OAGi JSON Schema Naming and Design Rules (NDR) and REST Usage Design Guide

Authors:
Michael Rowell, Oracle, OAGi
David Connelly, OAGi

Contributors:
Scott Nieman, Land O'Lakes
Steffen Fohn, ADP
Frank Heinrich, iBASEt
Kurt Kanaskie, Invited Expert

OAGi Document Version 0.04
As of March 12, 2015

NOTICE

Table of Contents

OAGi JSON Schema Naming and Design Rules (NDR)

Abstract

The Open Applications Group Integration Specification (OAGIS) provides a canonical business language for vertical industries. Individual organizations and entire supply chains may further extend the specification in ways that meet their own unique needs. It is important for OAGi to define the naming, design rules and guidelines used for OAGIS in such a manner that these organizations may follow them for their extension.

This specification provides a means to identify, capture and maximize the re-use of business information components within OAGIS and OAGIS extensions in order to support information interoperability across integrated environments.

Thank you to all who have contributed to the design, construction, and reviewing of the document. If we have missed anyone in our credits, we apologize to you.

This document will continue to grow as more details are added and updated.

Audience

The intended audience for this document is someone who has prior experience with OAGIS XML Naming and Design Rules. This document references the UN/CEFACT Naming and Design Rules 3.0.

Introduction

OAGIS JSON Schema NDR Basis

The OAGIS JSON Schema uses the UN/CEFACT Naming and Design Rules 3.0 as its basis.

JSON Schema differences from XSD NDR

OAGi - Naming Design Rules and differences for JSON

- Best practices for XML is to use UpperCamelCase for XML elements. Best practices for JSON is to use lowerCamelCase for JSON attributes.
    - This is matter of personal preference for the given developers. In order to be accepted by the developer community the preference must be followed.
    - The Working Group surveyed best practices and LowerCamelCase was chosen because many of the most prominent JSON implementations including Google,
- NOTE: Go get the list Scott sent out - before finalizing this decision
-
- XML has elements and attributes. JSON has only attributes.
    - All content will be expressed in JSON attributes meaning no distinction of level in how Supplementary Components and Content Components are represented. However, values will be expressed as the base data type in which they are defined.
- Examples of these differences:
  XML: <TotalAmount unitCode=”Each”>486<TotalAmount>
  JSON: {“totalAmount” : {“unitCode” : “Each”, amount : “486”}}

In JSON the convention is to use plurals form of the element tag for elements that have cardinality greater than 1. (Note this does always mean ending in ‘s’.)

OAGi endorses the OData URI convention for resource paths and query options, reflecting the goals of the Richardson Maturity Model.

OAGIS JSON Schema Architecture

The BOD architecture is NOT planned to be used at this time. The focus is on using Nouns and Components for the data definitions.

The Verbs used for OAGIS JSON Schema API's or messages uses the REST mechanism verbs in lieu of the OAGIS Verbs. The Verb guidelines are below.

OAGIS REST Verb Reference

This section is meant to be a short guide to implementing the OAGIS Verbs as transactions. We start with the Verb definitions, then describe applying them in design patterns.

Assumption: OAGIS will start with the base REST Verbs, GET, PUT, DELETE, POST
Definitions of each base verb below:

REST Verb	**Description**
POST	The POST is used to create a resource.
GET	The GET verb is used to read a resource. An important rule of thumb is that a GET operation is *safe*. That is, it can be done repeatedly without changing visibly the state of the resource. This property is very important for various reasons. First, indexing engines use GET to index the contents of a resource. So it would be bad if indexing a resource also changed it. Second, intermediaries, such as proxies, may cache results of a GET operation to accelerate subsequent accesses to the same resource.
PUT	The PUT and DELETE verbs allow a request to alter the state of a resource atomically. The PUT and DELETE verbs give us a simple mechanism to replace or destroy a resource. Note that PUT and DELETE apply to the entire resource and not just parts of it. So, when doing a PUT operation, the entire resource is replaced. This detail is very important. Important enough to be repeated: PUT acts on the entire resource! The same is true for DELETE: DELETE acts on the entire resource!

	The PUT and DELETE operations are atomic. If two PUT operations occur simultaneously, one of them will win and determine the final state of the resource. The same is true when a PUT and DELETE operation occur simultaneously. Either the resource's final state is updated or it is deleted, but nothing in between. In the case of two simultaneous DELETE operations, the order does not matter, because deleting a resource again has no effect. If the caller provides the ID of the resource the PUT is used to put the "updated" object. If the object does not exist create it. Created or replaced by the state of the representation payload. Put return the 201 created response. Identification of the object/resource is done by assigning an id.
DELETE	See PUT definition.

Source:
http://developer.mindtouch.com/REST/REST_for_the_Rest_of_Us

A short summary of examples:

```
/api/users     when called with GET, lists users
/api/users      when called with POST, creates user record
/api/users/1    when called with GET, shows user record
            when called with PUT, updates user record
                        when called with DELETE, deletes user record
```

Source: http://stackoverflow.com/questions/2001773/understanding-rest-verbs-error-codes-and-authentication

http://tools.ietf.org/html/draft-ietf-httpbis-p2-semantics-20#page-14

OAGIS Verbs and their REST counterparts

Responses are not included for the REST Verbs because there are no responses in REST based processing.

OAGIS Verb	**REST Verb**
Process (with Add)	POST
Process (Replace)	PUT
Process (Delete)	DELETE
Process (Change)	No Equivalent
Sync (with options selected for Add, Change, Delete, or replace)	POST or PUT, depending on who owns the data
Sync (with Add)	POST
Sync (Replace)	PUT
Sync (Delete)	DELETE
Sync (Change)	No Equivalent
Post (synonym for Process in financial scenarios)	POST
Load (Synonym for Sync in financial scenarios)	POST
Change	PUT
Update	PUT
Cancel	PUT for setting a status or marking for Delete; DELETE physically removes the record via the URI. The data has no meaning. If using URI resource use Delete else if using Data Model use PUT For now PUT
Get Note: Show can be used as an operational response to a Get	GET
Notify	Not applicable. This kind of processing is not compatible with a RESTful environment.

	Long Polling? Look it up and add a definition. Every Notification is a new thing – It should be a POST. The Notify has Add, Change, Replace, and Delete indicating that it is like the Process/Sync above. In the RESTful world a Notification is now a Noun, not a verb. So one would POST to a Notification.
Notify (with Add)	POST
Notify (Replace)	PUT
Notify (Delete)	DELETE
Notify (Change)	No Equivalent

This approach drives us to a unique ID instead of the different IDs used today for object integration. What is the identifying URI for the object. A URI with an ID for resources in a collection.