REST Web Services API For Clients

This is the technical API documentation (focusing on client devs) for the REST Module. For general REST web service information and user documentation, see the module page.

Resources

Every available object in the web services (ws) module is written up as a resource. The resource class defines the properties that are exposed and the setters that are available. The class also defines the representations and what goes in them (ref vs default vs full, see below for more on representations).

See documentation about resources and their URIs here: REST Web Service Resources in OpenMRS 1.9

You can also see some generated URI documentation on the "Help" page that the module puts into its Administration page section.

There is a uri that lists off the currently available resources in the installation (including module-provided resources):

```
/module/webservices/rest/apiDocs.htm
```

Subresources

There are some objects that are not defined or do not make sense apart from their parent object. These we refer to as subresources. Examples are PersonNames, PersonAddresses, ConceptNames, etc. You can act on subresources under the parent url:
Examples:

Adding a person name

```
POST /ws/rest/v1/person/uuidofperson/name
Body content:
{"givenName": "John", "familyName": "Smith"}
```

Editing a person's name

```
POST /ws/rest/v1/person/uuidofperson/name/uuidofname
Body content:
{"givenName": "Johnny"}
```

A subresource can have only one parent. If it seems like an object has two or more parents, then it is most likely a top-level resource. E.g. "encounters" should not be a subresource of "patient" + "location" (answering questions of "all encounters of a patient" and "all encounters at a location"). Instead, these should be queries on the encounter resource: /ws/rest/v1/encounter?patient=349234-2349234 and /ws/rest/v1/encounter?location=3423482-34923-23

Resources with Subtypes

Some resources can have multiple subtypes, for example the `order` resource contains `drugorder` and `testorder` subtypes. When creating a resource that has subtypes via a POST, you must specify which subtype of the resource you are creating, with a special `t` property of the object. For example:

```
POST /ws/rest/v1/order
Body content:
{"t": "testorder", /*... and other properties */}
```

If you GET a resource that has subtypes, each result will be of one of those subtypes, which you can see by looking at the special `t` property of each result. You may query for only a certain subtype of a resource by providing a `t` query parameter, like `GET /ws/rest/v1/order?t=drugorder&otherparams=values`.

URI Conventions

Because there are so many options when creating REST urls, we have laid out a set of conventions that all REST developers should follow. This will keep our web service api looking neat and uniform across all different types of objects and modules.

- For *resource*,
  - GET /ws/rest/v1/*resource*?q=*query* = search
  - GET /ws/rest/v1/*resource*/*uuid* = retrieve
  - POST /ws/rest/v1/*resource* = create
    - Request body contains data to persist (not in request params)
  - POST /ws/rest/v1/*resource*/*uuid* = partial update of the resource
    - Request body contains just the fields to update (not in request params)
  - ~~PUT = replace value of entire object~~ (we don't use this yet)
  - DELETE /ws/rest/v1/*resource*/*uuid* = void for data, retire for metadata
  - DELETE /ws/rest/v1/*resource*/*uuid*?purge=true = purge (aka delete from the database entirely)
- Do not put verbs in the URL; represent them with sub-resources.
  - For example, instead of addNameToPerson, we POST to /ws/rest/person/*uuid*/names
- URLS:
  - Are prefixed with /ws/rest/ due to servlet forwarding in OpenMRS, followed by an API version number (currently /v1)
  - The URL should be written in all-lowercase ASCII letters
  - No special characters (e.g. no spaces or underscores)
  - Hyphens are ok if absolutely necessary
  - No extensions allowed (acceptHeader will be used to specify json or xml content. All json for release 1.0)
- Subresources should have their URIs inside the URI of their parent (like /ws/rest/person/*uuid*/name/*nameuuid)*
- Domain objects that are separately managed get their own URI. (e.g. /ws/rest/v1/enrollment/*uuid* instead of /ws/rest/v1/program/*uuid*/enrollments/*uuid)*
- Hide (don't expose) "helper" classes as much as possible (e.g. web service clients should never see ConceptSet, etc)
- Resource names should usually be the same as the domain objects they represent, but they may differ if the domain object name is confusing.
  - For example org.openmrs.PatientProgram is /ws/rest/v1/enrollment

E.g: When saving or editing a property on a "Concept" object, the conceptDatatype property can be simply the uuid. In addition, for most metadata, the "name" is unique across all active metadata, so that can also be used in place of the uuid when saving as well (POST or PUT).

Sample REST calls

If you are looking for some sample REST calls, please see Sample REST calls. You can also explore controller tests to see which requests are possible: https://github.com/openmrs/openmrs-module-webservices.rest/search?q=Controller+test&type=Code

Limiting the number of results and paging

When you do a GET request that returns a very large number of results (either by getting all instances or doing a search), the number of results returned is limited.

Clients may request more or fewer results with the *limit* parameter. For example:

```
curl -u admin:test -i 'http://localhost:8080/openmrs/ws/rest/v1/concept?limit=2'
```

The above will return only 2 concepts. To retrieve the next page of two concepts, use the *startIndex* tag. For example:

```
curl -i 'https://localhost:8443/openmrs/ws/rest/v1/concept?limit=2&startIndex=2'
```

As a system administrator you can configure both the default limit (applied whenever the client does not specify a limit) and the absolute limit (which the client is not allowed to exceed), from the "Settings" page of the ws module.

Including Retired metadata and Voided/Deleted data

A normal REST query will not include Retired metadata or Deleted/Voided data. You may send an `includeAll=true` query parameter to include retired /deleted data.

Both the REST and Java API follow the same default behaviour except for the case of getting metadata from the Java API, see the summary table below:

REST API	Java API
`GET dataresource` **excludes** voided by default	`getAll[Data]` **excludes** voided by default
`GET metadataresource` **excludes** retired by default	`getAll[Metadata]` *includes* retired by default

Date/Time with Time Zones

It is strongly recommended to always submit the full date + time + timezone in any REST POST query.

The format of the date/time should be:

"**2016-12-25T19:02:34.232+0700**"

(the milliseconds are optional)

e.g.

Creating an obs

```
POST /ws/rest/v1/obs
Body content:
{"obsDatetime": "2016-12-25T19:02:34.232+0700", /*... and other properties */}
```

If no time zone is provided or it is not formatted correctly, the service will suppose that the date and time are provided at local time (server time). This can have deep repercussions, so make sure to always send a valid time zone.

Response Format

Spring allows us to offer both json and xml formats for every object. If you want to receive a json response, be sure to insert a header of "Accept: application/json". If xml is your thing, use "Accept: application/xml".

When POSTing content, use the "Content-Type" header to specify the data format you are sending to the server.

⚠ Browsers have their Accept header fixed at: Accept: text/html,application/xhtml+xml,application/xml;. Therefore you will always see xml returned in a browser.

Representations

The objects returned by web service calls are variable in their properties. In general there are three different representations that exist: ref, default, full. It is possible for modules to provide more representations for their objects. (TODO: document what method to call to get the available reps)

To change between representations, use the "v" query parameter: `...?v=ref` or `...?v=full`. Using `...?v=default` is invalid. Simply leave that parameter off.

Web service calls that return lists of objects will put those objects into the "ref" representation.

Ref

When an object is a child resource on another object (e.g. conceptDatatype property on Concept object), the full ConceptDatatype object is not returned by default. Instead, a "ref" kind of class with String properties for `uuid`, `links`, and a `display` fills the concept.conceptDatatype property. The ref looks like this:

Ref example

```
concept.conceptDatatype \->
{
display: "Numeric",
uuid: "8d4a4488-c2cc-11de-8d13-0010c6dffd0f",
links: {
  self: "http:/../openmrs/ws/rest/v1/conceptdatatype/8d4a4488-c2cc-11de-8d13-0010c6dffd0f"
  }
}
```

Custom Representations

(Note: requires version 1.1 of the module)

You can set the "v" parameter to a string that explicitly states all of the reps for the class and children classes:
e.g.

```
v=custom:(uuid,datatype:(uuid,name),conceptClass,names:ref)
```

fuller explanation:
the main resource will return:

- uuid property
- conceptClass property as default rep
- names as ref rep
- datatype will have custom rep:
    - uuid property
    - name property

To fetch the full ConceptDatatype data, a second call to the links.self uri: `http:/../openmrs/ws/rest/v1/conceptdatatype/8d4a4488-c2cc-11de-8d13-0010c6dffd0f` is needed.

Default

This representation is returned for objects when there is no "v=" parameter given. Most properties will be included and "refs" of some subobjects will also be listed.

Full

The full representation is meant to be used when subsequent calls are not desired or some uncommonly needed properties (like audit info) are needed.

Authentication

- Nearly every method (other than the /session endpoint) in the OpenMRS API requires authentication; therefore, every method in the webservices module needs to have an authenticated user in order to work.
- There is a filter defined on the module that intercepts *all calls* and authenticates the given request.
- Currently, only BASIC authentication is supported. Along with the HTTP request, a request header of **Authorization: Basic (base64 of username:password)** needs to be sent.
- Alternatively, a session token can be used. **GET /openmrs/ws/rest/v1/session** with the BASIC credentials will return the current token value. This token should be passed with all subsequent calls as a cookie named jsessionid=*token*.
- A DELETE call on the /openmrs/ws/rest/v1/session will logout the user and end the session.

Getting all location without authentication

Fetching locations requires you to authenticate the user first. If it is a required to display all locations prior to login or without authentication you can do it by using **GET .../openmrs/ws/rest/v1/location?tag=Login%Location**

Changing your password

Since version 2.17 of the webservices.rest module:

- After authenticating you can change your own password, by **POST .../openmrs/ws/rest/v1/password** with **oldPassword** and **newPassword** in the request body.
- An administrator (with the **EDIT_USER_PASSWORDS** privilege) can change the password for other users by **POST .../openmrs/ws/rest/v1/password/<uuidOfOtherUser>** with **newPassword** in the request body.

Links

A resource can have any number of "links" in the links attribute. Generally, these will be links to other representations (see above), but they also could be to other types of relationships between objects: things like "parent", etc.

A "ref" representation will contain a link to "self" that is the default rep. A "default" rep will have a link to "full" that is the same object except with all properties included.

Audit Info

The full representation of object contains a property called auditInfo. You can find the creator, changer, and voider/retirer information here.

auditInfo

`creator`
`dateCreated`
`changedBy`
`dateChanged`

If the object is data and so is voidable, you will additionally see:
voidedBy
dateVoided
voidReason

If the object is metadata and so is retirable, you will see:
retiredBy
dateRetired
retireReason

The "retired" and "voided" boolean properties are on the ref/default/full representations. The ref rep will only contain the property if the object if isVoided /isRetired returns true.

Versioning

The OpenMRS API will change over time and this will necessitate the rest representations and urls to be modified as well. The first release of the module will be version "v1". The entire rest api is versioned with this same number:

/ws/rest/v1/patient/2342-34-DDD-23-2ADF
/ws/rest/v1/concept/AFD21-239D3-5233234
etc

If/when (hopefully never) we make a major change to the entire rest setup, we will change this to v2. (e.g. change from json to kson)

Each resource can also refer to different underlying api object versions. Each object declares a resourceVersion property that specifies which version of the API it is required to work in. This works similar to an @since annotation.

The actual visible version of the '"module"' will be incremented independenty from openmrs AND from the version of the rest api.

There were discussions about using the Media-Type header instead of a number in the uri. However, looking at a lot of major players in the api space, we decided to go with a global api version in the uri.

ETag

(available with OpenMRS v1.8.1 or higher due to the Spring Framework 3.0.5 requirement)

The module also adds an ETag to response headers when presenting resources to clients. ETags have been implemented in shallow-mode (i.e. they save client bandwidth, but not server-side processing). As described here, as a client application you may want to look at ETag when making REST calls.

From the wikipedia article:

> *In typical usage, when a URL is retrieved the web server will return the resource along with its corresponding ETag value, which is placed in an HTTP "ETag" field:*
>
> ```
> ETag: "686897696a7c876b7e"
> ```

> *The client may then decide to cache the resource, along with its ETag. Later, if the client wants to retrieve the same URL again, it will send its previously saved copy of the ETag along with the request in a "If-None-Match" field.*
>
> ```
> If-None-Match: "686897696a7c876b7e"
> ```
>
> *On this subsequent request, the server may now compare the client's ETag with the ETag for the current version of the resource. If the ETag values match, meaning that the resource has not changed, then the server may send back a very short response with an HTTP 304 Not Modified status. The 304 status tells the client that its cached version is still good and that it should use that. However, if the ETag values do not match, meaning the resource has likely changed, then a full response including the resource's content is returned, just as if ETags were not being used. In this case the client may decide to replace its previously cached version with the newly returned resource and the new ETag.*

Example using curl:

```
$ curl -i -u admin:test http://127.0.0.1:8080/openmrs/ws/rest/v1/patient?q=Dar
```

RESPONSE HEADER IS:

```
HTTP/1.1 200 OK
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: JSESSIONID=1jw2itu9oyt5v;Path=/openmrs
Content-Type: application/json;charset=UTF-8
*ETag: "078c5b8fe25b332a40b4174bd38f5ee90"*
Content-Length: 399
Server: Jetty(6.1.10)
```

```
$ curl -i -u admin:test http://127.0.0.1:8080/openmrs/ws/rest/v1/patient?q=Darius
```

RESPONSE HEADER IS:

```
HTTP/1.1 200 OK
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: JSESSIONID=1jw2itu9oyt5v;Path=/openmrs
Content-Type: application/json;charset=UTF-8
*ETag: "078c5b8fe25b332a40b4174bd38f5ee90"*
Content-Length: 399
Server: Jetty(6.1.10)
```

We see that both the ETag are same and hence the client knows that the response would be the same and can save bandwidth.

After some days, if Darius's records have not been updated then, the client can send the ETag and check for modifications:

```
$ curl -i -H 'If-None-Match:"078c5b8fe25b332a40b4174bd38f5ee90"' -u admin:test http://127.0.0.1:8080/openmrs/ws
/rest/v1/patient?q=Darius
```

RESPONSE IS:

```
HTTP/1.1 304 Not Modified
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: JSESSIONID=dlcztsmushgm;Path=/openmrs
Content-Type: application/json;charset=UTF-8
ETag: "078c5b8fe25b332a40b4174bd38f5ee90"
Content-Length: 0
Server: Jetty(6.1.10)
```

From the **304 Not Modified**, we know that the records are the same and the client doesn't have to get the data again.