

RDF/JSON: A specification for serialising RDF in JSON

Keith.Alexander@Talis.com
(kwijibo in #swig,#talis irc.freenode.net)

What is JSON?

(JavaScript Object Notation)

```
{ "foo" : [ 0, 1, "hello world",  
           false, true ] }
```

JSON

- Lightweight data-interchange format
- an alternative to XML
- **a plain-text serialisation of universal data structures**

Why JSON is good

- relatively easy to parse
- translates transparently into native data structures (rather than a custom object model)
- so you can serialise data on the server, parse it on the client, and you have the same data structure.

Why represent RDF in JSON?

to publish it

- RDF is central to the Talis platform
- You can get JSON from SPARQL if you do a SELECT or an ASK
- but not a DESCRIBE or a CONSTRUCT (or by dereferencing URIs)
- DESCRIBE and CONSTRUCT are massively useful

to consume it

- no need for a special parser - just use any JSON parser.
- when you parse it, you have a usable data structure.

Why another RDF in JSON?

- there is prior art:
 - Simile Exhibit's data format (a subset of RDF)
 - JDIL (shown later)
 - ARC (v. 1)'s JSON output from DESCRIBE (not bad, but not great, and was being evolved with ARC 2 anyway)
 - various 'flat triple'-style structures
- these RDF JSON serialisations have their good points
- but accessing the data can take effort: lots of **fors** and **ifs**
- no particular community uptake

prior art: flat triples

```
{
  "data" : [
    {
      "s" : { "type" : "uri" , "uri" : "http://example.org/about" } ,
      "p" : "http://purl.org/dc/elements/1.1/creator",
      "o" : { "type" : "literal", "val" : "Anna Wilder" }
    } ,
    {
      "s" : { "type" : "uri" , "uri" : "http://example.org/about" } ,
      "p" : "http://purl.org/dc/elements/1.1/title",
      "o" : { "type" : "literal", "val" : "Anna's Homepage", "lang" :
"en" }
    }
  ]
}
```

flat triples

- nice simple, and well-defined structure
- predictable
- but clunky to use for common resource-oriented tasks, like rendering all the properties of a resource, or accessing a particular property of a resource

prior art: resource-oriented

```
{
  "@namespaces": {
    "dc": "http://purl.org/dc/elements/1.1/",
    "rss": "http://purl.org/rss/1.0/",
    "georss": "http://www.georss.org/georss/"
  },
  "@type": "rss:channel",
  "rss:items": [
    { "@type": "rss:item",
      "rss:title": "A visit to Astoria",
      "rss:description": "sample description",
      "dc:coverage": {
```

JDIL

JDIL

- **Namespaces:** good for reading and hand-editing data, but makes using it more complicated if you don't know in advance what prefixes have been used.
- **Nesting:** looks good, but makes it hard to find resources.
- **Not indexed by URI:** means you have to iterate over the whole graph.

What we wanted

- Express the whole RDF model - no information loss
- Easy to program with - requiring as few control structures as possible.
- Rigid - only one way to do it!

Talis' RDF/JSON

- { "S" : { "P" : [O] } }
- (bnode subjects use turtle-style notation: **_:a1**)
- The object of the triple O is represented as a 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*)

http://n2.talis.com/wiki/RDF_JSON_Specification

example:

Turtle

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

RDF/JSON

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

usage: accessing a resource

```
var resource = data['http://example.org/about'];  
// resource is an object containing  
// all the properties belonging to http://example.org/about
```

usage: accessing the title of a resource

```
var title = data[ 'http://example.org/about' ]  
  [ 'http://purl.org/dc/elements/1.1/title' ]  
  [ 0 ] [ 'value' ] ;
```

usage: iterating over a graph

```
for(var uri in data){
  for(var property in data[uri]){
    for(var i=0; i<data[uri][property].length; i++ ){
      var s = uri;
      var p = property;
      var o = data[uri][property][i]['value'];
      var o_type = data[uri][property][i]['type'];
      var o_lang = data[uri][property][i]['lang'];
      var o_datatype = data[uri][property][i]['datatype'];
    }
  }
}
```

Advantages of Community Adoption

- Data Interoperability (easy to combine data)
- Code Interoperability (pass data between components without reserialising and parsing)
- Developer familiarity (you get used to the code patterns to use with the data)

Implementations

- ARC 2 (<http://semsol.org>)
- Drupal RDF-API
- Raptor
- forthcoming Jena implementation (Talis)
- rdfapi-js

web services

- <http://triplr.org>
- <http://convert.test.talis.com> (beta, as you can guess from the url)

the future

- more implementations!
- make it available through Talis' own API everywhere we offer RDF / XML
- add named graph support (maybe)
- submission to the W3C