

Simple Linked Data Deployment Tutorial — using RDF-Turtle Notation



[Kingsley Uyi Idehen](#)

.

Published in

[OpenLink Software Blog](#)

.

7 min read

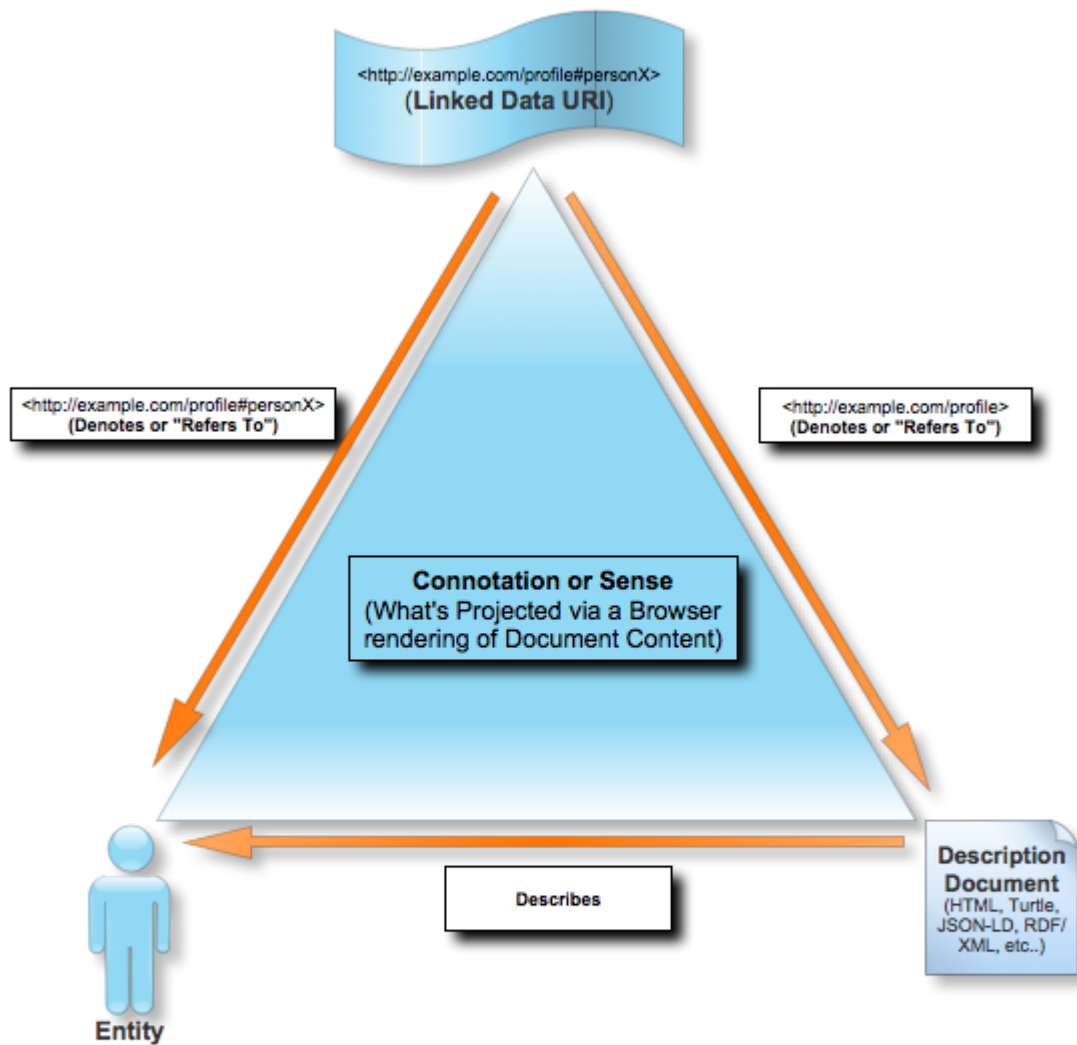
.

Jan 5, 2018

Here is a simple tutorial that demonstrates the fact that Linked Data deployment is easier than is generally assumed.

Why?

I want observations that I document to be "webby", or "web-like", in nature; i.e., I want entity relationships that can be navigated by clicking on hyperlinks (HTTP URIs) using a follow-your-nose pattern.

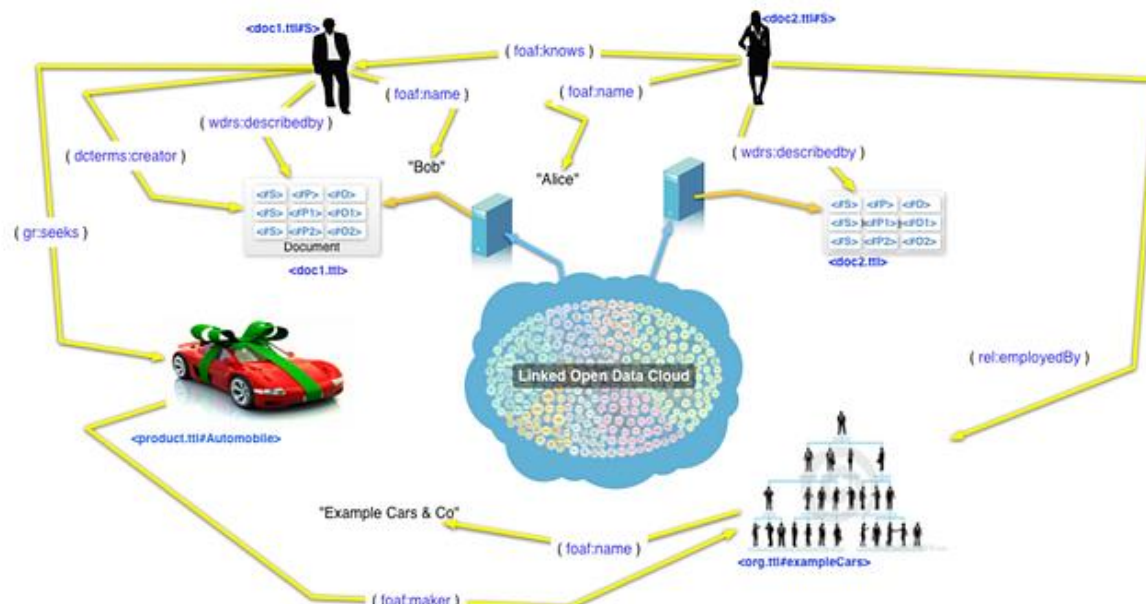


Linked Data Semiotic Triangle

What?

Describe how several entities are related, leveraging the data connectivity and access prowess of Linked Data and the descriptive power of RDF sentences:

1. a lady named "Alice"
2. a man named "Bob"
3. a car manufacturer named "Example Cars & Co"
4. an unbranded red automobile (car)



How?

1. Identify (name) entities using HTTP URIs so that anyone can look up their description documents — i.e., leverage the fact that HTTP URIs resolve to documents.
2. Use RDF Language *subject*→*predicate*→*object* structured sentences/statements (using a variety of notations) to store entity descriptions in a document that's also identified by an HTTP URI (more specifically in this case, a URL, since we are dealing with a document on an HTTP network).
3. Repeat the process whenever I encounter something of interest — i.e., take notes about what I observe using simple RDF Language sentences.

1. Create a folder on my local drive.
2. Create a document for each entity that I plan to describe — `doc1.ttl` for “Bob” [1], `doc2.ttl` for “Alice”[2], `org.ttl` for “Example Cars & Co” [3], `product.ttl` for the automobile [4].
3. Add content in the form of RDF statements (triples) that represent how these entities are related to each document.
4. Copy all of my files to an HTTP accessible folder that’s published to a public (e.g., World Wide Web) or private (e.g., Intranet) network.
5. Done!

doc1.ttl — about **Bob** (friend of Alice who seeks a Car)


```
{
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
```

```

@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix wdrs: <http://www.w3.org/2007/05/powder-s#> .<>
a foaf:Document ;
foaf:name "A Document that describes a Person literally identified as 'Bob'" ;
dcterm:description "A collection of RDF sentences that describe a Person
literally identified (or labeled) as 'Bob'" ;
foaf:primaryTopic <#bob> .<#bob>
a foaf:Person ;
foaf:name "Bob" ;
foaf:depiction <http://www.silhouettegraphics.net/wp-
content/uploads/2013/10/young-man-silhouettegraphics-.jpg> ;
gr:seeks <product.ttl#CoolRedCar> ;
foaf:knows <doc2.ttl#alice> ;
wdrs:describedby <> .
}

```

The screenshot shows the OpenLink Structured Data Sniffer interface. The browser address bar displays 'https://medi...'. The application is in 'Turtle' view. It presents two statement collections:

- Statement Collection #1:**
 - Entity:** <https://medium.com/openlink-software-blog/simple-linked-data-deployment-tutorial-a532e568c82f>
 - Attributes:**
 - rdf:type:** [foaf:Document](#)
 - foaf:name:** A Document that describes a Person literally identified as 'Bob'
 - dcterm:description:** A collection of RDF sentences that describe a Person literally identified (or labeled) as 'Bob'
 - foaf:primaryTopic:** [bob](#)
- Statement Collection #2:**
 - Entity:** [bob](#)
 - Attributes:**
 - rdf:type:** [foaf:Person](#)
 - foaf:name:** Bob
 - foaf:depiction:** 
 - gr:seeks:** <https://medium.com/openlink-software-blog/product.ttl#CoolRedCar>
 - foaf:knows:** <https://medium.com/openlink-software-blog/doc2.ttl#alice>

At the bottom, it shows 'ver: 2.15.9 OpenLink Structured Data Sniffer' and 'Copyright © 2015-2017 OpenLink Software'.

OpenLink Structured Data Sniffer rendering of RDF Sentences that describe 'Bob'

[doc2.ttl](#) — about Alice (friend of Bob who is employed by a Car Manufacturer)

```

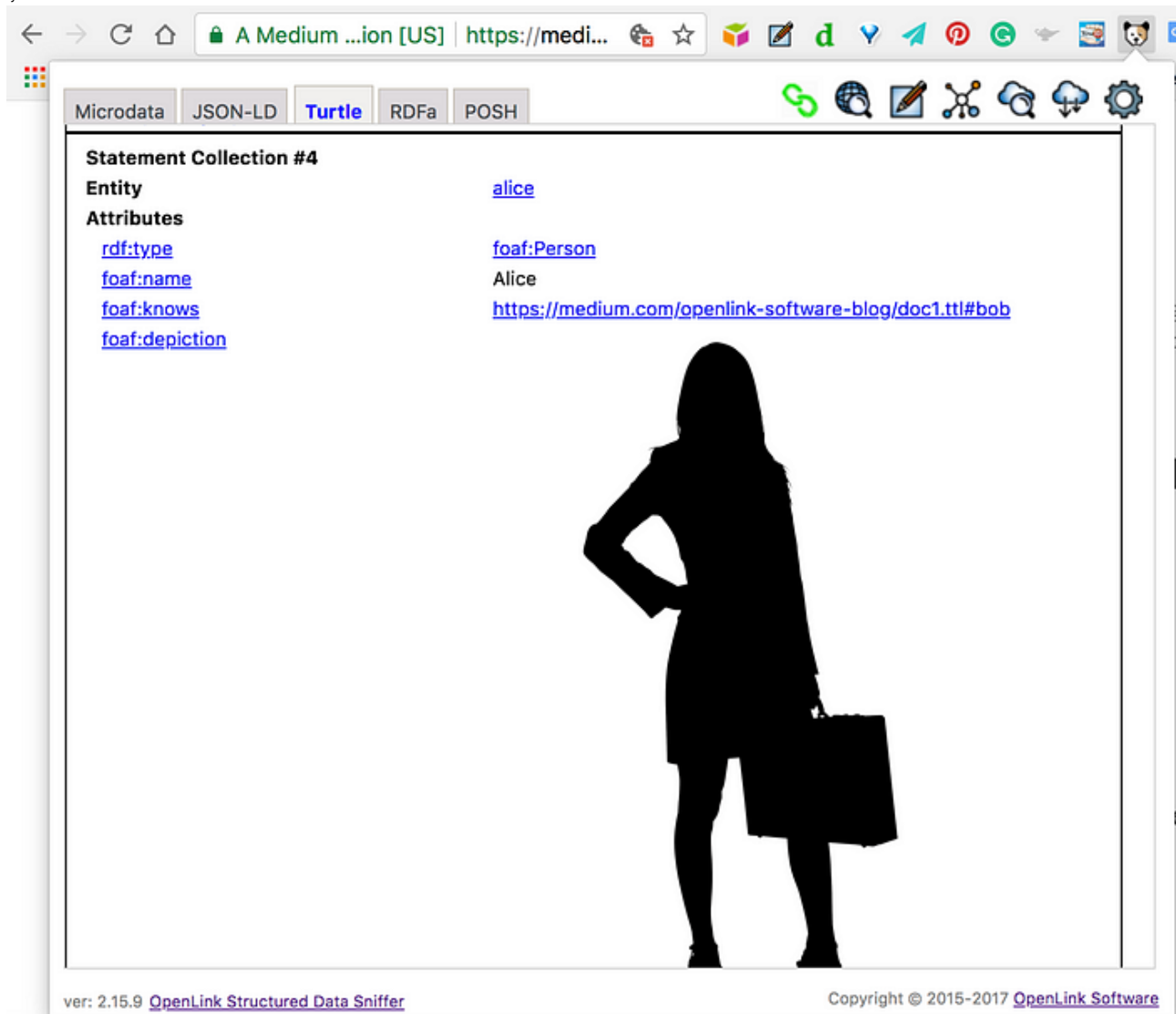
{
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix wdrs: <http://www.w3.org/2007/05/powder-s#> .
@prefix rel: <http://purl.org/vocab/relationship/> .
@prefix dcterm: <http://purl.org/dc/terms/> .
@prefix gr: <http://purl.org/goodrelations/v1#> .<>
a foaf:Document ;
foaf:name "A Document that describes a Person literally identified as 'Alice'" ;

```

```

dcterms:description "A collection of RDF sentences that describe a Person
literally identified (or labeled) as 'Alice'" ;
foaf:primaryTopic <#alice> .<#alice>
rdf:type foaf:Person ;
foaf:name "Alice" ;
foaf:knows <doc1.ttl#bob> ;
foaf:depiction <http://clipart-library.com/images/kiMb8ggKT.jpg> ;
rel:employedBy <org.ttl#exampleCars> ;
wdrs:describedby <> .
}

```



OpenLink Structured Data Sniffer rendering of RDF Sentences that describe ‘Alice’

[org.ttl](#) — **about Car Maker** (who manufactures Cars and employs Alice)


```

{
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix wdrs: <http://www.w3.org/2007/05/powder-s#> .<>
a foaf:Document ;
foaf:name "A Document that describes an Organization literally identified as

```

```
'Example Cars & Co.'" ;
dcterms:description "A collection of RDF sentences that describe an Organization
literally identified (or labeled) as 'Example Cars & Co.'" ;
foaf:primaryTopic <#exampleCars> .<#exampleCars>
rdf:type gr:BusinessEntity , foaf:Organization ;
foaf:name "Example Cars & Co." ;
foaf:depiction
<https://thumb1.shutterstock.com/display_pic_with_logo/3648824/532862638/stock-
vector-silhouette-automated-production-line-robotic-factory-banner-
532862638.jpg> ;
foaf:made <product.ttl#CoolRedCar> ;
wdrs:describedby <> .}
```

The screenshot shows the OpenLink Structured Data Sniffer interface. At the top, there's a browser-like address bar with the URL 'https://medi...'. Below it, a toolbar contains various icons for different data formats: Microdata, JSON-LD, Turtle (selected), RDFa, and POSH. The main content area is titled 'Statement Collection #6' and displays the following information:

- Entity:** [exampleCars](#)
- Attributes:**
 - [rdf:type](#) [gr:BusinessEntity](#)
 - [rdf:type](#) [foaf:Organization](#)
 - [foaf:name](#) Example Cars & Co.
 - [foaf:depiction](#) 
- [foaf:made](#) <https://medium.com/openlink-software-blog/product.ttl#CoolRedCar>
- [wdrs:describedby](#) <https://medium.com/openlink-software-blog/simple-linked-data-deployment-tutorial-a532e568c82f>

At the bottom, the version 'ver: 2.15.9' and copyright 'Copyright © 2015-2017 OpenLink Software' are displayed.

OpenLink Structured Data Sniffer rendering of RDF Sentences that describe 'Examples Car & Co.' the Car Manufacturer

[product.ttl](#) — **about Car** (Product made by Car Maker that employs Alice)

```
{
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix wdrs: <http://www.w3.org/2007/05/powder-s#> .<>
a foaf:Document ;
foaf:name "A Document that describes a Product literally identified as 'Gift
Wrapped Cool Red Car -- Brand Unknown'" ;
dcterms:description "A collection of RDF sentences that describe a Product
literally identified (or labeled) as 'Gift Wrapped Cool Red Car -- Brand
```




```

Unknown'' ;
foaf:primaryTopic <#CoolRedCar> .<#CoolRedCar>
rdf:type gr:ProductOrService ;
foaf:name "Gift Wrapped Cool Red Car -- Brand Unknown" ;
foaf:depiction <http://highroaddigital.com/wp-
content/uploads/2015/01/wonanewcar.jpg> ;
foaf:maker <org.ttl#exampleCars> ;
wdrs:describedby <> .
}

```

The screenshot displays the OpenLink Structured Data Sniffer interface. At the top, the browser address bar shows 'https://medi...'. Below the browser window, the application has tabs for 'Microdata', 'JSON-LD', 'Turtle' (selected), 'RDFa', and 'POSH'. The main content area is divided into two sections: 'Statement Collection #8' and 'Statement Collection #9'.

Statement Collection #8:

- Entity:** [CoolRedCar](#)
- Attributes:**
 - [rdf:type](#): [gr:ProductOrService](#)
 - [foaf:name](#): Gift Wrapped Cool Red Car -- Brand Unknown
 - [foaf:depiction](#): 
 - [foaf:maker](#): <https://medium.com/openlink-software-blog/org.ttl#exampleCars>
 - [wdrs:describedby](#): <https://medium.com/openlink-software-blog/simple-linked-data-deployment-tutorial-a532e568c82f>

Statement Collection #9:

- Entity:** <https://medium.com/openlink-software-blog/simple-linked-data-deployment-tutorial-a532e568c82f>
- Attributes:**
 - [rdf:type](#): [foaf:Document](#)
 - [foaf:name](#): A Document about illustrating a simple Linked Data Deployment exercise
 - [dcterms:description](#): A collection of RDF sentences that describe a Linked Data Deployment exercise

At the bottom, the version 'ver: 2.15.9' and 'OpenLink Structured Data Sniffer' are shown on the left, and 'Copyright © 2015-2017 OpenLink Software' is shown on the right.

OpenLink Structured Data Sniffer rendering of RDF Sentences that describe ‘Gift Wrapped — Cool Red Car’

[illustration.ttl](#) — an additional document functioning as an index of all the entity description documents

```

{
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ns4: <http://kingsley.idehen.net/dataspace/person/kidehen#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix wdrs: <http://www.w3.org/2007/05/powder-s#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix dcterms: <http://purl.org/dc/terms/> .<>
a foaf:Document ;
foaf:name "A Document about illustrating a simple Linked Data Deployment
exercise" ;
dcterms:description "A collection of RDF sentences that describe a Linked Data
Deployment exercise" ;

```

foaf:depiction

```
<http://kingsley.idehen.net/DAV/home/kidehen/Public/Linked%20Data%20Documents/Tutorials/slide38_linked_data_network_v2_SPO%20_latest_relative_uris.png> ;  
foaf:topic <GlossaryOfTerms.ttl>, <doc1.ttl>, <doc2.ttl>, <product.ttl>,  
<org.ttl> .  
}
```

Microdata JSON-LD **Turtle** RDFa POSH

[foaf:maker](#)
[wdrs:describedby](#)

<https://medium.com/openlink-software-blog/org.ttl#exampleCars>
<https://medium.com/openlink-software-blog/simple-linked-data-deployment-tutorial-a532e568c82f>

Statement Collection #9

Entity <https://medium.com/openlink-software-blog/simple-linked-data-deployment-tutorial-a532e568c82f>

Attributes

[rdf:type](#) [foaf:Document](#)
A Document about illustrating a simple Linked Data Deployment exercise

[foaf:name](#) A collection of RDF sentences that describe a Linked Data Deployment exercise

[dcterms:description](#) A collection of RDF sentences that describe a Linked Data Deployment exercise

[foaf:depiction](#)

[foaf:topic](#) <https://medium.com/openlink-software-blog/GlossaryOfTerms.ttl>
[foaf:topic](#) <https://medium.com/openlink-software-blog/doc1.ttl>
[foaf:topic](#) <https://medium.com/openlink-software-blog/doc2.ttl>
[foaf:topic](#) <https://medium.com/openlink-software-blog/product.ttl>
[foaf:topic](#) <https://medium.com/openlink-software-blog/org.ttl>

ver: 2.15.9 [OpenLink Structured Data Sniffer](#) Copyright © 2015-2017 [OpenLink Software](#)

OpenLink Structured Data Sniffer rendering of RDF Sentences that describe 'Illustration of Entity Relationships and the RDF documents from which they originate'

Additional Notes

RDF-Turtle and/or Nanotation

I am using Turtle (due to its human readability) as the notation for inscribing RDF statements to my documents.

By using [nanotation](#), I am able to use this document (this Medium post) as a 5-Star Linked Data source, as you will see if you install our [Structured Data Sniffer extension](#) and/or via the depiction below.

Using Relative URIs is the key to simple Linked Data Deployment. The HTTP Fragment Identifier is a powerful feature that enables HTTP URIs (Hyperlinks) to identify anything. More importantly, it enables their use when identifying the subject, predicate, and object of an RDF sentence; i.e., this is what makes the notion of a Semantic Web viable.

Due to what's outlined above, I am able to deploy this Linked Data Deployment tutorial using the age-old File Create, Save, Copy, and Share pattern across the proprietary storage services provided by Dropbox, Google Drive, Microsoft OneDrive, Amazon S3, etc, starting from my local file system.

Conclusion

By following the steps outlined above, you would have successfully described how four entities (two people, a company, and a product) are related. Even better, you would have achieved the goal of making your effort native to the Web; i.e., your document viewers (human or bots) can simply follow links presented in your documents to better understand the information encoded in the entity relationships you've represented in RDF sentences.

At this juncture, I strongly encourage you to watch this video excerpt from a recent documentary about

[Tim Berners-Lee](#)

that sheds light on the inspiration and insights that lead to the creation of the platform known as the World Wide Web.

Using Turtle

Turtle is a language. It is designed for expressing data, particularly linked data on the web. Just as HTML is used for linked text, Turtle is used for linked data. Turtle expresses stuff about things. Abstract things, real things, people, documents, So where HTML uses the hash sign like in `foo.html#intro` to refer to an anchor, which is a part of the document, Turtle uses the same hash sign to refer to something defined by the document, like `students.ttl#bob` or `profile.ttl#me`.

Turtle is simple. The data model it uses, called RDF, is simple. Turtle was originally designed for writing in chat messages, and on whiteboards, and making quick configuration files or test files, so it tries to be concise.

```
<profile.ttl#me>    <http://xmlns.com/foaf/0.1/knows>    <students.ttl#bob>    .
```

That is the simplest thing you can say in Turtle: one fact, one subject-verb-object triple. Note it ends with a dot.

- The data is sets of *triples*, or sentences with just a subject, a predicate, and an object. Like "I know bob".
- URIs like `students.ttl#bob` are used for the subjects, predicates and the objects.

There is a shorthand `a` which you can use as a verb to give the class of a thing, to say what sort of thing it is. Classes, like properties, also have URIs.

```
<profile.ttl#me>    a    <http://xmlns.com/foaf/0.1/Person>    .
```

Prefixes

Turtle uses URIs to identify people all the time, so it has a way of abbreviating them. You declare a prefix say at the top of the file, and then use `foo:` prefix with a colon:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
  
<profile.ttl#me>    a    foaf:Person    .
```

Tip: There are a ste of common prefixes for vocabularies we use a lot in `@@@`.

When you use URIs in angle brackets, then they relative URIs, relative to the URI of the current document. So if the document is, say, `https://alice.example.com/public/profile.ttl` then

```
<#me>                means <https://alice.example.com/public/profile.ttl#me>  
<#myHome>            means <https://alice.example.com/public/profile.ttl#myHome>  
<students.ttl#bill> means <https://alice.example.com/public/students.ttl#bill>
```

Things like `<#me>` and `<myhome>` are local identifiers within the file you are creating. They are like `const` in a JS file in a way, local identifiers. The business of the data document you are writing is done in terms of those local identifiers. Becasue local identifiers are used quite a lot, it is useful to declare a prefix, the empty string prefix `:`, so that they can be just written with a leading colon:

```
@prefix : <#>.    # The empty prefix means this document  
  
:alice a :Person; :age 77 . # All local IDs
```

In what follows we will often assume a file starts with

```
@prefix : <#>.
```

Property trees

As a shorthand, when you have more than one fact about the same subject, you can just use a semicolon ";" to add more :

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<profile.ttl#me> a foaf:Person ;
    foaf:name "Alice";
    foaf:knows <students.ttl#bob> .
```

You can also, when have multiple objects with the same subject *and* predicate, just list them with commas between them, so

```
<profile.ttl#me> a foaf:Person ;
    foaf:knows <students.ttl#bob>,
                <students.ttl#charlie>,
                <students.ttl#david>;
    foaf:name "Alice".
```

means the same as:

```
# Means the same as above
<profile.ttl#me> a foaf:Person.
<profile.ttl#me> foaf:knows <students.ttl#bob>.
<profile.ttl#me> foaf:knows <students.ttl#charlie> .
<profile.ttl#me> foaf:knows <students.ttl#david>.
<profile.ttl#me> foaf:name "Alice".
```

Periods are stronger than semicolons, which are stronger than commas, like in English.

Turtle is a free form language: you can add or remove spaces and new lines anywhere. Comments start with a hash sign.

Data Values

Above we have shown the triples each expressing the relationship between things: people and classes. You can also put data values in the object position, like the string "Alice" above. In Turtle (and the RDF data model underneath) values are typed.

"Alice" means the string Alice

"20" means a string "20", not the number 20

20 means the integer (whole number) 20.

20.0 means a decimal number

2.0e1 means a (double precision) floating point number 2×10^1

These data types cover a lot of values. You can also explicitly write something with and explicit data type from the XML data types @@link standard, XSD. To do that you write the data value as a string, then two carets ^^ and then the XSD datatype itself.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
<profile.ttl#me> a foaf:Person ;
    foaf:name "Alice";
```

```
foaf:age 21;
foaf:height 1.76e0;
foaf:bithDate @@@ "1990-03-31"^^xsd:date .
```

Its also worth mentioning that URIs are yor freind if you wat to given email addresses or telephone numbers, as there are RI schemes for those.

<tel:+1781-555-1212> How to do a phone number as a URI
 <mailto:alice@example.com> How to put in an email address.

Why give an explicit `tel:` URI or an explicitly typed data field instead of just a string? Because it means that the systems which re-use the data will be able to be smarter, to automatically pick the appropriate way of displaying it or converting into an other format, and so on. The tradition in linked data is stronger data typing, like Python or C rather than JS.

Structure

You can add structure to your data in Turtle using the square brackets [...], which can be read as "Something which has ...". So instead of a flat set of properties like

```
:Alice :homeAddressStreet "Accaia Ave";
      :homeAddressNumber 123;
      :homeAddressCity "Anytown";
      :homePhone <tel:+1-781-555-1212> .
```

its best to add structure for the home and the address.

```
:Alice :home [
      :address [
        :street "Accaia Ave";
        :number 123;
        :city "Anytown"
      ]
      :phone <tel:+1-781-555-1212>
    ] .
```

This reads in English like "Alice has a home which has an address with street "Aaccia Avenue", number 123 and city ANYtown, and has a phone number of +1785551212."

This sort of tree structure is very common: in JSON the nested things are objects; in XML they are elements. In Turtle they are called blank nodes, because they are not labelled with a URI. There is another way of writing them in Turtle, where you make up local ids for them like `_:foo` to keep track of them but **these are not URIs**. The `_:` is a special one for blank nodes.

```
:Alice :home _:a .
      _:a address _:b .
      _:b street "Accaia Ave";
          number 123;
          city "Anytown" .
      _a :phone <tel:+1-781-555-1212> .
```

The square bracket syntax is obviously clearer. You can use it in place of object or subject.

Lists

In turtle, ordered arrays of things are given with **round** brackets. They are called collections or lists, and are like LISP lists, or JS or Python arrays.

```
<#alice> foaf:children ( <#bob> <#charlie> <#dave> ) .
```

Imagine we store user input as a string, but separately generate the list of words.

```
<#userInput>  ex:line    "turn on light";
                ex:inWords ("turn" "on" "light") .
```

Conclusion

That's it. That's all there is to turtle. You now know it. It is good to get used to reading bits of turtle as though it was English. You can now use it in code snippets in chats and in code for things like forms, configuration parameters, and so on.

Extensions in rdflib

There are a few things which you should not use in data shared generally, but are shortcuts for test data and quick scripts. rdflib.js will understand these syntax but never generate them

Naked Dates

<pre>:Alice foaf:birthdate 2018-02-31 .</pre>	A raw ISO format date can be put in without explicit type
<pre>:x4 :lastModifiedTime 2018-02-31T08:24:00.0Z .</pre>	A raw ISO format date-time can be put in without explicit type

Reverse properties

A good philosophy is that when there are two relations which are inverse of each other, like `parent` and `child`, that we just use one, say `child`, in the data, like

```
:grampa fam:child :alice, :bob.
:gramma fam:child :alice, :bob.
:alice  fam:child :charlie, :davie, :ellie .
```

This makes the graph of relationships easier to query. In this example, you can in fact express all the information about Alice at once:

```
:alice  is fam:child of :grampa, :gramma;
        fam:child :charlie, :davie, :ellie .
```

The `is ... of` syntax was in the original N3 language which Turtle was derived from but unfortunately left out of the turtle standard.

Local document prefix

It is handy to define the empty string prefix `:` as being for the local document, so it can be used for local identifiers.

```
@prefix : <#>.
```

This is in fact assumed by default by rdflib.js so you can miss it out.

```
:this a :Example; :age 123.
```

is a fine test file.