Documenting REST Resources

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

- **Model getGETModel(Representation)**
  - A resource that supports GET operations such as 'getAll', 'getByUniqueId', and 'search'' should override this method.
- **Model getCREATEModel(Representation)**
  - A resource that supports CREATE operations should override this method.
- **Model getUPDATEModel(Representation)**
  - A resource that supports UPDATE operations should override this method.

It is not mandatory that you override these methods. However if not, their proper definitions won't appear in OpenMRS Swagger Spec. Thus the resource won't show up in the live documentation either. So it's recommended that you override these.

Prerequisite

These methods are introduced in openmrs-module-webservices.rest >= 2.20.0

So upgrade the openmrs-module-webservices.rest dependency version in your modules pom.xml if necessary.

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 following properties:

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

- *'uuid', 'display', 'name', 'description'* are just strings.
- *'retired'* can be either true or false
- *'operator'* can takes the values ALL, ONE, or 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:

OrderSetResource.java

```
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:

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

The enum property '*operator*' is documented as:

```
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

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

@Resource annotation can be used to identify resources.

ConceptResource1_8.java

```
@Resource(name = RestConstants.VERSION_1 + "/concept", …)
 public class ConceptResource1_8
```

@SubResource annotation can be used to identify sub-resources.

ConceptNameResource1_8.java

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

As shown in the example, 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 include 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 include 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:

```
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

- Registration App Configuration
- Setup Travis CI to Deploy Snapshots to Nexus
- Installing and Using OpenMRS Password Reset Open WebApp
- Documenting REST Resources
- OpenMRS SDK Step By Step Tutorials