?p=45>). As outlined in “REST in Practice”, a resource is defined as any resource accessible via a URI, and OpenRasta deals with this perfectly.

Back to the matter at hand, for the basic web service I created an ArtistHandler in the normal OpenRasta way (<http://trac.caffeine-it.com/openrasta/wiki/Doc/Tutorials/Handlers>), creating c# methods within the Handler for each of these four HTTP verbs:

GET for reading .  
POST for creating.  
PUT for updating.  
DELETE for deleting.

I used the [HttpOperation] attributes just to show that you can, OpenRasta will auto map a method with the name Post() to the POST verb.

The main aim of this exercise was to discover exactly what http response statuses and headers I should be returning, and whether it was possible to adhere strictly to the guidelines using OpenRasta.

The HTTP template I used for the endpoint was:

**/artist/{artistId}**

The Responses I wanted to give were structured as they are outlined in the book, and by 3w.org (<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>) e.g:

**GET /artist/{artistId}:**

* Returns a 400 BadRequest along with a list of errors, if artistId not supplied.
* Returns a 404 NotFound if record for that artist is not found
* Returns a 200 OK along with the record if the record was found
* Returns a 500 Internal Server Error on exception

**POST /artist**

* Returns a 400 BadRequest along with a list of errors, if any parameters not supplied.
* Returns a 302 Found along with the Location uri of the resource if it already exists.
* Returns a 201 Created along with the Location uri of the new resource on success (this could also contain the body of the new resource)
* Returns a 500 Internal Server Error on exception

**PUT /artist/{artistId}**

* Returns a 400 BadRequest along with a list of errors, if any parameters not supplied.
* Returns a 404 NotFound if record for that artist is not found
* Returns a 204 NoContent along with the Location uri of the updated resource on success(not sure about this myself, but was specified in the book)
* Returns a 500 Internal Server Error on exception

**DELETE /artist/{artistId}**

* Returns a 400 BadRequest along with a list of errors, if any parameters not supplied.
* Returns a 404 NotFound if record for that artist is not found
* Returns a 204 NoContent on success.
* Returns a 405 MethodNotAllowed on any IO exception
* Returns a 503 Service Unavailable on any other exception

I had a couple of issues with responses and OpenRasta, for instance, there is not a set object representing a 503 Service Unavaiable response, but I could create my own by changing some settings in an InternalServerError Response.

Also, I wasn’t able to pass POX (Plain Old Xml) to the POST endpoint without OpenRasta throwing an internal exception, something which I’ll have a look at in due course.

I used Curl to test the endpoints, I tried Fiddler, but OpenRasta would always return a 415 Media Not Supported response, I imagine due to one of the headers not being specified properly, again this may be worth looking into. Using Curl is quick and easy, I just used variations on the following

One thing you need to do to make a service move towards a Level 3 rating, is to offer up links to be able to access endpoints related to this resource, e.g. links to page to the previous or next record, or a link to fulfil or pay for an order. As a nod to this, I created a link to DELETE a record that is returned when you GET an artist e.g.  
  
<link rel=”artist” href=”http://localhost/restful\_service/artist/10010” method=”DELETE”/>

REST in practice recommends the use of Atom feeds to truly create a Level 3 restful service, but Martin Fowlers post on the Richardson maturity Model suggests simply using standard html style link tags like I have used for the DELETE link above.

There were many other things I would have liked to look at, namely Caching, E-Tags, creating Atom feeds and implementing OAuth, but I ran out of time. At the time of writing, OpenRasta does not support OAuth out of the box, but according to this post <http://groups.google.com/group/openrasta/browse_thread/thread/c55f9aaf157b4f04?fwc=1> it is something they are looking into. An interesting move forward would be to create an OAuthAuthenticationScheme : IAuthenticationScheme within our own fork of OpenRasta. (<https://github.com/7digital/openrasta-stable>)

You can grab the project from here:  
<https://github.com/gregsochanik/RESTfulService>

Links:

<http://martinfowler.com/articles/richardsonMaturityModel.html>

<http://trac.caffeine-it.com/openrasta>

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

<http://www.amazon.co.uk/REST-Practice-Hypermedia-Systems-Architecture/dp/0596805829>

<http://trac.caffeine-it.com/openrasta/wiki/Doc/Tutorials/Handlers>

<http://www.zephyros-systems.co.uk/blog/?p=45>