talis

Home

About The Talis Platform

- Contact Us
- Service Checklist
- Terms And Conditions

Augment Service Content Negotiation

Getting Started

- Authentication
- Authentication FAQ
- Bounded Descriptions In RDF
- Capabilities
- Changeset Protocol
- Changesets
- Client Libraries
- Constructing A Changeset
- Field Predicate Map
- Introduction to the Platform
- Platform FAQ
- Query Profile
- SPARQL
- SPARQL Extensions
- Store Configuration
- Store Naming
- Use of REST

Platform API

- API Site Map
- Augment Service
- Contentbox
- Full Text Search Syntax
- Full Text Searching
- Metabox
- Output Types
    - Content Negotiation
    - JSONP Content Negotiation
    - **RDF JSON**
    - RDF PHP
- Scheduled Job
- Store
- Store Access Status
- Store Facet Service
- Store Field Predicate Map
- Store MultiSPARQL Service
- Store OAI Service
- Store Query Profile
- Store Snapshot Collection
- Store SPARQL Service
- XSLT Service

Sitemap

Platform API > Output Types >

RDF JSON

JSON (the serialisation of data in javascript object notation) is an increasingly popular data format, largely because it is easy to parse (or, in the case of java
structure of the consumer's programming language of choice.

This is a specification for a resource-centric serialisation of RDF in JSON. It aims to serialise RDF in a structure that is easy for developers to work with.

Syntax Specification

RDF/JSON represents a set of RDF triples as a series of nested data structures. Each unique subject in the set of triples is represented as a key in JSON objec
dictionary or hash table). The value of each key is a object whose keys are the URIs of the properties associated with each subject. The value of each propert
representing the value of each property.

Blank node subjects are named using a string conforming to the nodeID production in Turtle . For example: _:A1

In general, a triple (subject **S**, predicate **P**, object **O**) is encoded in the following structure:

```
{ "S" : { "P" : [ O ] } }
```

The object of the triple **O** is represented as a further JSON object with the following keys:

type
: one of 'uri', 'literal' or 'bnode' (**required** and must be lowercase)

value
: the lexical value of the object (**required**, full URIs should be used, not qnames)

lang
: the language of a literal value (**optional** but if supplied it must not be empty)

datatype
: the datatype URI of the literal value (**optional**)

The 'lang' and 'datatype' keys should only be used if the value of the 'type' key is "literal".

For example, the following triple:

```
<http://example.org/about> <http://purl.org/dc/elements/1.1/title> "Anna's Homepage" .
```

can be encoded in RDF/JSON as:

```
{
  "http://example.org/about" :
    {
       "http://purl.org/dc/elements/1.1/title": [ { "type" : "literal" , "value" : "Anna's Homepage" } ]
    }
}
```

Here is an example of the RDF JSON specification in the format of a JSON Schema . The latest version can also be found in the schema section of the SOA

```
{
     "version":"0.3.0",
     "id":"RDF-JSON",
     "description":"RDF JSON definition",
     "type":"object",
     "properties":{
     },
     "additionalProperties":{
         "type":"object",
         "description":"subject (root object)",
         "optional":"true",
         "properties":{
         },
         "additionalProperties":{
             "type":"array",
             "description":"predicate (subject object)",
             "optional":"true",
             "items":{
                 "type":"object",
                 "description":"object (value array)",
                 "properties":{
                     "description":"content (value object)",
                     "type":{
                         "type":"string",
                         "enum":["uri","bnode","literal"]
                     },
                     "value":{
                         "type":"string"
                     },
                     "lang":{
                         "optional":true,
                         "description":"See ftp://ftp.isi.edu/in-notes/bcp/bcp47.txt",
                         "type":"string"
                     },
                     "datatype":{
                         "optional":true,
                         "format":"uri",
                         "type":"string"
                     }
                 }
             }
         }
     }
}
```

Examples

The following RDF/XML:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://example.org/about">
    <dc:creator>Anna Wilder</dc:creator>
    <dc:title xml:lang="en">Anna's Homepage</dc:title>
    <foaf:maker rdf:nodeID="person" />
  </rdf:Description>
  <rdf:Description rdf:nodeID="person">
    <foaf:homepage rdf:resource="http://example.org/about" />
    <foaf:made rdf:resource="http://example.org/about" />
    <foaf:name>Anna Wilder</foaf:name>
    <foaf:firstName>Anna</foaf:firstName>
    <foaf:surname>Wilder</foaf:surname>
    <foaf:depiction rdf:resource="http://example.org/pic.jpg" />
    <foaf:nick>wildling</foaf:nick>
    <foaf:nick>wilda</foaf:nick>
    <foaf:mbox_sha1sum>69e31bbcf58d432950127593e292a55975bc66fd</foaf:mbox_sha1sum>
  </rdf:Description>
</rdf:RDF>
```

Can be represented as the following RDF/JSON structure:

```
{
    "http://example.org/about" : {
        "http://purl.org/dc/elements/1.1/creator" : [ { "value" : "Anna Wilder", "type" : "literal" } ],
        "http://purl.org/dc/elements/1.1/title"   : [ { "value" : "Anna's Homepage", "type" : "literal", "lang" : "en" } ] ,
        "http://xmlns.com/foaf/0.1/maker"         : [ { "value" : "_:person", "type" : "bnode" } ]
    } ,

    "_:person" : {
        "http://xmlns.com/foaf/0.1/homepage"      : [ { "value" : "http://example.org/about", "type" : "uri" } ] ,
        "http://xmlns.com/foaf/0.1/made"          : [ { "value" : "http://example.org/about", "type" : "uri" } ] ,
        "http://xmlns.com/foaf/0.1/name"          : [ { "value" : "Anna Wilder", "type" : "literal" } ] ,
        "http://xmlns.com/foaf/0.1/firstName"     : [ { "value" : "Anna", "type" : "literal" } ] ,
        "http://xmlns.com/foaf/0.1/surname"       : [ { "value" : "Wilder", "type" : "literal" } ] ,
        "http://xmlns.com/foaf/0.1/depiction"     : [ { "value" : "http://example.org/pic.jpg", "type" : "uri" } ] ,
        "http://xmlns.com/foaf/0.1/nick"          : [
                                                      { "type" : "literal", "value" : "wildling"} ,
                                                      { "type" : "literal", "value" : "wilda" }
                                                    ] ,
        "http://xmlns.com/foaf/0.1/mbox_sha1sum"  : [ {  "value" : "69e31bbcf58d432950127593e292a55975bc66fd", "type" : "literal" } ]
    }
}
```

Serialisation Algorithm

Refer to http://json.org/ for definitions of terminology

1. Start a JSON object (called the root object)
2. Group all the triples by subject
3. For each subject:
    1. Create a JSON object for the subject (called the subject object)
    2. Group all triples having the current subject by predicate
    3. For each predicate:
        1. Create a JSON array (called the value array)
        2. Select all triples having the current subject and current predicate
        3. For each value:
            1. Create a JSON object (called the value object)
            2. Add a key/value pair to the value object with the key being the string "value" and the value being the lexical value of the triple va
            3. Add a key/value pair to the value object with the key being the string "type" and the value being one of "literal", "uri" or "bnode" value
            4. If the triple's value is a plain literal and has a language then add a key/value pair to the value object with the key being the string language token
            5. If the triple's value is a typed literal then add a key/value pair to the value object with the key being the string "datatype" and valu
            6. Push the value object onto the end of the value array
        4. Add a key/value pair to the subject object with the key being the predicate URI and the value being the value array
    4. Add a key/value pair to the root object with the key being the URI or blank node identifier of the subject and the value being the subject object

Further Examples

RDF/XML can be converted into the specified RDF/JSON format by using the http://convert.test.talis.com web service.

Publishing RDF/JSON on the web

If doing content-negotiation, respond to, and send the content-type as `application/json`. An empty graph (ie: no triples) should be served as an empty object:

References

1. Tags for the Identification of Languages
2. RDF JSON Brainstorming
3. http://json.org/
4. Uniform Resource Identifier (URI): Generic Syntax