Adding a Web Service Step by Step Guide for Core Developers

This guide walks a core developer through adding a new set of web service methods for a core object. It is a similar process for module developers, but for that setup, see Adding a Web Service Step by Step Guide for Module Developers

Get the Code

The module can be checked out through git at https://github.com/openmrs/openmrs-module-webservices.rest

You can build the module like any other mavenized module by doing an *mvn clean install *in the root module package.

Read the Conventions

The first thing to do is to read through the conventions section on the REST Web Services API For Clients. This outlines the very basics of the how to name the urls, design the objects, etc.

> **Note:** When creating a REST resource and sub-resource URI and name respectively, the absolute and relative path names should be written in all-lowercase ASCII letters. Avoid capital letters, camel caps to mention a few. The norm is to use lower case letters.
>
> Example:
>
> /ws/rest/v1/conceptclass

Testing a URL

Through the webapp

With the webservices.rest module installed, there is a **Test** page linked to from on the **Administration** page.

From this page you can make jquery POST/PUSH/GET/DELETE web servces calls to the server and see the results.

The default values are to do a GET on a person. Check your database for a valid person UUID and try the query. (The post data does not matter in this case)

From command line with CURL

A valid curl call with authentication:

```
curl -u admin:password http://localhost:8080/openmrs/ws/rest/person
/b3f64e4c-860c-11e0-94eb-0027136865c4
```

Adding new Web Service URLs

> Our best-practices may change over the course as we learn new things, but ConceptResource1_8 should always be an up-to-date best-practice example that you can use as a pattern.

This example will use the **Location** object in the OpenMRS API.

<u>Create a new LocationResource1_8 object for the object (note the 1_8 postfix indicating the lowest version of OpenMRS this resource can work with)</u>
See also code for LocationResource1_8

The resource objects are used for the automatic translation into json/xml.

1. In the **org.openmrs.module.webservices.rest.web.v1_0.resource.openmrs1_8** package, create a new class that extends MetadataDelegatingCrudResource<Location>.
    1. public class LocationResource1_8 extends MetadataDelegatingCrudResource<Location>

        1. The MetadataDelegatingCrudResourceand DataDelegatingCrudResource class are helper classes for "metadata" and "data" OpenmrsObjects.
        2. The DelegatingCrudResource super class can be used for any other object type
        3. It's also possible to not have a resource class for some simple rest urls. See the SessionController class.
2. Add **@Resource(name = "v1/location", supportedClass = Location.class, supportedOpenmrsVersions = "1.8.**")* annotation to the class.
    1. This tells the framework to put the resource at uri location: /ws/rest/location/uuid
    2. By convention, resource names are all lower case.
    3. SupportedClass indicates to use this resource for the Location class and supportedOpenmrsVersions indicates to use this resource only when the module is running on OpenMRS 1.8.*.
    4. Optionally you can specify order, which will allow you to overwrite some other resource e.g. a module-provided resource for Location. Note that this will work only if you are using the same uri to access the resource you are overwriting.
3. Expose properties that are on the Location object through the resource:
    1. Create a method named "DelegatingResourceDescription **getRepresentationDescription**(Representation rep)"
    2. Switch on the "rep" argument (or use if/else)
    3. Create a DelegatingResourceDescription according to the rep, using addProperty for the properties that you need
    4. Tip: Use *description.addProperty("auditInfo", findMethod("getAuditInfo"))* for **not** creator/dateCreated/changedBy/dateChanged properties
    5. Tip2: the MetadataDelegatingCrudResource class defines the "ref" representation for you.
4. **DO NOT** add *locationId* to the getRepresentationDescription method. That is an internal number that should never be exposed over web services
5. Adding a new property (fullAddress) that is **not on the Location object** (only done here as an example. Patient.name is a better example)
    1. There is no Location.getFullAddress() method, so automatic translation of this into json/xml will fail in the webservices module.
    2. Add *description.addProperty("fullAddress", findMethod("getFullAddress"))* to the getRepresentationDescription method
    3. Add a method **SimpleObject getFullAddress(Location location)** on the **LocationResource** class.
        1. This method is called anytime a user requests to see the "fullAddress" property in the json/xml
    4. The implementation of the method can be anything using the Location object. Something like:
        1. return location.getAddress1() + " " + location.getAddress2() + " " + location.getCityVillage() + " " + location.getStateProvince()
6. Add a newDelegate() method
    1. This is used by the parent class to know how to construct our object (our delegate)
7. Add a save(Location location) method
    1. This is used by the parent class to know how to persist our object to the database. This is typically just Context.getSomeService().saveMyObject(object)
8. Add a getByUniqueId(String s) method
    1. The parent class calls this for all getters. You should look up your object by uuid. If your object can also be looked up by a unique name, you can do that in this method as well as the uuid.
    2. This method is what allows your object to be a spring "Converter". So other WS methods can refer to your object by uuid/name as well
    3. DO NOT look up by primary key. Primary keys should not exposed over web services
9. Add a purge(Location location, RequestContext context) method
    1. You can get the reason (if needed) out of the request context
10. Add a doGetAll(RequestContext context)
    1. This should return a list of every location objects that is still valid (not retired)
    2. The context object has parameters like "limit", "maxresults", "start", etc
11. Add doSearch(String query, RequestContext context) (see Adding Web Service Search Handler on how to implement custom searches)
    1. This should return a list of location objects with a fuzzy search on the query.
    2. The context object has parameters like "limit", "maxresults", "start", etc
    3. There is a helper class named ServiceSearcher that you can use to help with the paging, index, etc
12. If you have a non-standard setter method:
    1. **TODO**
13. Sub resources (like patient name)
    1. **TODO**