Documenting REST Resources

Before we get Started on how to get a Resource to be in the OpenMRS Swagger Documentation we should be comfortable with the concept of Resources and SubResources in OpenMRS.

1. **Resources** are OpenMRS objects (persons, patients, encounters, observations, etc.) which are exposed by the REST Web Services modules as a REST resources

2. **SubResources** are those resources(PersonNames, PersonAddresses, ConceptNames, etc) are not defined or do not make sense apart from their parent object. We refer to these as subresources

So now, for a resource to be appeared in OpenMRS Swagger Documentation, one should override these methods, based on its supported CRUD operations.

1. **Model getGETModel(Representation)**   A resource that supports ‘GET’ operations such as ‘getAll’, ‘getByUniqueId’, and ‘search’ should override this method.

2. **Model getCREATEModel(Representation)** A resource that supports ‘CREATE’ operations should override this method.

3. **Model getUPDATEModel(Representation)** A resource that supports ‘UPDATE’ operations should override this method.

You do not need to override these methods. However, if not, then their proper definitions won’t appear in OpenMRS Swagger specification. Thus the resource won’t show up in the live documentation here (while the live demo swagger documentation can be found inside the advanced system administration section of the reference application on the demo server) either.

So it’s recommended that you override these.

**How to find which properties to be documented?**

getGETModel(Representation), getCREATEModel(Representation), and getUPDATEModel(Representation) methods correspond to getGetRepresentation(), getCreatableProperties(), and getUpdatableProperties() methods respectively. So they can be used as reference when documenting resources.

One can also look into corresponding test classes and its supportedClass

Example

When documenting OrderSetResource1_12 one can use OrderSet.class, OrderSetResource1_12Test, and OrderSetController1_12Test as reference.

**Documenting GET representation of a resource**

**Model getGETModel(Representation)**

This method should return a Swagger Model object representing the GET representation schema for that resource. Returned Model may change depending on the representation type (DEFAULT, FULL, REF) being passed to that method.

If you don’t understand the difference between DEFAULT, FULL, and REF representations, please refer to:

REST Web Services API For Clients - Representations

Let's begin by documenting OrderSetResource1_12

Its DEFAULT representation returns a JSON object with the following properties:

```json
{ uuid, display, name, description, retired, operator, orderSetMembers }
```

- ‘uuid’, ‘display’, ‘name’, ‘description’ are just strings.
- ‘retired’ can be either true or false.
- ‘operator’ can take the values ALL, ONE, or ANY
- ‘orderSetMembers’ is an array containing REF representations of OrderSetMember resources.

Based on the type of its properties, we can document the getGETModel for OrderSet resource as follows:
public Model getGETModel(Representation rep) {
    ModelImpl model = (ModelImpl) super.getGETModel(rep);
    if (rep instanceof DefaultRepresentation || rep instanceof FullRepresentation) {
        model.property("operator", new EnumProperty(OrderSet.Operator.class));
    }
    if (rep instanceof DefaultRepresentation) {
        model.property("orderSetMembers", new ArrayProperty(new RefProperty("#/definitions/OrdersetOrdersetmemberGetRef")));
    } else if (rep instanceof FullRepresentation) {
        model.property("orderSetMembers", new ArrayProperty(new RefProperty("#/definitions/OrdersetOrdersetmemberGet")));
    }
    return model;
}

As you could see, the properties: uuid, display, name, description, and retired are not documented because they are inherited by the overridden method of its superclass. If they aren’t inherited, they should be added like:

```java
model
    .property("uuid", new StringProperty())
    .property("display", new StringProperty())
    .property("name", new StringProperty())
    .property("retired", new BooleanProperty());
```

The enum property 'operator' is documented as:

```java
model
    .property("operator", new EnumProperty(OrderSet.Operator.class));
```

The constructor of EnumProperty takes a Class of type Enum.

**Swagger Definition Naming**

When referencing to a GET representation of a resource, use:
- `ResourcenameGet` (for default representation)
- `ResourcenameGetRef`
- `ResourcenameGetFull`

When referencing to a CREATE representation of a resource:
- `ResourcenameCreate`

If the referencing resource is an `@SubResource`, the referencing name should be:
- `ParentnameResourcenameGet`
- `ParentnameResourcenameGetRef`
- `ParentnameResourcenameGetFull`
- `ParentnameResourcenameCreate`

Example

```java
.property("name", new RefProperty("#/definitions/ConceptNameGet"))
```

`@Resource` annotation can be used to identify resources.

**ConceptResource1_8.java**

```java
@Resource(name = RestConstants.VERSION_1 + "/concept", ...)
public class ConceptResource1_8
```
@SubResource annotation can be used to identify sub-resources.

```java
@SubResource(parent = ConceptResource1_8.class, path = "name", ..)
public class ConceptNameResource1_8
```

As shown in the example, the definition `name` of ConceptResource1_8 is Concept (identified by the `name` property its @Resource annotation). So the definition name of ConceptNameResource1_8 sub-resource should be ConceptName. As you see, the definition name of a sub-resource is retrieved by appending the value of the `path` property of its @SubResource annotation to its parent's name.

**RefProperty Type**

When one resource includes another resource as its property (e.g., conceptDatatype property on Concept object), the FULL representation of that property object is never included in the parent.

Eg :- DEFAULT representation of Concept resource includes the following reference properties:

- name, datatype, conceptClass

name is a ConceptNameResource, datatype is a ConceptDatatypeResource, conceptClass is a ConceptClassResource.

In DEFAULT representation of a Concept resource

- DEFAULT representation of name is included
- REF representation of datatype is included
- REF representation conceptClass is included

So the code would be like:

```java
model
    .property("name", new RefProperty("#/definitions/ConceptNameGet"))
    .property("datatype", new RefProperty("#/definitions/ConceptDatatypeGetRef"))
    .property("conceptClass", new RefProperty("#/definitions/ConceptClassGetRef"));
```

**How to test the made changes?**

Once the changed modules are deployed into your OpenMRS server, the changes can be checked from either:

- `/openmrs/module/webservices/rest/apiDocs.htm`
- `/openmrs/module/webservices/rest/swagger.json`

**Related articles**

- Documenting REST Resources
- Registration App Configuration
- Overriding Requests to Pages and Fragments
- OpenMRS SDK Step By Step Tutorials
- OpenShift Quickstart cartridge