[Introduction 1](#_Toc211267216)

[Relationship with Naked Objects 2](#_Toc211267217)

[Building from Source 2](#_Toc211267218)

[Prerequisites 2](#_Toc211267219)

[Source Code Repository 2](#_Toc211267220)

[Building Naked Objects 3](#_Toc211267221)

[Building from the Command Line 3](#_Toc211267222)

[Building in Eclipse 3](#_Toc211267223)

[Building Restful Objects 4](#_Toc211267224)

[Building from the Command Line 4](#_Toc211267225)

[Building within Eclipse 4](#_Toc211267226)

[Running the Example 6](#_Toc211267227)

[Using Restful Objects from a Web Browser 7](#_Toc211267228)

[Walkthrough 8](#_Toc211267229)

[Developing in Eclipse 16](#_Toc211267230)

[Coding Standards and so on 16](#_Toc211267231)

[Hints and Tips 16](#_Toc211267232)

# Introduction

Restful Objects provides a RESTful API to Naked Objects[[1]](#footnote-2). In effect, for a given domain application it exposes both the Naked Objects meta-model and the objects managed by the Naked Objects runtime.

The representation language used by RestfulObjects is XHTML, therefore viewable in a web browser (eg Firefox or IE7). This XHTML isn’t pretty to look at; the content is intended to be consumed by a client-side program. Nevertheless, it is useful from a development perspective to be able to easily view the representations created by the Restful Objects codebase.

Restful Objects goes further than this though. In usual RESTful style, the resources (i.e. domain objects) are manipulated using HTTP GET, PUT, DELETE and POST. The XHTML representations generated include XHTML FORM elements to invoke set properties, add/remove objects to collections, and to invoke actions. Because current browsers cannot perform PUT and DELETE methods, the XHTML forms invoke simple Javascript to make the HTTP calls. A dedicated client-side program is expected to ignore these XHTML FORMs and to invoke the HTTP resources directly.

# Relationship with Naked Objects

Restful Objects is a sister project to Naked Objects, and as such conforms to the general architecture defined by Naked Objects. Specifically, it provides a viewer component, analogous to Naked Objects’ own HTML viewer or indeed to the viewer provided by Scimpi[[2]](#footnote-3) (another sister project).

Like those other viewers, Restful Objects ultimately consists of a set of servlets and servlet filters. For a production deployment these need to be assembled into a webapp; the intent is to provide an archetype to assist with this.

For development a less heavyweight approach is appropriate. Restful Objects therefore uses some of the infrastructure provided by Naked Objects, deploying its servlets within a standalone Jetty server. This is implemented a (Naked Objects) *ServletListener*. Bottom line is that Restful Objects can be run as a standalone command line program that bootstraps the Jetty webserver. More on this below.

# Building from Source

If you are thinking about modifying or contributing to Restful Objects, then you’ll want to be able to build the code from source and to deploy in a development environment.

Both Restful Objects and Naked Objects are built using Maven 2. We recommend that you build Naked Objects itself from source, installing the Naked Objects modules into your local Maven repository. Then, you can build Restful Objects.

Although you can just use Maven from the command line, you’ll almost certainly want to use an IDE for proper development. We use Eclipse IDE with the M2Eclipse plugin. The maven-eclipse-plugin can be used to generate the Eclipse .project and related files; that’s why they aren’t checked into source code. If you use another IDE, then you may well find a similar Maven plugin.

## Prerequisites

Install a Subversion client, for example TortoiseSVN[[3]](#footnote-4).

Install Java 5, setup JAVA\_HOME

Install Maven 2.0.9 or later, setup MAVEN\_HOME, add *mvn* to PATH.

Install Eclipse 3.4 (JEE edition recommended).

Also highly recommended are the M2Eclipse and the Subclipse plugins[[4]](#footnote-5).

## Source Code Repository

Restful Objects is hosted at  [https://restfulobjects.svn.sourceforge.net/svnroot/restfulobjects](https://nakedobjects.svn.sourceforge.net/svnroot/nakedobjects). Naked Objects is hosted at <https://nakedobjects.svn.sourceforge.net/svnroot/nakedobjects>.

## Building Naked Objects

### Building from the Command Line

The Naked Objects framework is at .../framework/trunk. Check out that directory. Then build using:

$ mvn clean install

### Building in Eclipse

First, at the command line, use

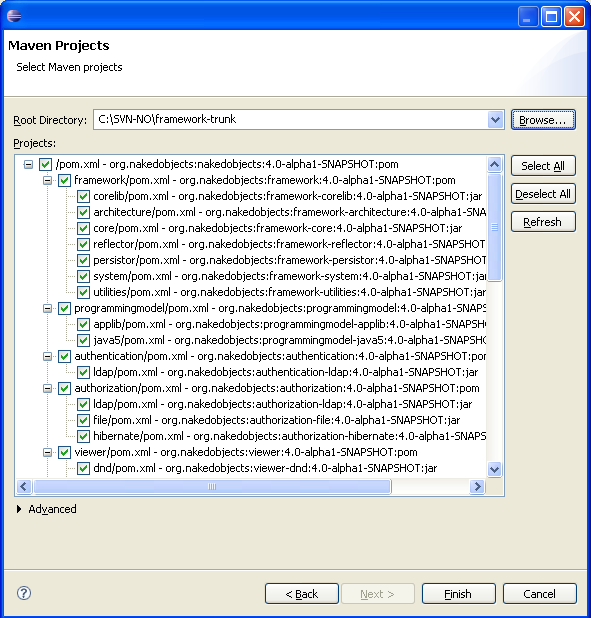
$ mvn eclipse:m2eclipse

This (re)generates the Eclipse project files from the pom.xml files.

Then, in Eclipse create a new empty workspace outside the SVN directory tree. Then import all the projects using:

File > Import > General > Maven Projects

Specify the framework/trunk as root directory. The M2Eclipse plugin will locate all the Maven projects referenced:



Hit Finish. You’ll end up with a bunch of projects. You might then want to organize into working sets. We tend to organize by the top-level modules.

At the time of writing there seems to be one slight issue with the M2Eclipse: the classpath is incorrect for any webapp projects. There is currently only one such project: bootstrap-webapp. There is a howto doc in the root of that project that explains how to fix the classpath (basically: add the ‘Web App Libraries’ to the Java Build Path).

## Building Restful Objects

The Restful Objects codebase lives in trunk/restfulobjects. The modules are:

* viewer – the servlets that generates the representations using the services of Naked Objects
* server-jetty - implementation of a Naked Objects ServerListener, running Restful Objects within an embedded Jetty servlet container.

To try out the code we also need a domain application; we’ll use the example project in trunk/examples/carserv. Its modules are:

* carserv-dom is the domain model. The domain is a car servicing application (Customer, Car and Service).
* carserv-fixture is initial fixture set
* carserv-bootstrap is used to run up Restful Objects as a Naked Objects server listener using in-built Jetty.

This example is only intended for use with in-memory or xml object stores; there are no hibernate mappings or hibernate implementations of repositories. It’s good enough for testing purposes though. The example has one additional dependency on a non-Maven library. Further details on installing this below.

The process for building Restful Objects is exactly the same as for Naked Objects.

### Building from the Command Line

To build Restful Objects, use:

$ cd .../trunk/restfulobjects  
$ mvn clean install

To build the CarServ example, first download the timeandmoney[[5]](#footnote-6) JAR file (v0.5.1) from (the non-Maven dependency). Then, install into local repository using:

$ mvn install:install-file -D file=timeandmoney-v0\_5\_1.jar -D groupId=com.domainlanguage  
 -D artifactId=timeandmoney -D version=0.5.1 -D packaging=jar -D generatePom=true

You can then build the CarServ example:

$ cd .../trunk/examples/carserv  
$ mvn clean install

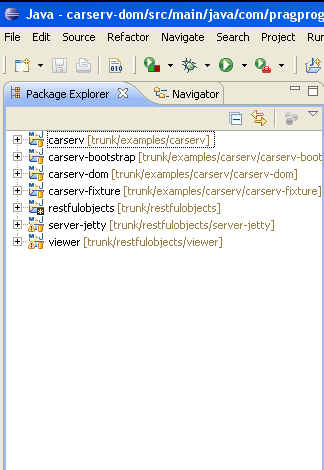
### Building within Eclipse

To generate the Eclipse project files, use:

$ mvn eclipse:m2eclipse

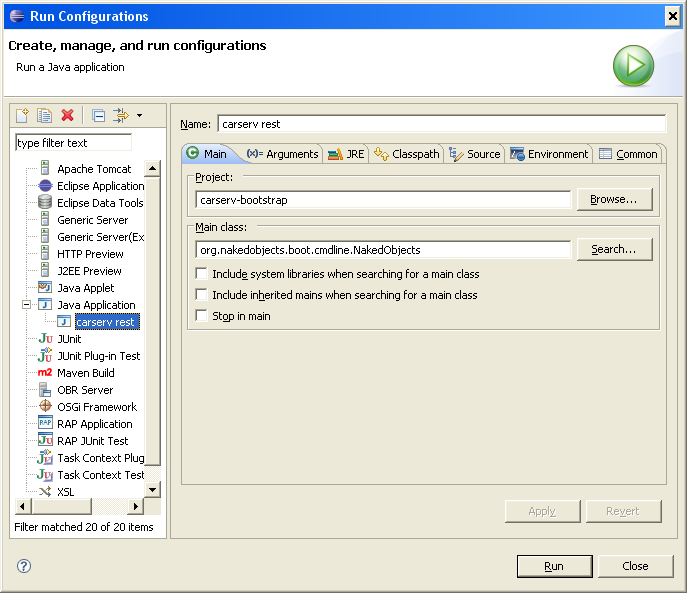
Do this for trunk/restfulobjects, and then for trunk/examples/carserv.

Then, in Eclipse, create a new workspace and import first Restful Objects and then the CarServ example. You should end up with:

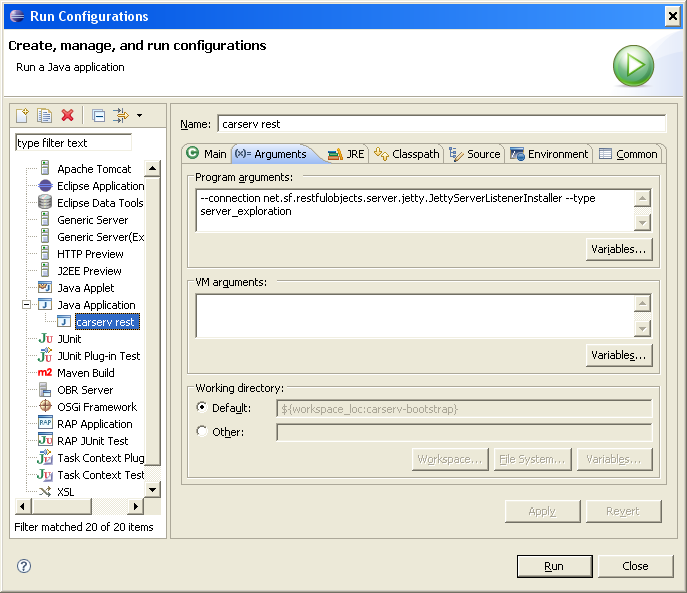


# Running the Example

The application is run using org.nakedobjects.boot.cmdline.NakedObjects (which contains the main(...) method), specifying the RestfulObject’s ServerListener. The easiest approach is to create the following Eclipse launch configuration:



The main class is: *org.nakedobjects.boot.cmdline.NakedObjects*



The arguments are:

*--connector net.sf.restfulobjects.server.jetty.JettyServerListenerInstaller --type server\_exploration*

This will run Naked Objects in (server-side) exploration mode, meaning no authentication is performed, using an in-memory object store.

It is also possible to run in prototype mode (requiring authentication, but still with an in-memory object store), or to run in regular “production” mode. The table below summarizes the options:

Table : Type options

|  |  |  |  |
| --- | --- | --- | --- |
| Type | --type | Object Store | Authentication |
| Exploration | server\_exploration | In-memory | No |
| Prototype | server\_prototype | In-memory | Yes; see config/passwords |
| Production | Server | XML ; change using –object-persistor | Yes; uses config/passwords |

NB: to run up the DnD (to compare/contrast), the arguments are --type exploration --viewer dnd.

## Using Restful Objects from a Web Browser

The representations of the meta-model and domain objects generated by Restful Objects are intended to be consumed by a client-side program. However (as discussed earlier), because the representations are XHTML plus some Javascript, it is possible to use a Web browser.

The representations fall into the following categories:

* /services

all of the registered services (corresponding to icons in the DnD viewer)

* /object/*encoded\_OID*

shows a specific object. Note that services are singleton objects

* /specs/

lists all known NakedObjectSpecifciations

* /specs/*fullyQualifiedClassName*

shows a specific NakedObjectSpecifciation.

* /user

shows the currently logged-on user.

Each is generated by corresponding XxxResource classes. So for example /services is generated by ServicesResource, /specs by SpecsResource. Each also supports various subresources.

If running in server\_prototype mode, then the user/password should be specified as parameters:

<http://localhost:7070/services?user=rcm&password=pass>

where config/passwords contains:

fbloggs:pass

rcm:pass:role1|role2|role3

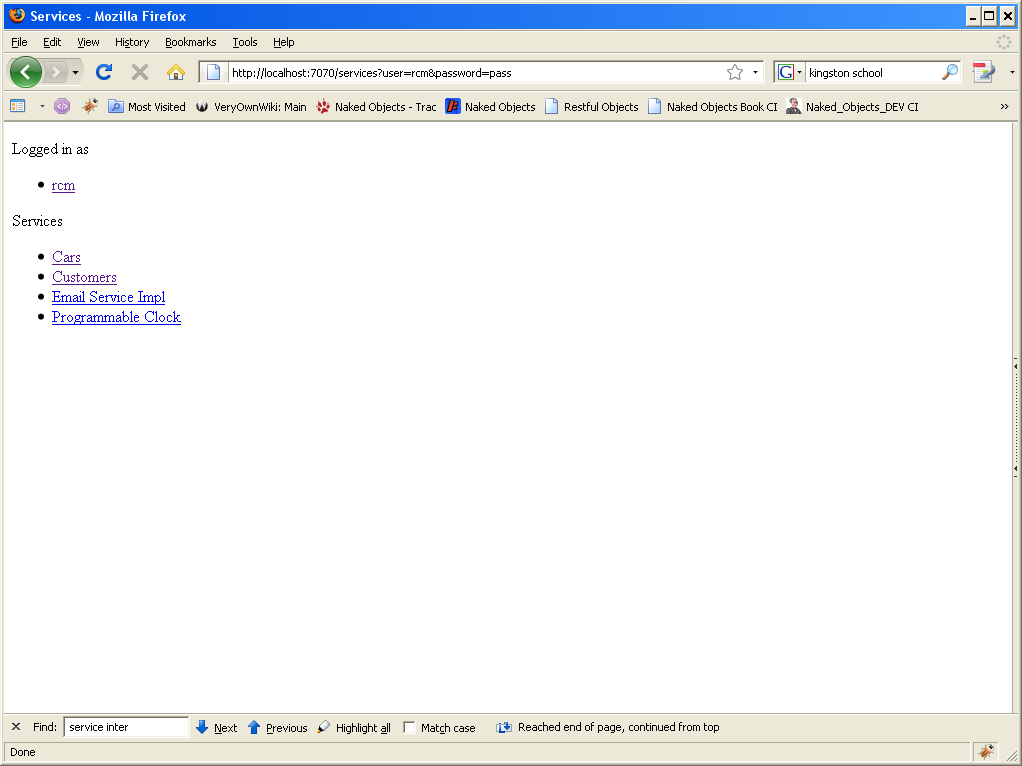
dgs:pass

This admittedly isn’t very secure, but it’s something to look at later. (Would SSL be sufficient, passing the parameters as HTTP headers?)

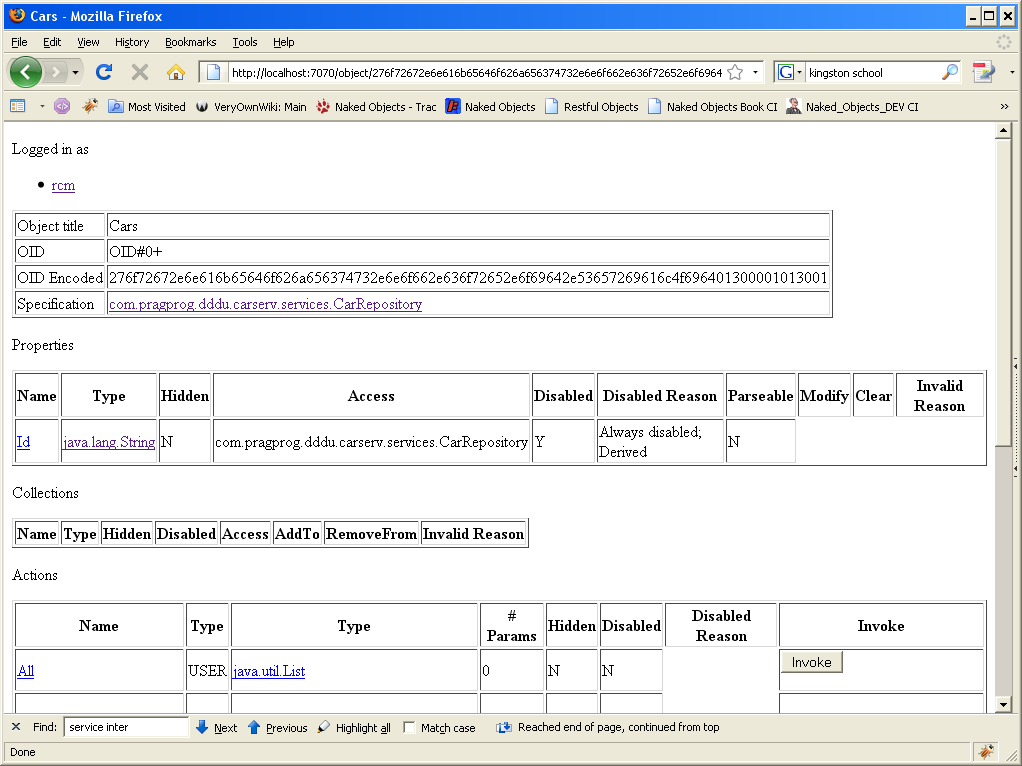
## Walkthrough

(You could also run through this using the DnD viewer).

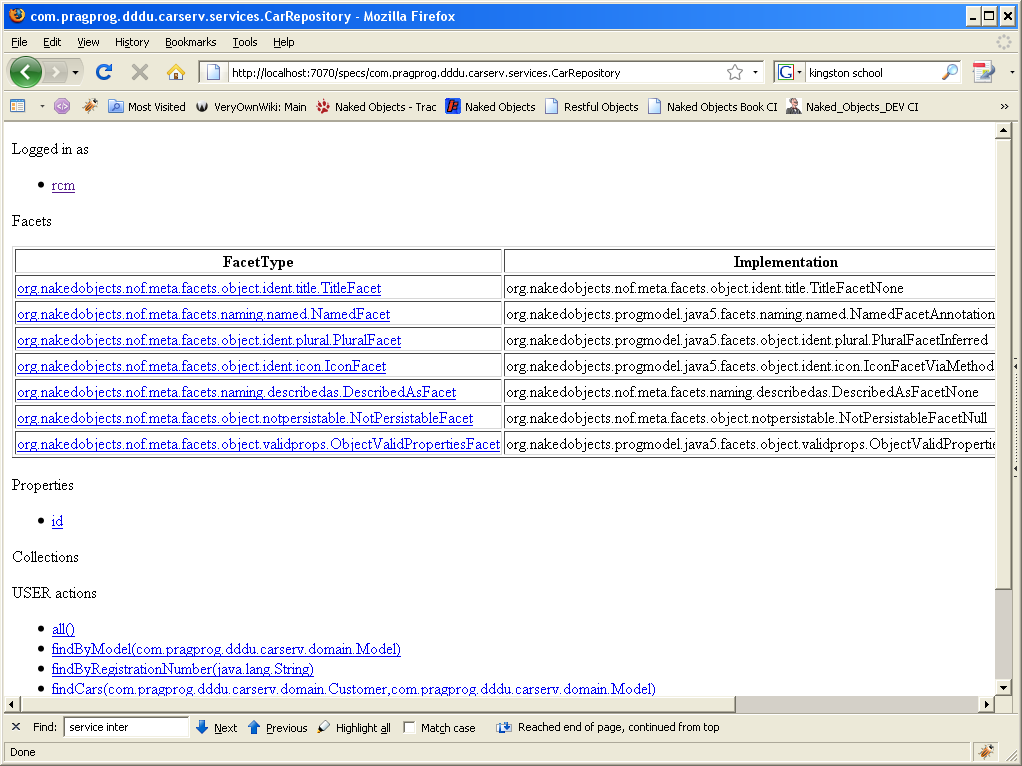
List services:

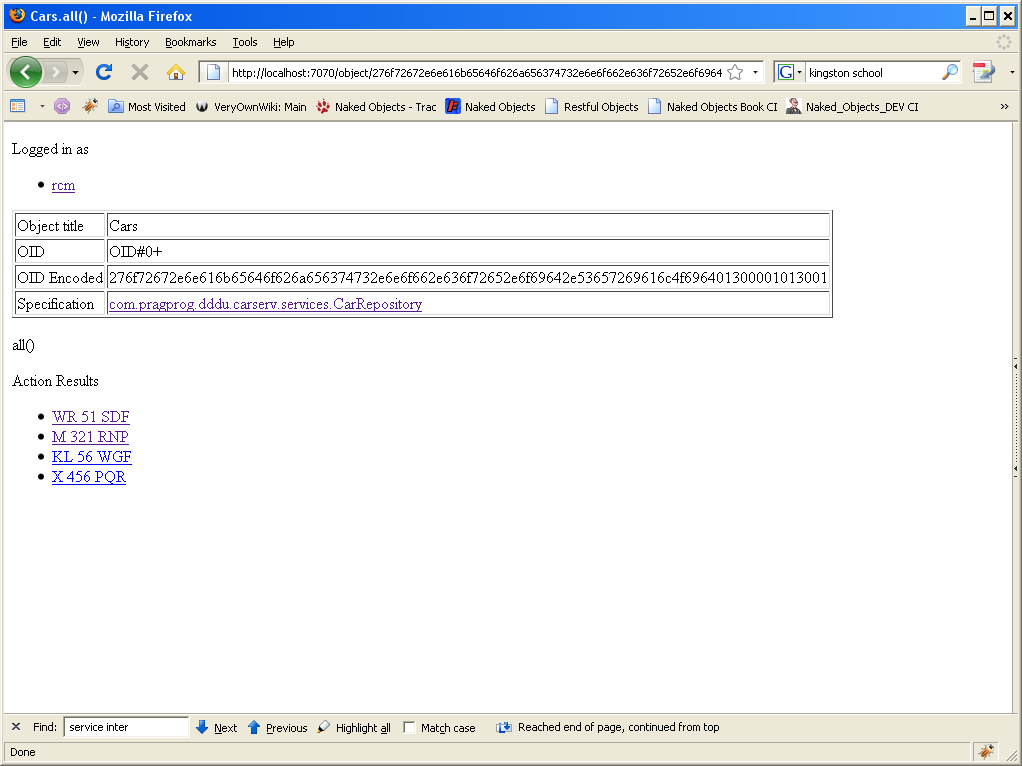


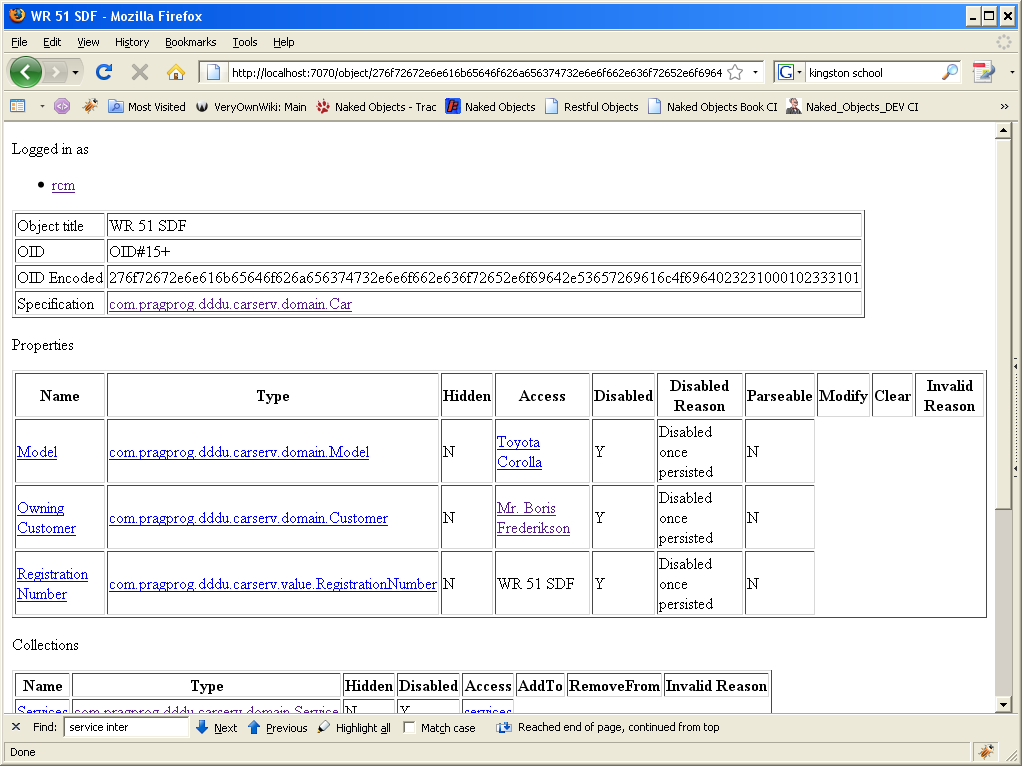
Click on Cars link, to bring up the CarRepository service:

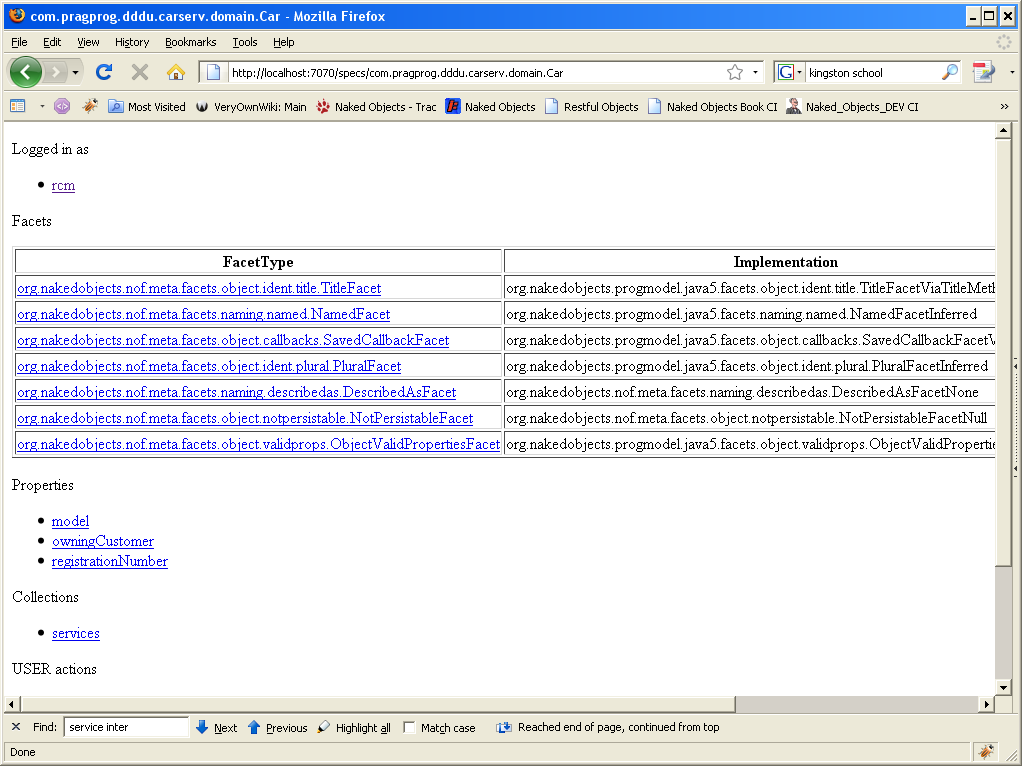


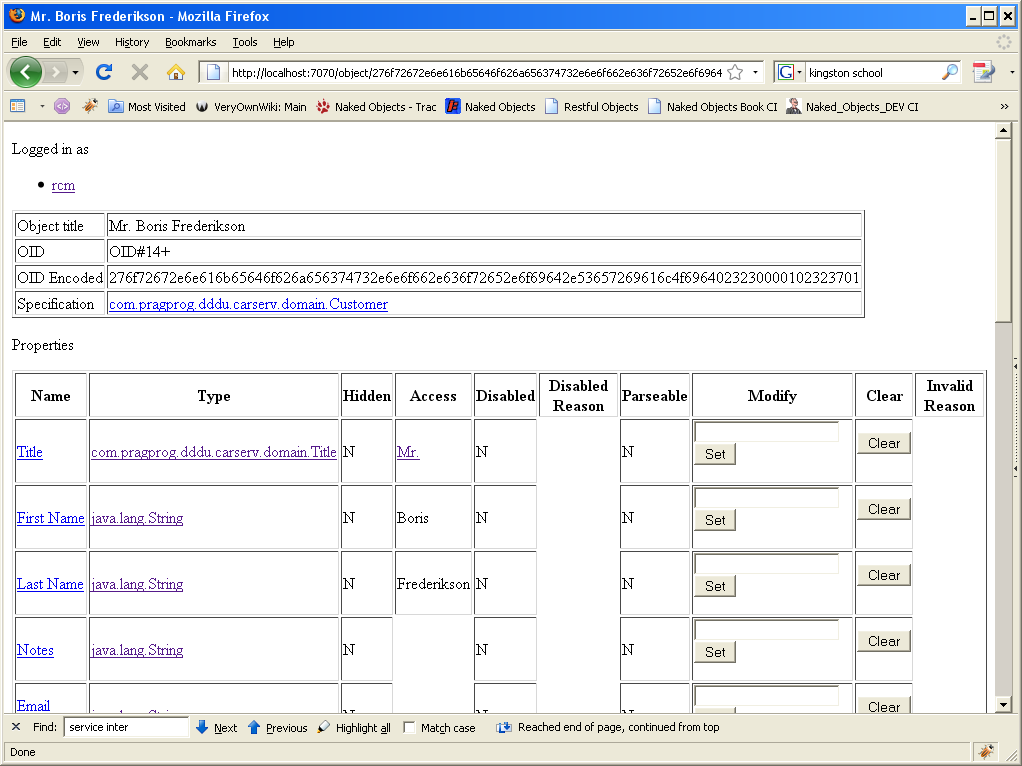
Click on the Specification link to view the structure of CarRepository:



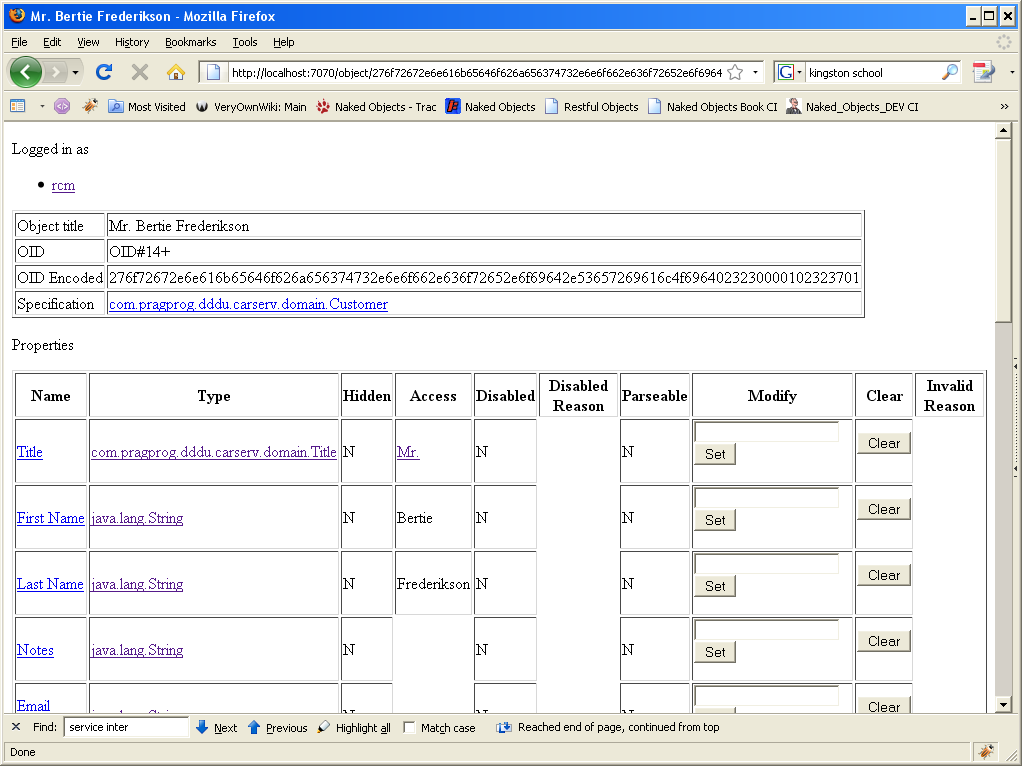
Navigate back to the CarRepository object.   
Click on the “All” actions Invoke button (this performs a POST):p

Click on “WR 51 SDF” link:

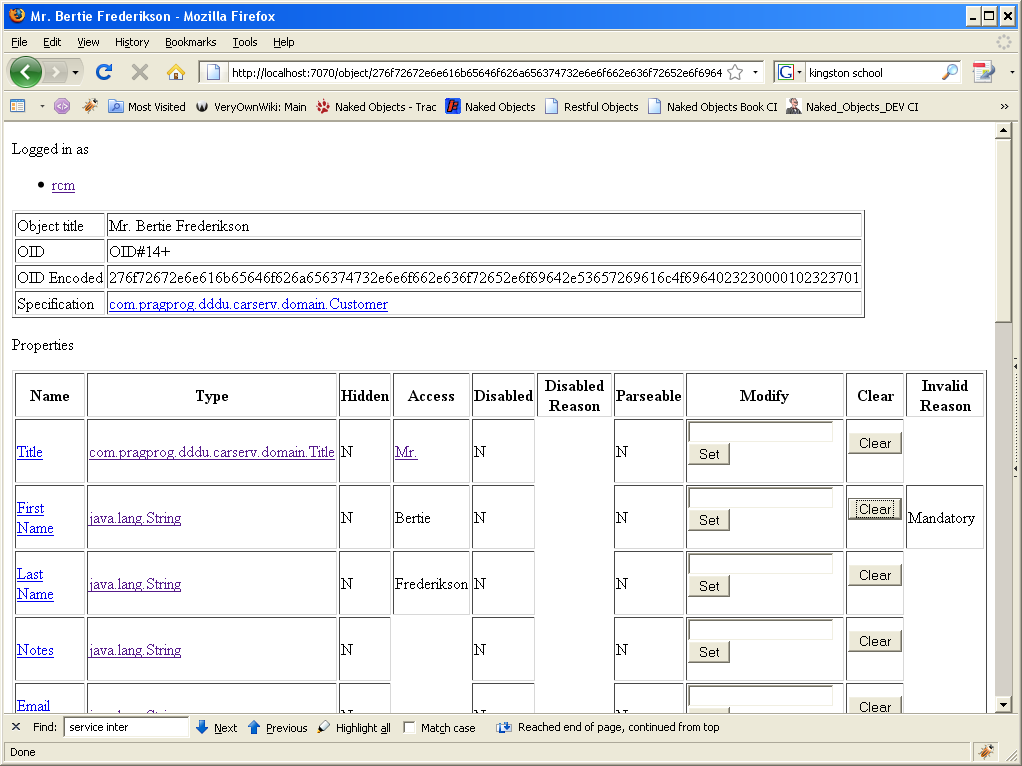
Click on Specification link to view the structure of the Car class:

Navigate back to the “WR 51 SDF” car.  
Click on “Mr. Boris Frederikson” (referenced as the owning customer):

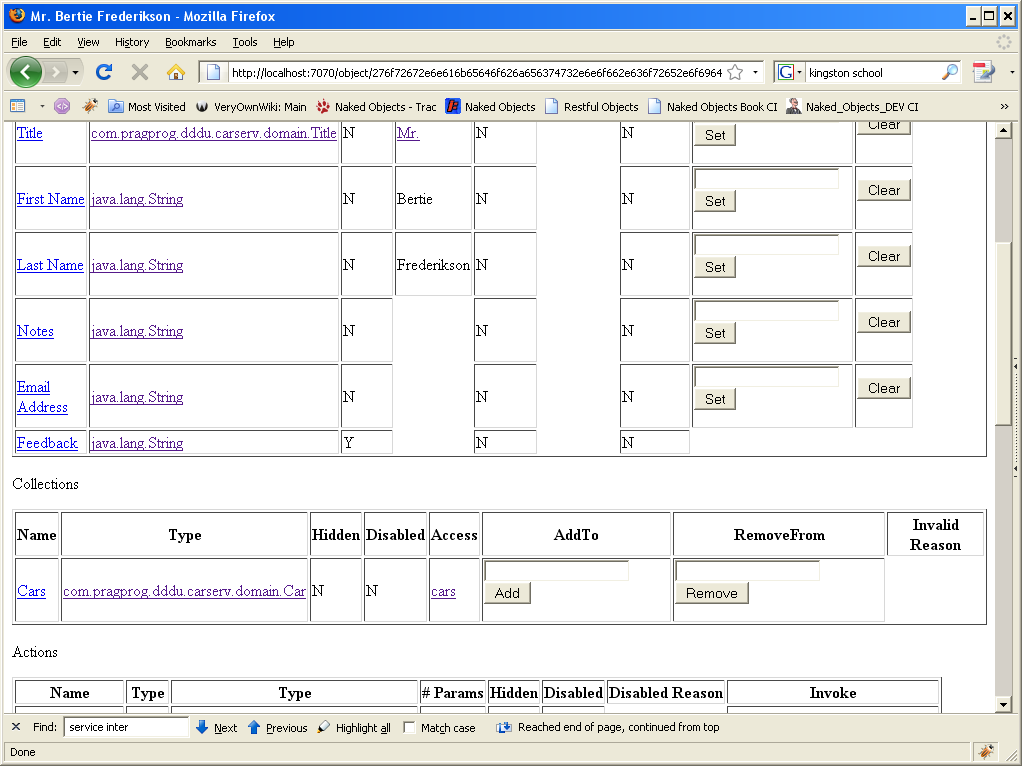
For the “FirstName” property, in the “Modify” column, enter “Bertie”, then hit “Set”. This calls Javascript to perform a PUT. If OK (as in this case) then the Javascript then additionally reloads the same resource:

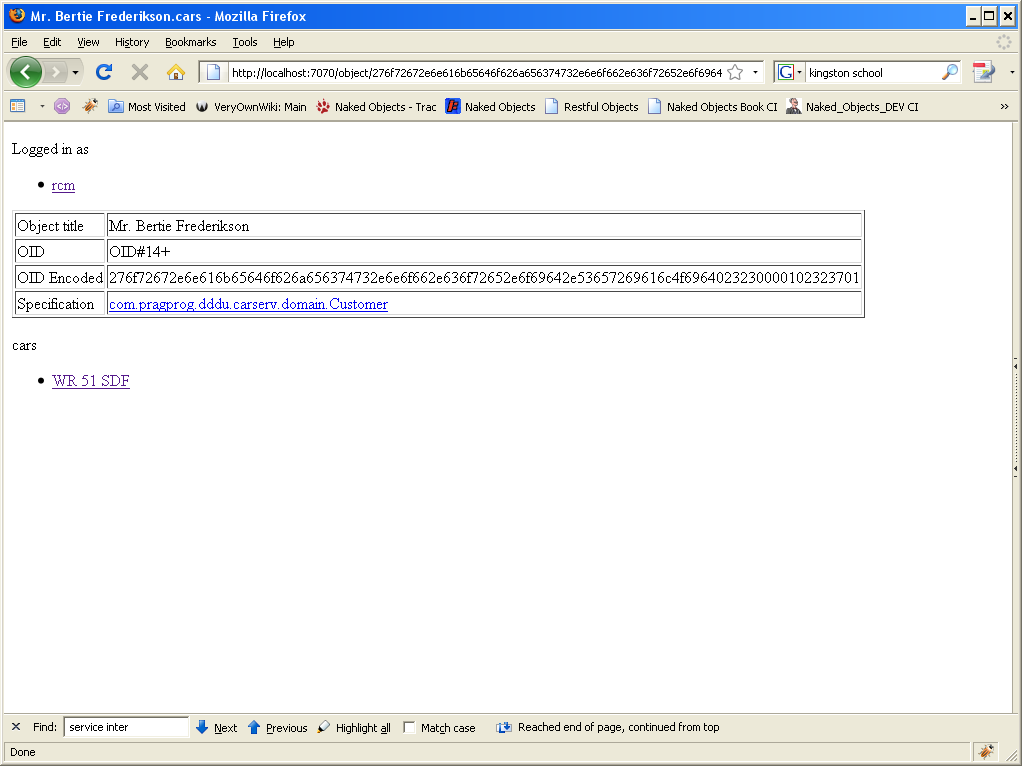


For the “FirstName” property, select the “Clear” button. This calls Javascript to perform a DELETE. Since this isn’t valid, the DELETE (or it could have been a PUT) returns an error. The Javascript updates the InvalidReason DOM Id:



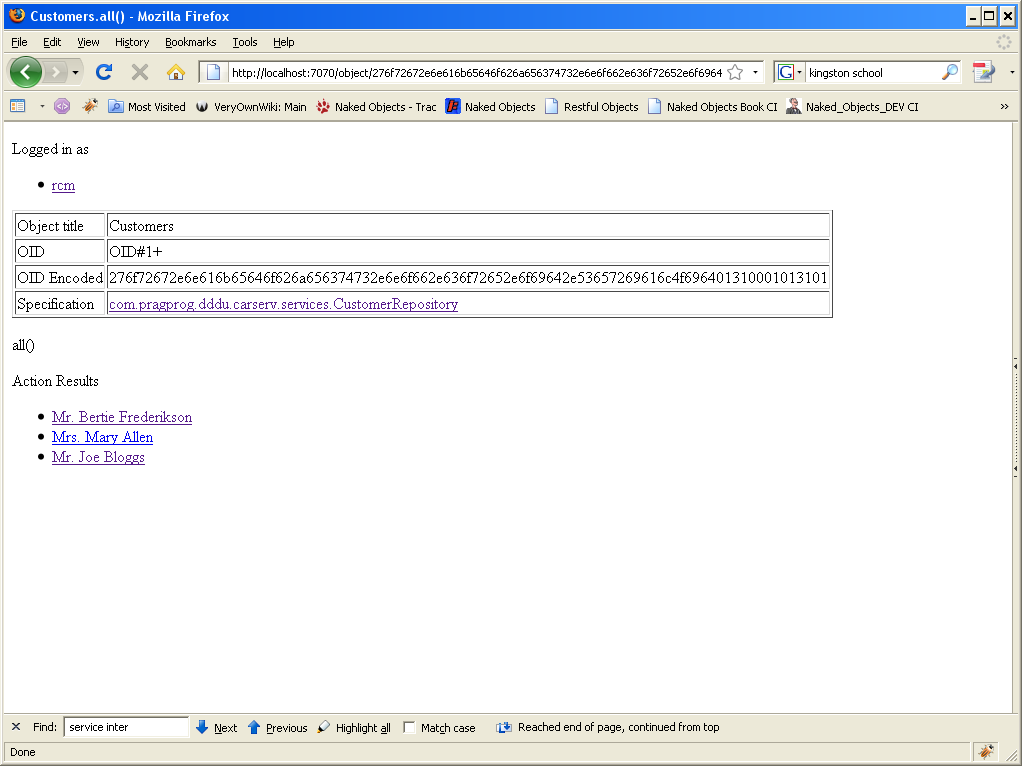
Scroll down to the customer’s “cars” collection ...

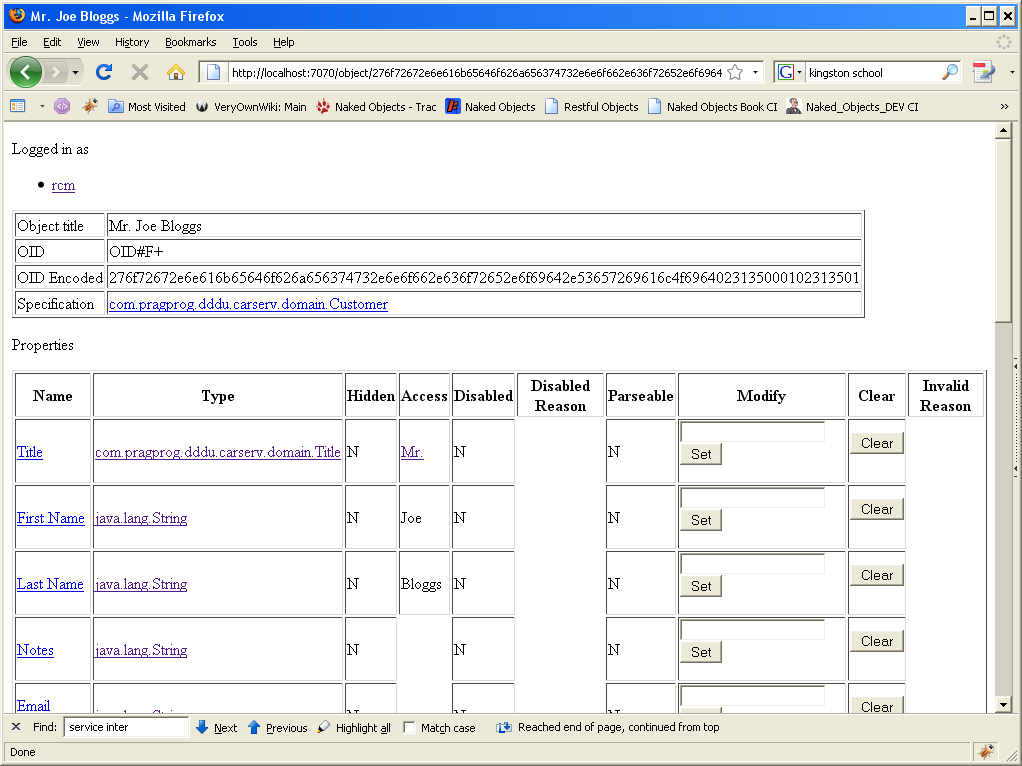


... and click to view the Cars in this collection:

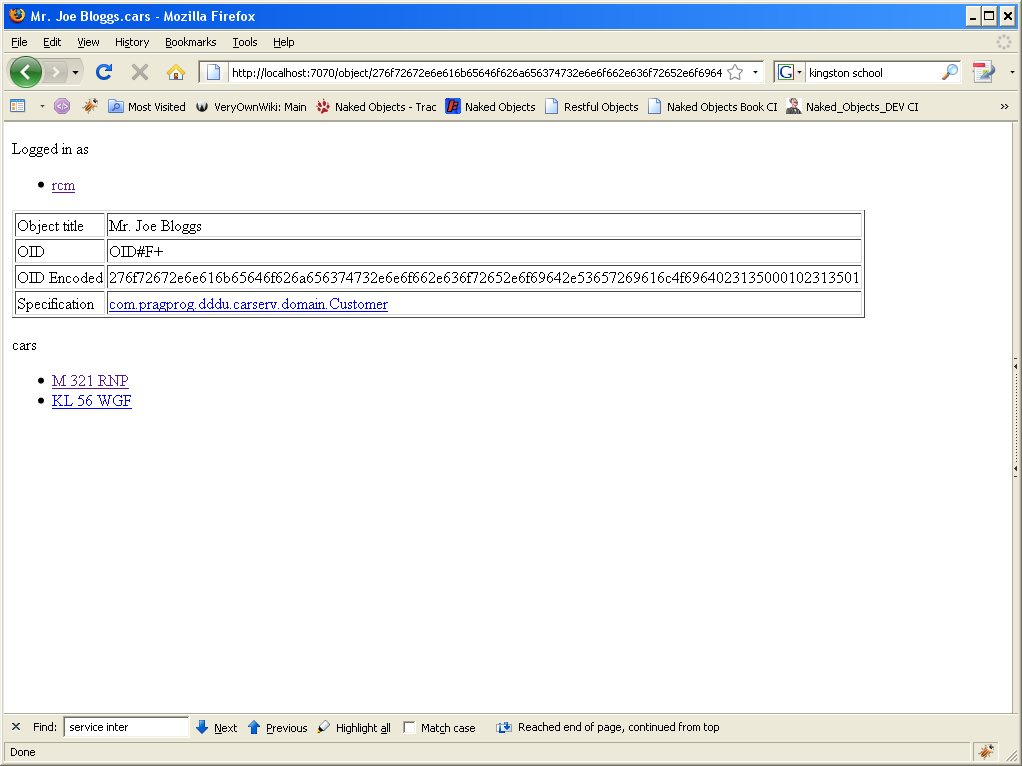
Copy the “OID Encoded” string into the clipboard.

Now, navigate back to the /services resource, and click on Customers, then Invoke All():

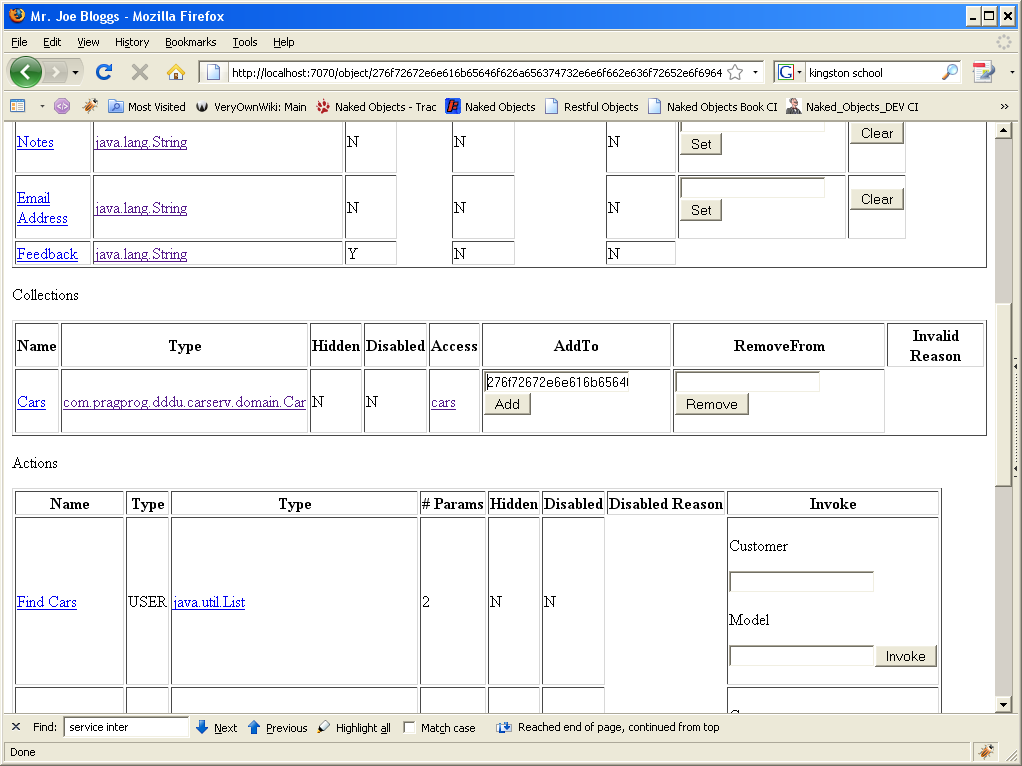


Select “Mr Joe Bloggs”:  


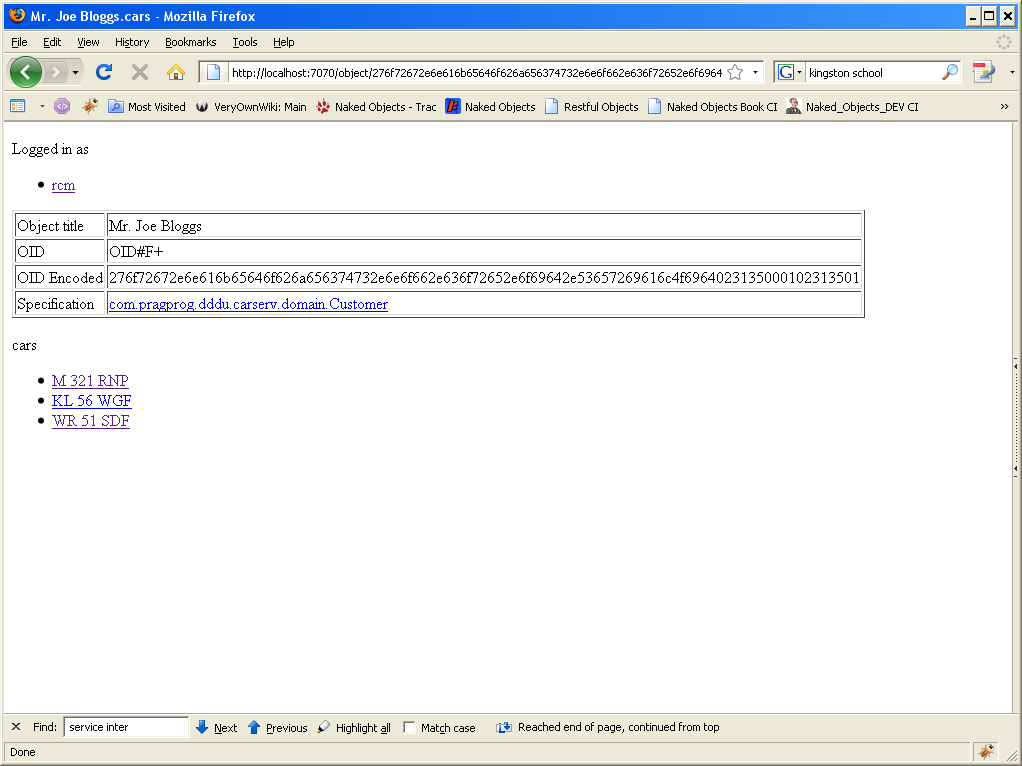
Scroll down to his cars collection, and click the link. Confirm there are two cars:



Navigate back to the customer.  
Scroll down again to the cars collection, and this time paste the encoded OID of “WR 51 SDF” into the addTo form. This performs a PUT, and then reloads the resource.



Scroll down to the “cars” collection, and click the link. There should now be 3 cars in the collection.



# Developing in Eclipse

### Coding Standards and so on

Restful Objects uses the same code formatting as Naked Objects. These are defined in terms of Eclipse config files, in .../framework-trunk/eclipse.

Import the Java formatting using:

Windows > Preferences > Java > Code Style > Formatter

and specify

.../framework-trunk/eclipse/java-format.xml

Import the cleanup profile (which references the Java formatting) using:

Windows > Preferences > Java > Code Style > Cleanup

and specify:

.../framework-trunk/eclipse/nakedobjects-cleanup-profile.xml

Currently there are no checkstyle, FindBugs or PMD checks.

### Hints and Tips

When you invoke a build in Eclipse, m2eclipse delegates to the Maven builder. You can use the Maven console to see the builder do its thing. Behind the scenes this is just an embedded instance of the Maven runtime, so the output is pretty much the same as running Maven from the command line.

If you perform a Maven build (mvn clean install) from the command line, then all modules are picked up from the local repository. m2eclipse sets up the classpath slightly differently so that all Maven modules that are in the workspace are referenced locally, and everything else is picked up from your local repository. In practice this means that you can develop in Eclipse pretty much the same as before.

It is worthwhile performing builds occasionally using Maven command line; there are sometimes slight differences between the Sun javac and Eclipse’s built-in Java compiler. If you perform a command-line build and it fails half-way, it’s usually sufficient in Eclipse to do an perform a Project > Clean All followed by a build workdspace. On occasion I’ve found though that this doesn’t work; M2Eclipse thinks that all class files are up-to-date and doesn’t build anything. To force M2Eclipse to rebuild everything, you can try to File > Refresh, and also Project > Update All Maven Dependencies. This pretty much almost always does the trick.

1. [www.nakedobjects.org](http://www.nakedobjects.org) [↑](#footnote-ref-2)
2. [www.scimpi.org](http://www.scimpi.org). Another sister project of Naked Objects, providing a collection of taglibs to build Naked Objects web applications. [↑](#footnote-ref-3)
3. <http://tortoisesvn.tigris.org> [↑](#footnote-ref-4)
4. <http://m2eclipse.codehaus.org/> and <http://subclipse.tigris.org/> [↑](#footnote-ref-5)
5. The timeandmoney library is hosted at: <http://sourceforge.net/projects/timeandmoney/>. [↑](#footnote-ref-6)