Haystack 4

Exporting Haystack Definitions to RDF

Matthew Giannini

2019 Haystack Connect
Smart Data. Smart Devices. Smart Buildings. Smart Business.

WG551-RDF

- Haystack defs are not in a format that can be consumed by traditional semantic tools
- WG551-RDF subgroup
- Goals
  - Export Haystack defs as RDF statements
  - Document a set of rules to apply to generate RDF statements that add semantic meaning equivalent to the def

What is RDF?

- Resource Description Framework
- Used to express information about “resources”
- What’s a Resource? – Anything (site, equip, point…)
- Resources are identified by IRI (International Resource Identifier)

RDF Data Model

- RDF is used to assert facts about resources. These facts are called **statements**
- All statements have the same structure (**triple**)

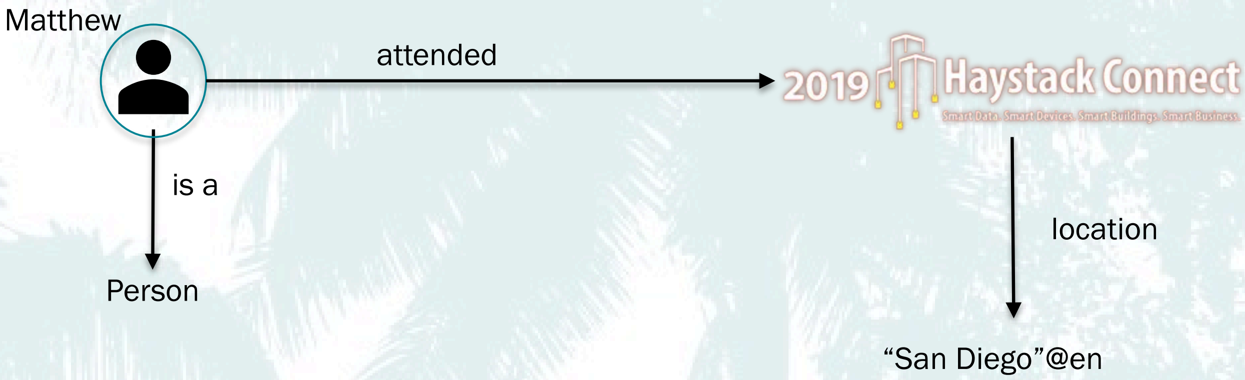
```
<subject> <predicate> <object>
```

- A statement expresses a relationship between two resources
  - The **subject** and the **object** are the two resources being related
  - The **predicate** describes how they are related (denotes a **property** of the subject)
- **subject** always a resource; **object** may be resource *or* literal.

```
<Matthew> <is a> <person>
<Matthew> <attended> <HaystackConnect>
<HaystackConnect> <location> “San Diego”@en
```

RDF Data Model

- A collection of RDF triples (i.e. statements) can be represented as a directed **Graph**



IRIs

- Resources are identified by IRI (International Resource Identifier)
- IRIs can be appear in all three positions of a triple:

  https://haystackconnect.org/people/Matthew

  http://www.w3.org/1999/02/22-rdf-syntax-ns#type

  http://xmlns.com/foaf/0.1/Person
- IRIs are frequently expressed with a prefix syntax:

  `haystack:people, rdfs:type, foaf:Person`

RDF Schema (RDFS)

- Supports the definition of **vocabularies**
- You can define the semantic meaning of your statements

RDFS - Classes

- Resources can be divided into groups called **classes**
  - `foaf:Person rdfs:type rdfs:Class`
  - `hay:Speaker rdfs:type rdfs:Class`
  - `hay:Speaker rdfs:subClassOf foaf:Person`
- Members of a class are called **instances**
  - `hay:matthew a hay:Speaker`
- An inference engine would infer that matthew is a person.

RDFS - Properties

- A **property** is a relation between a **subject** and an **object**
  - `foab:knows a rdf:Property`
  - `facebook:marriedTo a rdf:Property`
  - `facebook:marriedTo rdfs:subPropertyOf foab:knows`
- Now we can say
  - `facebook:Bob facebook:marriedTo facebook:Alice`
- An inference engine would infer that Bob knows Alice too

RDFS – domain and range

- **rdfs:domain** predicate is used to state that any resource with a given property is a member of one or more **classes**

```
foaf:knows rdfs:domain foaf:Person
```

- **rdfs:range** predicate is used to state that the values of a property are instances of one or more **classes**

```
foaf:age rdfs:range xsd:integer
```

Web Ontology Language (OWL)

- Adds more vocabulary for describing **properties** and **classes**
  - Relations between classes (disjointness)
  - Cardinality
  - Equality
  - “Richer” typing of properties
  - Characteristics of properties (e.g. symmetry)
  - Enumerated Classes

OWL

- **owl:Class** is functionally equivalent to **rdfs:Class**
  - `foaf:Person a owl:Class`
- **owl:ObjectProperty** indicates that a predicate relates two individuals
  - `foaf:knows a owl:ObjectProperty`
- **Owl:DatatypeProperty** indicates that a predicate relates and individual to a literal
  - `foaf:age a owl:DatatypeProperty`

RDF Export - Turtle

- Very popular export format for RDF Graphs
- More compact and natural expression of triples

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
...tons of prefixes...

facebook:Bob a foaf:Person ;
  foaf:age 31 ;
  facebook:marriedTo facebook:Alice .

facebook:Alice a foaf:Person ;
  facebook:marriedTo facebook:Bob ;
  library:favoriteBook library:Dune, library:NameOfTheWind .
```

From “def” to “rdf”

```
def: ^site
is: [^entity, ^geoPlace]
doc: "Site is a geographic location of the built
environment“
-------------


phIoT:site a owl:Class ;
    rdfs:subClassOf ph:entity, ph:geoPlace ;
    rdfs:label "site" ;
    rdfs:comment "Site is a geographic location of
the built environment" ;
```

General Mapping Rules - Basics

- The symbol for a def becomes the **subject** of an RDF statement
- Each tag/value pair becomes the **predicate** and **object** respectively of an RDF statement
  - Values of the **is** tag become distinct statements

General Mapping Rules - IRIs

- Every def symbol must be converted to an IRI
  - {baseUri}/{version}#{symbol}
  - https://project-haystack.org/def/phIoT/4.0#site

A Rather Useless RDF Mapping

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ph: <https://project-haystack.org/def/ph/4.0#> .
@prefix phScience: <https://project-haystack.org/def/phScience/4.0#> .
@prefix phIoT: <https://project-haystack.org/def/phIoT/4.0#> .

phIoT:site is ph:entity, ph:geoPlace ;
  ph:doc "Site is a geographic location of the built environment" ;
  ph:mandatory ph:marker .
```

Marker Tags

- Defs for marker tags are subtypes of **^marker** via the **is** tag
- Marker tag defs become instances of **owl:Class**
- The supertype tree defined by the **is** tag maps to a set of **rdfs:subClassOf** statements

supertypes	**marker**	Marker labels a dict with typing information
	entity	Top-level dicts with a unique identifier
	geoPlace	Geographic place

```
phIoT:site a owl:Class ;
  rdfs:subClassOf ph:entity, ph:geoPlace ;
```

Data Types

- Direct sub-types of **^scalar** are declared as instances of **owl:DatatypeProperty** (except markers)
  – They are declared as **rdfs:subClassOf** best xsd datatype

```
 ph:dateTime a owl:DatatypeProperty ;
    rdfs:subClassOf xsd:dateTime ;
    rdfs:comment "ISO 8601 timestamp followed by timezone identifier" ;

  ph:number a owl:DatatypeProperty ;
    rdfs:subClassOf xsd:double ;
    rdfs:comment "Integer or floating point numbers annotated with an
optional unit" ;
```

Value Tags

- Any def that is *not* a subtype of **^marker**
- **^ref** or **^choice** subtypes become instances of **owl:ObjectProperty**
  - Otherwise **owl:DatatypeProperty**
- If the def(x) has **^tagOn**, then specify the **rdfs:domain** to be all referent entities
- The **rdfs:range** of a **^ref** or **^choice** is determined by the value of the **^of** tag (if specified)
  - Otherwise, the **rdfs:range** is the appropriate data type for that tag

Example: ^ref

```
def: ^siteRef
is:  ^ref
of:  ^site
doc: "Site which contains the entity"
```

```
phIoT:siteRef a owl:ObjectProperty ;
    rdfs:range phIoT:site ;
    rdfs:label "siteRef" ;
```

Example: ^choice

```
def: ^conveys
is:  ^equipFunction
of:  ^phenomenon
doc: "Equipment conveys a substance or phenomenon."
```

supertypes	aspect	Aspects model a relationship between two definitions
	choice	Choice specifies an aspect with an single exclusive value
	equipFunction	Models one of the primary functions of an equipment type

```
phIoT:conveys a owl:ObjectProperty ;
    rdfs:range phScience:phenomenon ;
    rdfs:label "conveys" ;
    rdfs:comment "Equipment conveys a substance or
phenomenon." ;
```

Example: ^tz

```
def: ^tz
is: ^str
doc: "Timezone identifier from standard timezone database“
---
defx: ^tz
tagOn: ^point
---
defx: ^tz
tagOn: ^site
```

```
ph:tz a owl:DatatypeProperty ;
    rdfs:domain phIoT:point,
        phIoT:site ;
    rdfs:range ph:str ;
    rdfs:label "tz" ;
    rdfs:comment "Timezone identifier from standard
timezone database" ;
```

Mapping Instances

- An "instance" is a Dict (entity) with an **id** tag
- Instances are modeled with "blank" nodes labeled with the id
- Use **rdf:type** ("**a**") to indicate which ph:entity class the instance is a member of
- Tag values are encoded according to their data type
  - All marker tags are expressed using **ph:hasTag**

Example: site instance

```
id:@24192ca1-0c85f75d "Headquarters"
site
area:140797ft²
tz:New_York
dis:Headquarters
geoCoord:C(37.545826,-77.449188)
primaryFunction:Office
yearBuilt:1999
```

```
_:24192ca1-0c85f75d
  a phIoT:site ;
  ph:hasTag phIoT:site ;
  phIoT:area 140797 ;
  ph:tz "New_York" ;
  ph:dis "Headquarters" ;
  ph:geoCoord "C(37.545826,-
               77.449188)" ;
  phIoT:primaryFunction "Office" ;
  phIoT:yearBuilt 1999 .
```

Example: point instance

```
_:243e6c39-fbaf8e65 a phIoT:point ;
    ph:hasTag
        phScience:air, phIoT:cmd,
        phIoT:cur,     phIoT:discharge,
        phIoT:fan,     phIoT:his,
        phIoT:point ;
    rdfs:label "Short Pump RTU-2 Fan" ;
    phIoT:siteRef _:243e6c39-c9304b27 ;
    phIoT:equipRef _:243e6c39-b8030657 ;
    phIoT:curStatus "ok" ;
    phIoT:curVal true ;
    ph:enum "off,on" ;
    phIoT:hisMode "cov" ;
    core:kind "Bool" ;
    ph:tz "New_York" .
```

Pending Work

- How to handle units for numbers?
- How to indicate inverse relationships?
- How to indicate transitive containment?
- Are there other OWL statements we should use?