OFFICIAL

Instantiation Patterns

Based on version 4.2.0

Introduction

Formal ontologies like IES4 are normally designed with very opinionated semantics based on logical and/or philosophical commitments like the BORO 4D approach or set theory. This makes them ideal to be used as general-purpose data exchange standards. Consequently, these commitments necessitate additional considerations when mapping equivalent data to IES from representations like the JSON below.

```
{
        “name”: “Megan”,
        “accent”: “BRUMMIE”,
        “job”: “CEO”
}
```

One set of considerations is how things are instantiated. This document explores the most common instantiation patterns and articulates the rules that apply to their use.

These instantiation patterns will be articulated using the mappings of the JSON attributes introduced above, into IES. A mix of UML diagrams and RDF triples are used to articulate these patterns. The triples presented will utilise the following prefixes:

ies: – referring to things in the IES ontology

ont: – referring to things in an example, local ontology

data: – referring to things in an example, instance dataset

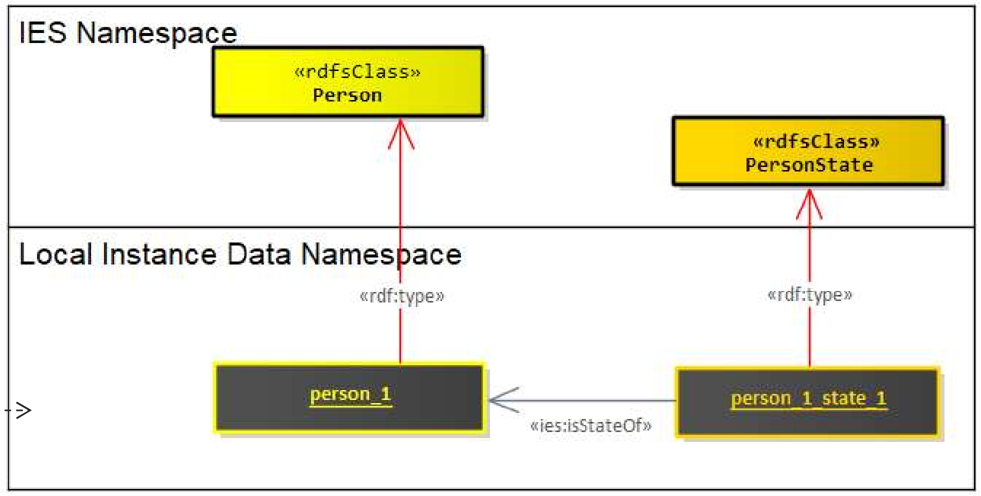
Element instances (1 of 2)

```
{
        "name": "Megan",
        "accent": "BRUMMIE",
        "job": "CEO"
}
```

This is the most common and naturally intuitive pattern of instantiating a thing with IES. Commonly used for most IES Elements. Here is an example of instantiating an ies:Person and an associated ies:PersonState (a temporal slice of a person).

Human readable identifiers associated to these elements (e.g., the name Megan) are normally found a few node-hops away from these elements on instances of ies:Name or ies:Identifier. These *Name instances* will be discussed later in this document.

```
data:person_1           a               ies:Person .
data:person_1_state_1   a               ies:PersonState .
data:person_1_state_1   ies:isStateOf   data:person_1 .
```



Element instances (2 of 2)

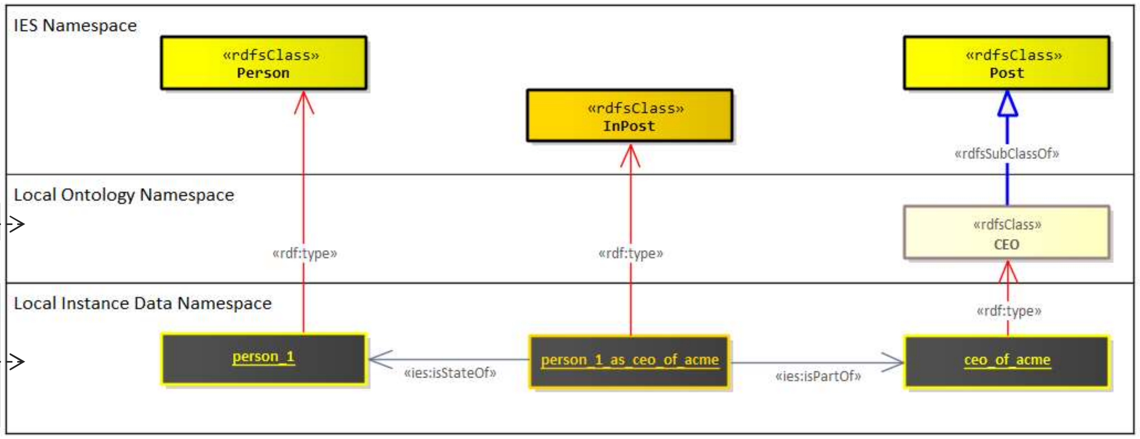
{
“name”: “Megan”,
“accent”: “BRUMMIE”,
“job”: “CEO”
}

Occasionally, creating instances of certain elements requires additional effort. Certain classes are naturally too broad to cover certain data requirements e.g., Vehicle, Device and Post. Typically, you might want to instantiate against more detailed categories, like a particular brand and model of a Vehicle or Device, or a specific job post within an organisation. Ideally, we should first build out such categories or taxonomies into our local ontology before instantiating them. There will be times when these types are data-driven or sourced from free-text fields resulting in these types needing to be created ”on-the-fly”.

For more details on creating subclasses against the IES ontology, see the “Extending IES4” guidance document.

```
ont:CEO                      rdfs:subClassOf          ies:Post .
```

```
data:person_1                  a                  ies:Person .
data:person_1_as_ceo_of_acme   a                  ies:InPost .
data:ceo_of_acme               a                  ont:CEO .

data:person_1_as_ceo_of_acme   ies:isStateOf      data:person_1 .
data:person_1_as_ceo_of_acme   ies:isPartOf       data:ceo_of_acme .
```



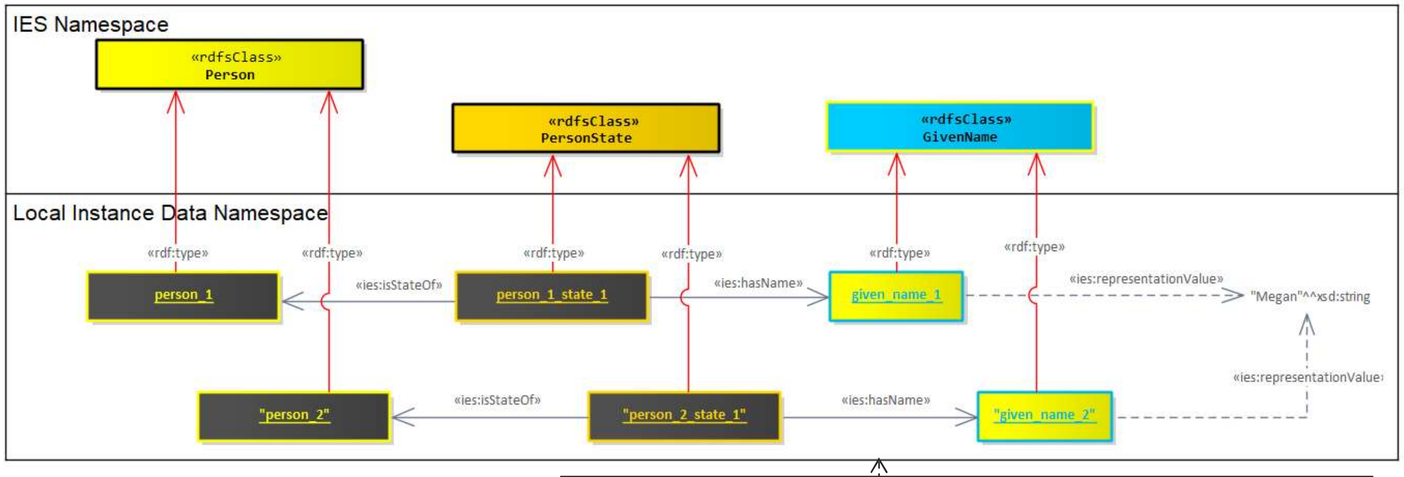
```
{
        "name": "Megan",
        "accent": "BRUMMIE",
        "job": "CEO"
}
```

Name instances

Names are a special form of representation used for identifying things. Anything can by identified by many names (or its subclass, identifiers). An important nuance of names in IES, is that when we instantiate a name, like here, the given name of Megan; that instance is not shared with other instances of people called Megan. Instead, each instance of a given name is a unique form of utterance for identifying a single thing.

The thing that is shared between two things with the same name, is the string literal at the end of the *representationValue* attribute.

This pattern for names is based on P.F. Strawson's theory of description and utterances, and Quine's Roots of Reference. Note, this pattern does not apply to the superclass of Name, *Representation*.



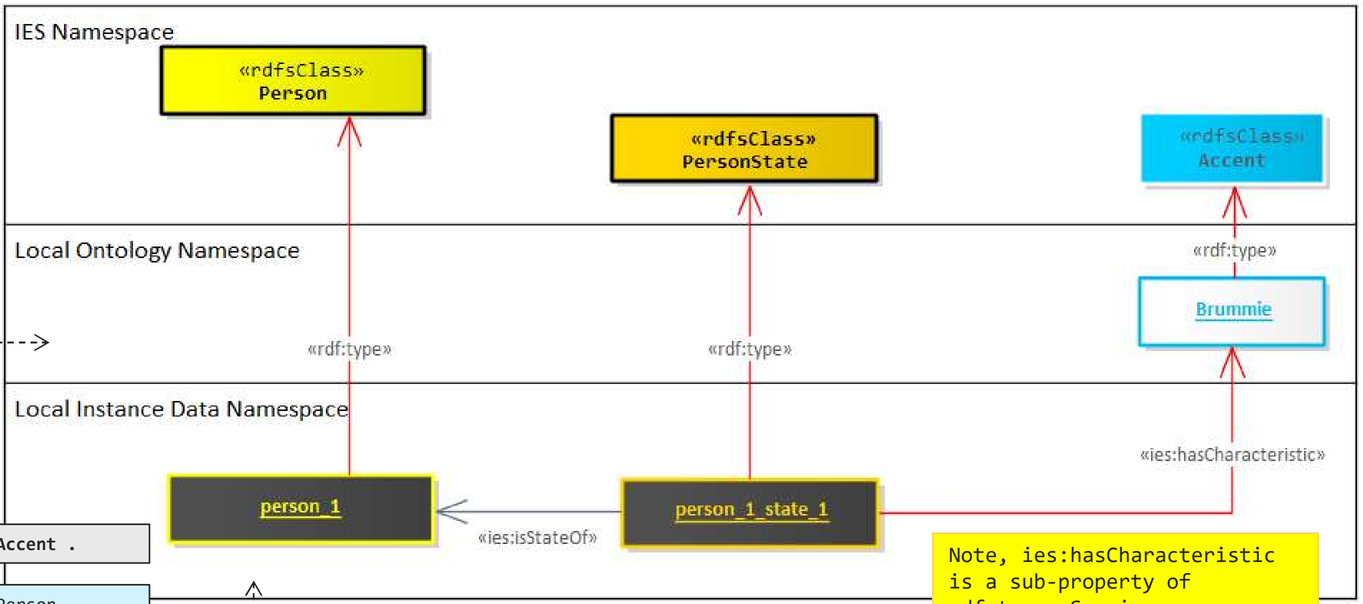
```
data:person_1           a                       ies:Person .
data:person_1_state_1   a                       ies:PersonState .
data:person_1_state_1   ies:isStateOf           data:person_1 .

data:person_2           a                       ies:Person .
data:person_2_state_1   a                       ies:PersonState .
data:person_2_state_1   ies:isStateOf           data:person_2 .


data:person_1_state_1   ies:hasName             data:given_name_1.
data:given_name_1       a                       ies:GivenName .
data:given_name_1       ies:representationValue "Megan"^^xsd:string .

data:person_2_state_1   ies:hasName             data:given_name_2.
data:given_name_2       a                       ies:GivenName .
data:given_name_2       ies:representationValue "Megan"^^xsd:string .
```

```
{
        "name": "Megan",
        "accent": "BRUMMIE",
        "job": "CEO"
}
```

Class instances

BORO ontologies such as IES allow the instantiation of classes that are themselves members of other classes. Instances of characteristics, measures and representations are such examples where this pattern is used. In this example we create a new class instance of Accent for the *Brummie* accent.

A common mistake is to assume that the human-readable value for this class instances is to be found at the end of the *representationValue* attribute. This only applies for instances of representations. All other class instances should be treated as equivalent to extensions to the IES ontology. If you do need a human-readable string for such instances, use rdfs:label.



```
ont:Brummie              a                      ies:Accent .
```

```
data:person_1            a                      ies:Person .
data:person_1_state_1    a                      ies:PersonState .
data:person_1_state_1    ies:isStateOf          data:person_1 .

data:person_1_state_1    ies:hasCharacteristic  ont:Brummie .
```

Note, ies:hasCharacteristic is a sub-property of rdf:type. So, in essence, person_1_state_1 is being allocated membership to two classes here.