A Quick Tutorial on the Turtle RDF Serialization

Filed under: Semantic Web, Web Architectures by Edward Benson on Thursday, 6 November 2008 | 3 Comments
Tags: RDF, Semantic Web, turtle

This post was going to be an email to a project mate in my database class, but I found myself wanting to markup the text in HTML so I figured I'd put it here.

Unlike web specifications like HTML, which have only one representation on paper/disk, RDF is just an abstract model — it doesn't matter how you represent it as long as you stay true to its abstract properties. So one of the first things people ask when it comes to RDF is, "Ok, so how to I write it?"

If you Google the web for this answer, you will probably come up with RDF/XML as your answer, which is the "official" representation. Unfortunately it is also a pretty painful and hard-to-read one, too. It would be as if the United States made its official food the brussel sprout.

So this is a quick tutorial in Turtle, one of the more humane RDF serializations, IMHO, because it is simple to understand, quite readible/editable in raw form by a human, and relatively compact as far as RDF goes. A good comparison of the non-XML serialization formats for RDF can be found here — scroll down to the bottom to see the big Venn diagram.

Warning: This description of how things should be notated only applies to Turtle, not RDF in general

URIs, Literals, and Triples

There are two datatypes in RDF. This may or may not be the official jargon:

URIs reference some conceptual entity. They are usualy written like you see web addresses but surrounded in < > brackets, as in `<http://people.csail.mit.edu/eob#ted>.` Note that the < > brackets do not represent an XML tag – you don't need to backslash it as a singleton, liks so: <br />. The brackets are just to dilineate the URI.

Literals are literal datatypes — strings, ints, etc. Literals should enclosed in quotes but may be typed with a ^^ operator. Literals are usually typed by XSD datatypes but don't have to be.

Two untyped literals may be represented as

- "123″
- "Ted"

and two typed literals may be represented as

- "123″^^xsd:integer
- "Ted"^^xsd:string

Triples are the atomic block of expression in RDF. They are always represented by:

1. A subject, which *must* be a URI (called a Resource in this sense)
2. A property, which *must* be a URI (called a Property in this sense)
3. An object, which can be either a URI (called a Resource in this sense) or a literal

So two triples (or *statements*) might be:

- <http://example.org/ted> <http://example.org/name> "Ted"
- <http://example.org/ted> <http://example.org/knows> <http://example.org/Tim>

Namespaces

Writing full URIs all the time takes up a lot of space, so Turtle lets you declare namespaces to prefix them. Prefixes should all come at the beginning of your Turtle file.

The prefix:

```
@prefix haystack: <http://haystack.csail.mit.edu/>
```

Allows you to write the phrase

```
haystack:ted
```

and it will be interpreted as

```
<http://haystack.csail.mit.edu/ted>
```

You can also create a *default prefix*:

```
@base  <http://db.csail.mit.edu/6.830/project#>
```

Allows you to write the phrase

```
:name
```

and it will be interpreted as

```
<http://db.csail.mit.edu/6.830/project#name>
```

Typed Resources

The RDF spec defines a particularly important property, **rdf:type** (note the use of a prefix) that is used to type a particular resource. A *shortcut* for **rdf:type** in Turtle is simply **a**.

So you can write

```
haystack:ted a someprefix:student
```

And it will be interpreted as

```
haystack:ted rdf:type someprefix:student
```

A Simple Statement with a Full Stop

Statements are written SUBJECT PREDICATE OBJECT and end in a period. The period indicates a full stop. For example:

```
haystack:ted a someprefix:student .
```

Multiple Statements about one Subject with a Soft Stop

A semicolon at the end of a statement means that the subject continues as an *implied subject* and more properties and values will continue. For example:

```
haystack:ted a                  someprefix:student ;
             person:firstName   "Ted"^^xsd:string ;
             person:lastName    "Benson"^^xsd:string ;
             person:knows       haystack:mihir .
```

Finish this block of statements with a full stop.

Blank Nodes

Blank nodes are evil. At least when recording data; they're a great shortcut when writing a query. But if you must use them, they are connoted with square brackets **[]**.

You can use the blank node as the ultimate subject itself:

There exists a student with first name "Ted"

```
[]           a                  someprefix:student ;
             person:firstName   "Ted"^^xsd:string .
```

or you can use a blank node as the object of an RDF statement:

haystack:ted knows haystack:adam and some student with first name "Mihir"

```
haystack:ted a                   someprefix:student ;
             person:knows        [
                                   a                 someprefix:student ;
                                   person:firstName "Mihir"^^xsd:string .
                                 ] ;
             person:knows        haystack:adam .
```

Serializing Data to Turtle

If you are writing data out to a file, you don't really need to worry about all the shortcuts in notation above. You can just write SUBJECT PREDICATE OBJECT statements followed by full stops (periods).

```
@prefix haystack: <http://haystack.csail.mit.edu/>
@prefix person: <http://example.org/person#>

haystack:ted   a                someprefix:student .
haystack:mihir a                someprefix:student .
haystack:ted   person:knows     haystack:mihir .
haystack:ted   person:firstname "Ted"^^xsd:string .
haystack:ted   person:someprop  <http://example.org/some/unprefixed/uri> .
```

And that's it! You can always run the Turtle file through a formatter that will generate the "pretty" version of it for you, so it isn't so important to worry about writing a serialization routine that generates pretty Turtle unless you are really concerned about space.