REST API Guide

Table of Contents

The DataNucleus REST API provides a RESTful interface to persist JSON objects to the datastore. All entities are accessed, queried and stored as resources via well defined HTTP methods. This API consists of a **servlet** that internally handles the persistence of objects (using JDO). Your POJO classes need to be accessible from this servlet, and can use either JDO or JPA metadata (annotations or XML). The REST API automatically exposes the persistent class in RESTful style, and requires minimum configuration as detailed in the sections linked below.

Servlet Configuration

The configuration of the REST API consists in the deployment of jar libraries to the CLASSPATH and the configuration of the servlet in the */WEB-INF/web.xml*. After it's configured, all persistent classes are automatically exposed via RESTful HTTP interface. You need to have enhanced versions of the model classes in the CLASSPATH.

Libraries

DataNucleus REST API requires the libraries: `datanucleus-core.jar`, `datanucleus-api-rest.jar`, `datanucleus-api-jdo.jar`, `javax.jdo.jar` as well as `datanucleus-rdbms.jar` (or whichever datastore you wish to persist to if not RDBMS). You would also require `datanucleus-api-jpa.jar` and `javax.persistence.jar` if using JPA metadata (XML/annotations) in your model classes.

In WAR files, these libraries are deployed under the folder */WEB-INF/lib/*.

web.xml

The DataNucleus REST Servlet class implementation is *org.datanucleus.api.rest.RestServlet*. It has to be configured in the `/WEB-INF/web.xml` file, and it takes one initialisation parameter.

Parameter	Description
persistence-context	Name of a PMF (if using `jdoconfig.xml`), or the name of a persistence-unit (if using `persistence.xml`) accessible to the servlet

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">

    <servlet>
        <servlet-name>DataNucleus</servlet-name>
        <servlet-class>org.datanucleus.api.rest.RestServlet</servlet-class>
        <init-param>
            <param-name>persistence-context</param-name>
            <param-value>myPMFName</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>DataNucleus</servlet-name>
        <url-pattern>/dn/*</url-pattern>
    </servlet-mapping>

    ...
</web-app>
```

changing *myPMFName* to the name of your PMF, or the name of your persistence-unit, and changing */dn/** to the URL pattern where you want DataNucleus REST API calls to be answered.

HTTP Methods

The persistence to the datastore in your application is performed via HTTP methods as following:

Method	Operation	URL format	Return	Arguments
POST	Insert object	/{full-class-name}	The JSON Object is returned.	The JSON Object is passed in the HTTP Content.
PUT	Update object	/{full-class-name}/{primary key}	The JSON Object is returned.	The JSON Object is passed in the HTTP Content. The primary-key is specified in the URL if the PK is application-identity single field or if it is datastore-identity
DELETE	Delete object	/{full-class-name}/{primary key}		The primary key fields are passed in the HTTP Content (JSONObject) if the PK uses multiple PK fields, otherwise in the URL.
DELETE	Delete all objects of type	/{full-class-name}		
GET	Fetch all objects of type	/{full-class-name}[?fetchGroup={fetchGroupName}][&maxFetchDepth={depth}]	JSON Array of JSON objects	
GET	Fetch a single object	/{full-class-name}/{primary key}[?fetchGroup={fetchGroupName}][&maxFetchDepth={depth}]	A JSON object	The primary key fields are passed in the HTTP Content (JSONObject) if the PK uses multiple PK fields, otherwise in the URL
GET	Query objects via a filter. Returns a JSON Array of objects	/{full-class-name}?[filter={filter}][&fetchGroup={fetchGroupName}][&maxFetchDepth={depth}]	JSON Array of JSON objects	Filter component of a JDOQL query
GET	Query objects via JDOQL. Returns a JSON Array of objects	/jdoql?query={JDOQL-single-string-query}[&fetchGroup={fetchGroupName}][&maxFetchDepth={depth}]	JSON Array of JSON objects	JDOQL single string query
GET	Query objects via JPQL. Returns a JSON Array of objects	/jpql?query={JPQL-single-string-query}[&fetchGroup={fetchGroupName}][&maxFetchDepth={depth}]	JSON Array of JSON objects	JPQL single string query

Method	Operation	URL format	Return	Arguments
HEAD	Validates if an object exists	/{full-class-name}/{primary key}		The primary key fields are passed in the HTTP Content (JSONObject) if the PK uses multiple PK fields, otherwise in the URL
HEAD	Validates if an object exists	/{full-class-name}?filter={filter}		Object is uniquely defined using the filter.

Example REST Usages

Note that the URL in all of these examples assumes you have "/dn/*" in your *web.xml* configuration.

Insert a new object of class using application identity

```
This inserts a Greeting object. The returned object will have the "id" field set.
```

```
POST http://localhost/dn/mydomain.Greeting
{"author":null,
 "content":"test insert",
 "date":1239213923232}
```

Response:

```
{"author":null,
 "content":"test insert",
 "date":1239213923232,
 "id":1}
```

Insert a new object with related (new) object

This inserts a User object and an Account object (for that user).

```
POST http://localhost/dn/mydomain.User
{"id":"bert",
 "name":"Bert Smith",
 "account":{"class":"mydomain.model.SimpleAccount",
        "id":1,
        "type":"Basic"}
}
```

Note that the "class" attribute specified for the related object is an artificial discriminator so that DataNucleus REST knows what type to persist on the server. If the Account type (referred to by User.account) has no subclasses then "class" is not required and it will persist an Account object.

Insert a new object of class using datastore identity

This inserts a Person object. The returned object will have the "_id" property set.

```
POST http://localhost/dn/mydomain.Person
{"firstName":"Joe",
 "lastName":"User",
 "age":15}
```

Response:

```
{"firstName":"Joe",
 "lastName":"User",
 "age":15,
 "_id":2}
```

Update an object of class using application identity

This updates a Greeting object with id=1, updating the "content" field only.

```
PUT http://localhost/dn/mydomain.Greeting/1
{"content":"test update"}
```

Update an object using datastore identity

This updates a Person object with identity of 2, updating the "age" field only.

```
PUT http://localhost/dn/mydomain.Person/2
{"age":23}
```

Fetch all objects of class using application identity

This gets the Extent of Greeting objects.

```
GET http://localhost/dn/mydomain.Greeting
```

Response:

```
[{"author":null,
  "content":"test",
  "date":1239213624216,
  "id":1},
 {"author":null,
  "content":"test2",
  "date":1239213632286,
  "id":2}]
```

Fetch object with id 2 using datastore identity

```
GET http://localhost/dn/mydomain.Person/2
```

Response:

```
{"firstName":"Joe",
 "lastName":"User",
 "age":23,
 "_id":2}
```

Note that it replies with a JSONObject that has "_id" property representing the datastore id.

Query object of class using application identity

This performs the JDOQL query internally

```
SELECT FROM mydomain.Greeting WHERE content == 'test'
```

```
GET http://localhost/dn/mydomain.Greeting?content=='test'
```

Response:

```
[{"author":null,
  "content":"test",
  "date":1239213624216,
  "id":1}]
```

Fetch object using Application PrimaryKey Class (JSON)

```
GET
http://localhost/dn/google.maps.Markers/{"class":"com.google.appengine.api.datastore.K
ey","id":1001,"kind":"Markers"}
```

Response:

```
{"class":"google.maps.Markers",
 "key":{"class":"com.google.appengine.api.datastore.Key",
        "id":1001,
        "kind":"Markers"
       },
 "markers":[
            {"class":"google.maps.Marker",
             "html":"Paris",
             "key":{"class":"com.google.appengine.api.datastore.Key",
                    "id":1,
                    "kind":"Marker",
                    "parent":{"class":"com.google.appengine.api.datastore.Key",
                              "id":1001,
                              "kind":"Markers"
                             }
                   },
             "lat":48.862222,
             "lng":2.351111
            }
           ]
}
```