**Innovation time January 2011**

After having read the o’Reilly book “REST in Practice” (Grahams book if anyones interested), 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 Richardsons 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 – see below)
* 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 – see below)
* 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 201 Created along with the Location uri of the new resource (this could also contain the body of the new resource – see below)
* Returns a 503 Internal Server Error on exception

[Issues with responses and OpenRasta - not being able to send xml as a response when created to save round trip]

[Using Curl to test]

[Not being able to POST Xml ]

[Adding Location headers for 201 Created/204 No Content]

[Creating a link for a Delete – very basic homage to Atom:link]

[Things I wanted to do – Caching/Etags, Atom feeds, OAuth (no OpenRasta support as yet), talk about track download endpoint]

[Not being able to create a meaningful response to a Bad Request (e.g. telling you which params are missing)]

[Link to project]

REST in practice

**Day 1**  
Build a RESTful service using OpenRasta  
Use album, artist, track model  
Use xsd to validate schema  
Has to be fully RESTful as outlined in the book, without using Atom

**Day 2**

Build another service that can only be accessed with oAuth

Get Service A talking to service B (e.g., as with the RestBucks voucher system, could be a track purchase/download?)

If there is time, look at implementing Atom links.