SHACL

John Beverley

Assistant *Professor, University at Buffalo*
Co-Director, *National Center for Ontological Research*
Affiliate Faculty, *Institute of Artificial Intelligence and Data Science*

Outline

- Data Validation
- Motivating SHACL
- SHACL Features
- Exercises

Outline

- Data Validation
- Motivating SHACL
- SHACL Features
- Exercises

Validation Schema

- Relational databases store data according to fixed schemas of known data types

- More flexible databases - such as MondoDB - allow users to reshape data as needed

- This can be, however, both a blessing and a curse

Validation Schema

- Developers aren’t required to impose fixed structure to data from the outset, but they also don’t have much fixed structure required of data

- A *validation schema* is a set of rules defining how a data set should be structured

- Validation schemas are used to provide constraints to data for specific purposes, which can be weakened or strengthened as needed

Validation Ecosystem

- Validation schema exist for various data models; data represented in:
  - XML may be validated against XML Schema
  - JSON may be validated against JSON Schema
- Data represented in RDF often needs more robust structure for specific representation goals

XML

- eXtensible Markup Language (XML) is a markup language designed for describing data governed by minimal syntax rules

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>John</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body>Include an xml example!</body>
</note>
```

XML

- eXtensible Markup Language (XML) is a markup language designed for describing data governed by minimal syntax rules

Must begin with the XML declaration
Must have one unique root element
start-tags must have matching end-tags
Elements are case sensitive
All elements must be closed
All elements must be properly nested
All attribute values must be quoted
Entities must be used for special characters

XML Schema

- XML Schema provides a way to validate XML

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

XML Schema

- XML Schema provides a way to validate XML

Must reference the correct XSD schema
Each element is checked to ensure it exists in the schema
Element and attribute values must conform to data types
Elements must appear the correct number of times
Unique identifiers (keys) must be valid if specified in the schema
Must use the correct namespaces as defined by the schema

Outline

- Data Validation
- Motivating SHACL
- SHACL Features
- Exercises

Data Validation

- RDF artifacts circumscribe how terms *may be* used, but not how they *are* actually used

- RDF artifacts may be extended in consistent ways not envisaged during initial development

Data Validation

- RDF artifacts circumscribe how terms *may be* used, but not how they *are* actually used

Data validation supports data alignment to intended RDF models

- RDF artifacts may be extended in consistent ways not envisaged during initial development

Data validation provides a way to construct documentation of such extensions for users and developers

Validation with OWL?

- OWL allows users to apply property restrictions for *inferencing*, but restrictions aren't data constraints and inferencing isn't validation

Validation with OWL?

- OWL allows users to apply property restrictions for *inferencing*, but restrictions aren't data constraints and inferencing isn't validation

- Suppose ex:Person has **owl:maxCardinality 1** ex:Father and some instance of ex:Person **has two** ex:Father values

- An OWL processor will infer these two values are the same

Validation with OWL?

- OWL allows users to apply property restrictions for *inferencing*, but restrictions aren't data constraints and inferencing isn't validation

- Suppose ex:Person has **owl:maxCardinality 1** ex:Father and some instance of ex:Person **has two** ex:Father values

- An OWL processor will infer these two values are the same

OWL rejects the *Unique Name Assumption*

13.7.2 Unique Names Assumption

Instead of being agnostic about the equality of each term and expecting the user to axiomatize which names denote the same individual and which denote different individuals, it is often easier to have the convention that different ground terms denote different individuals.

Example 13.45. *Consider a student database example where a student must have two courses as science electives. Suppose a student has passed* $math302$ *and* $psyc303$*; then you only know whether they have passed two courses if you know* $math302 \neq psyc303$*. That is, the constants* $math302$ *and* $psyc303$ *denote different courses. Thus, you must know which course numbers denote different courses. Rather than writing* $n * (n-1)/2$ *inequality axioms for* n *individuals, it may be better to have the convention that every course number denotes a different course and thus the use of inequality axioms is avoided.*

Hesperus & Phosphorus

Venus

Rejecting Unique Names

- OWL rejects this simplification, opting instead to allow for resources to have more than one name

- It provides resources for asserting sameness:

  - **owl:sameAs** holds between two constants when they refer to the same individual
  - **owl:differentFrom** holds between two constants when they refer to different individuals

Validation with OWL?

- Suppose every person must have exactly one mother who is a person
- Suppose ex:John a ex:Person and ex:Sam a ex:Person ex:has_mother ex:Matilda

Validation with OWL?

- Suppose every person must have exactly one mother who is a person
- Suppose ex:John a ex:Person and ex:Sam a ex:Person ex:has_mother ex:Matilda
- OWL processors **will not report an error** because John doesn't have a mother, **but will add** ex:Matilda a ex:Person to the data set

Validation with OWL?

- Suppose every person must have exactly one mother who is a person
- Suppose ex:John a ex:Person and ex:Sam a ex:Person ex:has_mother ex:Matilda
- OWL processors **will not report an error** because John doesn't have a mother, **but will add** ex:Matilda a ex:Person to the data set

OWL adopts the *Open World Assumption*

5.6 Complete Knowledge Assumption

A database is often complete in the sense that anything not implied is false.

Example 5.27. *You may want the user to specify which switches are up and which circuit breakers are broken so that the system can conclude that any switch not mentioned as up is down and any circuit breaker not specified as broken is ok. Thus, down is the default value of switches, and ok is the default value for circuit breakers. It is easier for users to communicate using defaults than it is to specify the seemingly redundant information about which switches are down and which circuit breakers are ok. To reason with such defaults, an agent must assume it has complete knowledge; a switch's position is not mentioned because it is down, not because the agent does not know whether it is up or down.*

KNOWN KNOWNS

"things that we're aware that we know"

KNOWN UNKNOWNS

"things that we're aware that we **don't** know"

UNKNOWN KNOWNS

"things that we're **unaware** that we know"

UNKNOWN UNKNOWNS

"things that we're **unaware** of and **don't** know"

Open World Assumption

- OWL adopts the open-world assumption to allow for defeasible inferencing

Just because you don’t know something, doesn’t mean it’s false

Summary

- You cannot simply add data to an OWL ontology and use a reasoner to determine whether the *data conforms to the ontology*

- Because OWL processesors attempt to *conform the data to the ontology*, rather than report an error

- And even if a contradiction arises, it’s not easily traced to the problematic data that is its cause

Validation with SPARQL?

- SPARQL may be used to validate RDF data, either with direct queries against data...

- ...or through SPARQL Inferencing Notation (SPIN)

Validation with SPARQL?

- SPARQL may be used to validate RDF data, either with direct queries against data...

Resulting in complex, expressive, and often idiomatic queries

- ...or through SPARQL Inferencing Notation (SPIN)

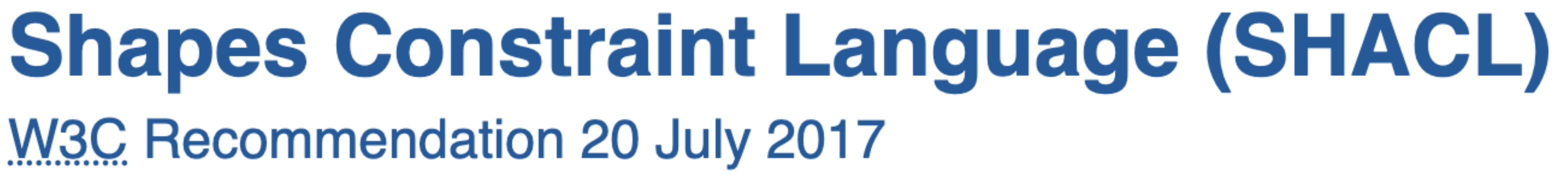
With limited expressivity since restricted to constraints on classes

Validation Ecosystem

- Validation schema exist for various data models; data represented in:
    - XML may be validated against XML Schema
    - JSON may be validated against JSON Schema

Validation Ecosystem

- Validation schema exist for various data models; data represented in:
  - XML may be validated against XML Schema
  - JSON may be validated against JSON Schema
  - RDF may be validated against the Shapes Constraint Language
- SHACL is a W3C recommended data validation standard



Shapes Constraint Language (SHACL)

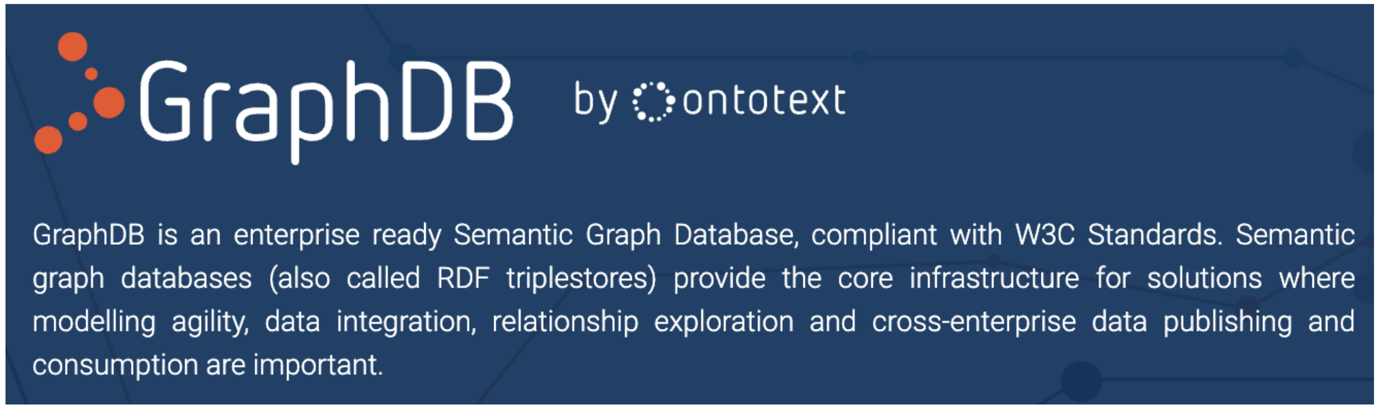
W3C Recommendation 20 July 2017

Outline

- Data Validation
- Motivating SHACL
- SHACL Features
- Exercises

Validation Workflow

- RDF graphs and SHACL graphs are imported to most often either:
  - RDF triplestores with SHACL validation support, e.g. GraphDB, Jena
  - SHACL APIs, e.g. SHACL Playground


GraphDB by ontotext

GraphDB is an enterprise ready Semantic Graph Database, compliant with W3C Standards. Semantic graph databases (also called RDF triplestores) provide the core infrastructure for solutions where modelling agility, data integration, relationship exploration and cross-enterprise data publishing and consumption are important.

Validation Workflow

- RDF graphs and SHACL graphs are imported to most often either:

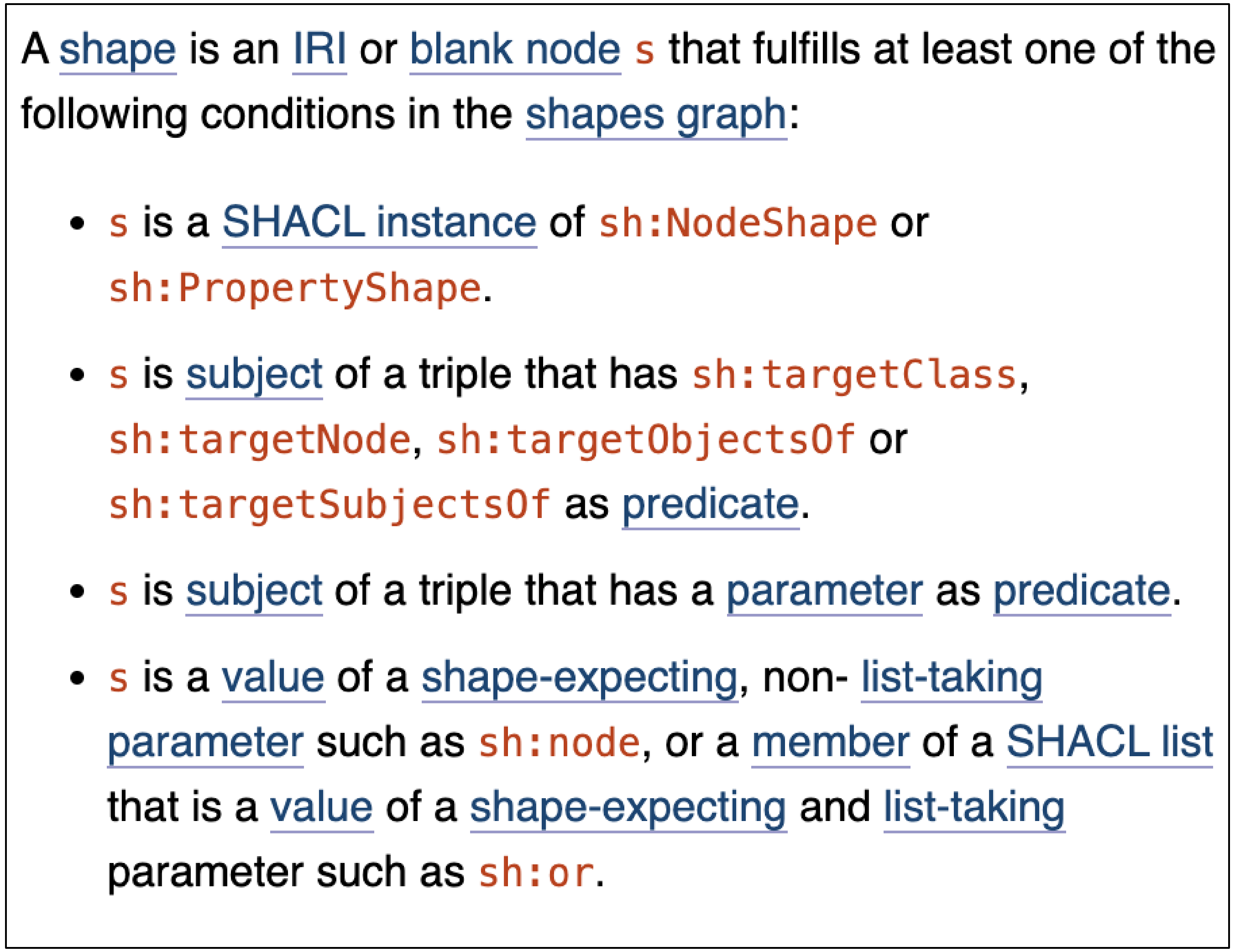
  - RDF triplestores with SHACL validation support, e.g. GraphDB, Jena
  - SHACL APIs, e.g. SHACL Playground

- SHACL graphs contain conditions which RDF graphs must satisfy to be validated

- After processing, a validation report is produced as RDF

SHACL Syntax Rules

shape



A shape is an IRI or blank node `s` that fulfills at least one of the following conditions in the shapes graph:

- `s` is a SHACL instance of `sh:NodeShape` or `sh:PropertyShape`.
- `s` is subject of a triple that has `sh:targetClass`, `sh:targetNode`, `sh:targetObjectsOf` or `sh:targetSubjectsOf` as predicate.
- `s` is subject of a triple that has a parameter as predicate.
- `s` is a value of a shape-expecting, non- list-taking parameter such as `sh:node`, or a member of a SHACL list that is a value of a shape-expecting and list-taking parameter such as `sh:or`.

SHACL Features

- *Node Shapes* declare constraints on a node
- *Property Shapes* declare constraints on values associated nodes through sh:path
- *sh:path* declares there must be a link from a given node to a specified value

SHACL Features

- *Node Shapes* declare constraints on a node
- *Property Shapes* declare constraints on values associated nodes through sh:path
- *sh:path* declares there must be a link from a given node to a specified value

```
sh:PersonShape
    a sh:NodeShape
    sh:targetClass ex:Person
    sh:property [
        sh:path rdfs:label ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ; ] ;
```

SHACL Features

- *Node Shapes* declare constraints on a node
- *Property Shapes* declare constraints on values associated nodes through sh:path
- *sh:path* declares there must be a link from a given node to a specified value

```
sh:PersonShape
    a sh:NodeShape
    sh:targetClass ex:Person
    sh:property [
        sh:path rdfs:label ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ; ] ;
```

$\longrightarrow$

This shape
    is a NodeShape

SHACL Features

- *Node Shapes* declare constraints on a node
- *Property Shapes* declare constraints on values associated nodes through sh:path
- *sh:path* declares there must be a link from a given node to a specified value

```
sh:PersonShape
    a sh:NodeShape
    sh:targetClass ex:Person
    sh:property [
        sh:path rdfs:label ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ; ] ;
```

$\longrightarrow$

This shape
    is a NodeShape
    that applies to the class "Person"

SHACL Features

- *Node Shapes* declare constraints on a node
- *Property Shapes* declare constraints on values associated nodes through sh:path
- *sh:path* declares there must be a link from a given node to a specified value

```
sh:PersonShape
    a sh:NodeShape
    sh:targetClass ex:Person
    sh:property [
        sh:path rdfs:label ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ; ] ;
```

```
This shape
    is a NodeShape
    that applies to the class “Person”
    such that
        instances are linked to rdfs:label
```

SHACL Features

- *Node Shapes* declare constraints on a node
- *Property Shapes* declare constraints on values associated nodes through sh:path
- *sh:path* declares there must be a link from a given node to a specified value

```
sh:PersonShape
    a sh:NodeShape
    sh:targetClass ex:Person
    sh:property [
        sh:path rdfs:label ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ; ] ;
```

This shape
    is a NodeShape
    that applies to the class “Person”
    such that
        instances are linked to rdfs:label
        at least once
        and at most once

SHACL Features

- *Node Shapes* declare constraints on a node
- *Property Shapes* declare constraints on values associated nodes through sh:path
- *sh:path* declares there must be a link from a given node to a specified value

```
sh:PersonShape
    a sh:NodeShape
    sh:targetClass ex:Person
    sh:property [
        sh:path rdfs:label ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ; ] ;
```

This shape
    is a NodeShape
    that applies to the class “Person”
    such that
        instances are linked to rdfs:label
        at least once
        and at most once
        where the label is a string

SHACL Features

- *sh:targetClass* declares constraints on a class
- *sh:minCount* declares sh:path must have a minimum
- *sh:maxCount* declares sh:path must have a maximum

```
sh:PersonShape
    a sh:NodeShape
    sh:targetClass ex:Person
    sh:property [
        sh:path rdfs:label ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ; ] ;
```

This shape
    is a NodeShape
    that applies to the class "Person"
    such that
        instances are linked to rdfs:label
        at least once
        and at most once
        where the label is a string

SHACL Validation Report

- SHACL processing returns an RDF graph validation report which either indicates that the associated graph sh:conforms to the shapes file, or does not.

SHACL Validation Report

- SHACL processing returns an RDF graph validation report which either indicates that the associated graph sh:conforms to the shapes file, or does not.

- If "sh:conforms false" is returned, it is accompanied by:
  - The IRI for the node that was being validated when the error occurred
  - The sh:path from the node
  - The value that violated the constraint, where applicable
  - The shape the node was validated against
  - Any user supplied message from the shapes graph
  - The user supplied severity of the violation

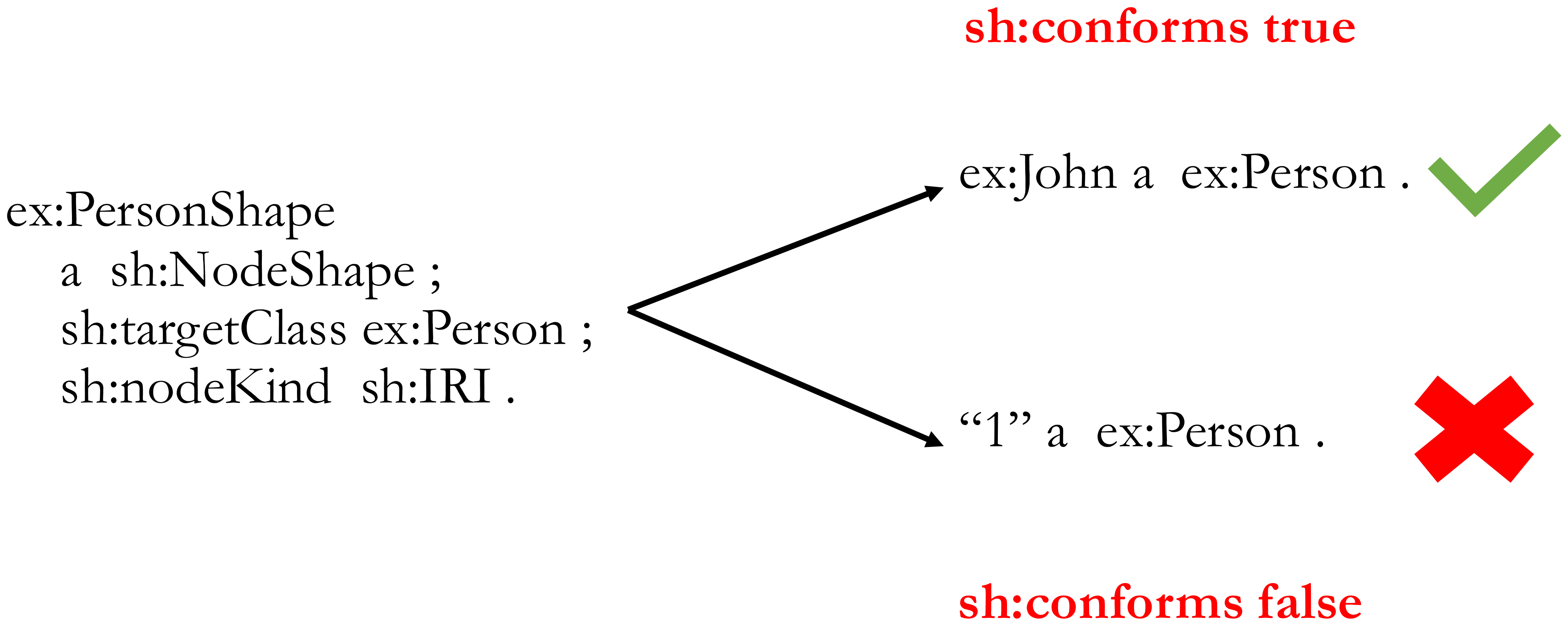
```
ex:PersonShape
    a  sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:nodeKind  sh:IRI .
```

Shapes File

```
ex:John a  ex:Person .


"1" a  ex:Person .
```

Data



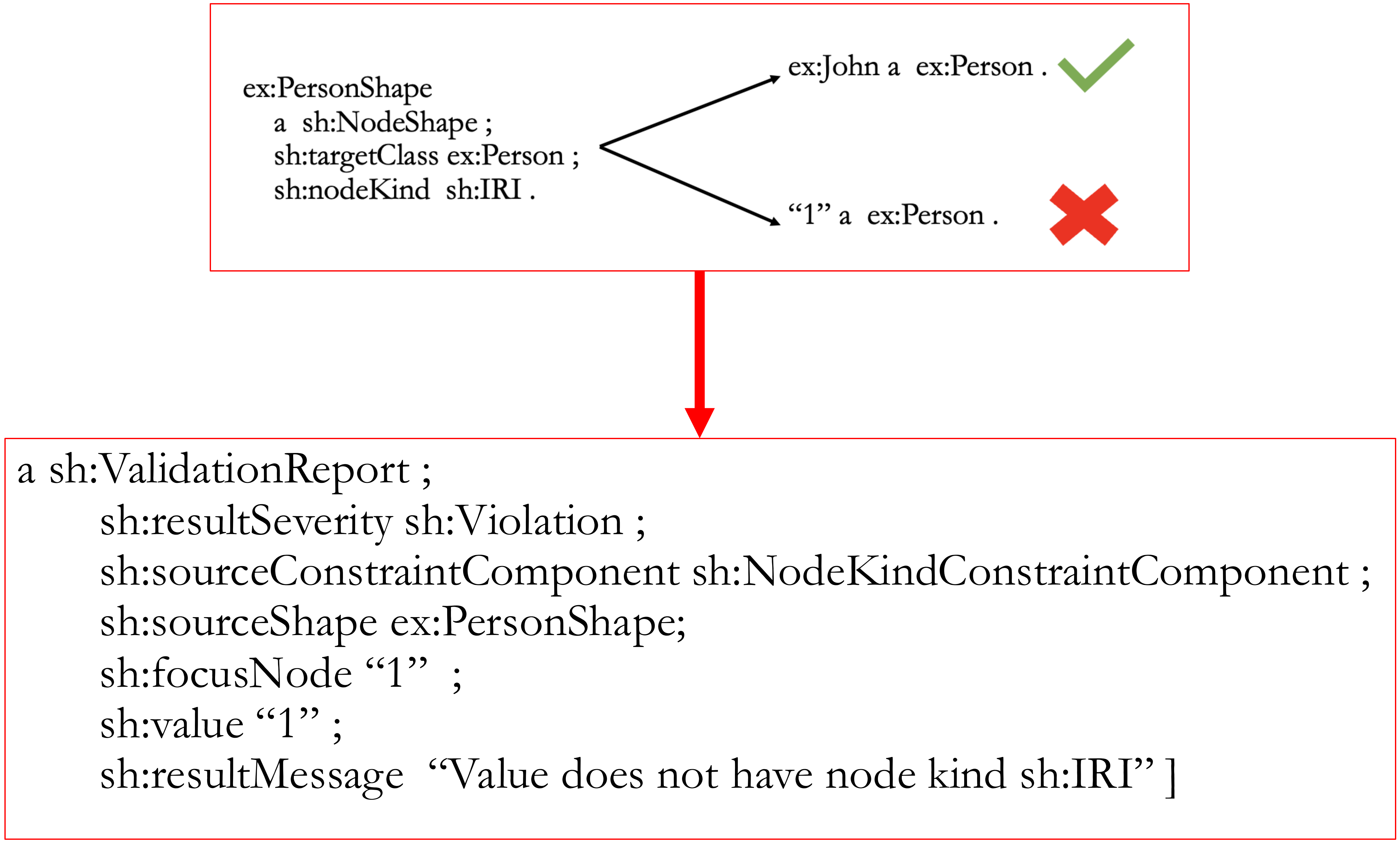
```
ex:PersonShape
    a  sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:nodeKind  sh:IRI .
```

sh:conforms true

```
ex:John a  ex:Person .
```
✓

```
"1" a  ex:Person .
```
✗

sh:conforms false



```
ex:PersonShape
   a  sh:NodeShape ;
   sh:targetClass ex:Person ;
   sh:nodeKind  sh:IRI .
```

```
ex:John a  ex:Person . ✓
```

```
"1" a  ex:Person . ✗
```

```
a sh:ValidationReport ;
      sh:resultSeverity sh:Violation ;
      sh:sourceConstraintComponent sh:NodeKindConstraintComponent ;
      sh:sourceShape ex:PersonShape;
      sh:focusNode "1"  ;
      sh:value "1" ;
      sh:resultMessage  "Value does not have node kind sh:IRI" ]
```

Messaging

sh:message declares an sh:resultMessage value returned from violation

```
ex:NameTagShape
    a sh:NodeShape ;
    sh:targetClass ex:NameTag ;
    sh:property [ sh:path ex:tags ;
                          sh:nodeKind sh:IRI ;
                          sh:class ex:Person ;
                          sh:minCount 1 ;
                          sh:maxCount 1 ;
                          sh:message “Name tags must tag exactly one person.”
```

Messaging

sh:message declares an sh:resultMessage value returned from violation

```
ex:NameTagShape
    a sh:NodeShape ;
    sh:targetClass ex:NameTag ;
    sh:property [ sh:path ex:tags ;
                    sh:nodeKind sh:IRI ;
                    sh:class ex:Person ;
                    sh:minCount 1 ;
                    sh:maxCount 1 ;
                    sh:message "Name tags must tag exactly one person."
```

More Target Practice

sh:targetSubjectsOf declares constraints on the domain of a property

sh:targetObjectsOf declares constraints on the range of a property

More Target Practice

sh:targetSubjectsOf declares constraints on the domain of a property

sh:targetObjectsOf declares constraints on the range of a property

```
ex:TagsInversePropertyShape
    a sh:NodeShape ;
    sh:targetSubjectsOf ex:tags ;
    sh:or ( [ sh:not [ sh:property
                        sh:path ex:tags ; ] ]
            [ sh:property
              sh:path [ sh:inversePath ex:tagged_by ] ;
              sh:minCount 1 ; ] ] ) ;
```

More Target Practice

sh:targetSubjectsOf declares constraints on the domain of a property

sh:targetObjectsOf declares constraints on the range of a property

```
ex:TagsInversePropertyShape
   a sh:NodeShape ;
   sh:targetSubjectsOf ex:tags ;
   sh:or ( [ sh:not [ sh:property
                    sh:path ex:tags ; ] ]
        [ sh:property
          sh:path [ sh:inversePath ex:tagged_by ] ;
          sh:minCount 1 ; ] ] ) ;
```

This node shape

$\longrightarrow$ targets subjects of ex:tags

More Target Practice

sh:or supplements constraints with logical disjunction

sh:not supplements constraints with logical negation

```
ex:TagsInversePropertyShape
   a sh:NodeShape ;
   sh:targetSubjectsOf ex:tags ;
   sh:or ( [ sh:not [ sh:property
                    sh:path ex:tags ; ] ]
           [ sh:property
             sh:path [ sh:inversePath ex:tagged_by ] ;
             sh:minCount 1 ; ] ] ) ;
```

This node shape

targets subjects of ex:tags
such that if a subject
ex:tags something

More Target Practice

sh:inversePath declare constraint applies to an inverse of sh:path

```
ex:TagsInversePropertyShape
    a sh:NodeShape ;
    sh:targetSubjectsOf ex:tags ;
    sh:or ( [ sh:not [ sh:property
                       sh:path ex:tags ; ] ]
            [ sh:property
              sh:path [ sh:inversePath ex:tagged_by ] ;
              sh:minCount 1 ; ] ] ) ;
```

This node shape

targets subjects of ex:tags

such that if a subject

ex:tags something

then the subject is ex:tagged_by

→ at least one thing

Language Constraints

sh:languageIn declares sh:path must have language value from list
sh:uniqueLang declares sh:path may have at most one language

```
ex:BelovedHasUniqueEnglishNameShape
   a sh:NodeShape ;
   sh:targetObjectsOf  ex:loves ;
   sh:property
      [ sh:path ex:has_name ;
        sh:languageIn (“en”) ;
        sh:uniqueLang true ; ]
   sh:message “Beloved objects must have a unique English name.” .
```

Language Constraints

sh:languageIn declares sh:path must have language value from list
sh:uniqueLang declares sh:path may have at most one language

```
ex:BelovedHasUniqueEnglishNameShape
   a sh:NodeShape ;
   sh:targetObjectsOf ex:loves ;
   sh:property
      [ sh:path ex:has_name ;
        sh:languageIn ("en") ;
        sh:uniqueLang true ; ]
   sh:message "Beloved objects must have a unique English name." .
```

E.g. documentation for skos:prefLabel requires values have unique language tag

Regex Constraints

sh:pattern declares constraints based on regex string matching

```
ex:BelovedHasBirthYearShape
   a sh:NodeShape ;
   sh:targetObjectsOf  ex:born ;
   sh:minLength 4 ;
   sh:maxLength 4 ;
   sh:pattern “(19|20)[0-9][0-9]”) ;
   sh:message “Beloved objects must have a birth year.” .
```

Regex Constraints

sh:pattern declares constraints based on regex string matching

```
ex:BelovedHasBirthYearShape
   a sh:NodeShape ;
   sh:targetObjectsOf ex:born ;
   sh:minLength 4 ;
   sh:maxLength 4 ;
   sh:pattern “(19|20)[0-9][0-9]”) ;
   sh:message “Beloved objects must have a birth year.” .
```

Values of ex:born must begin with either “19” or “20”

Regex Constraints

sh:pattern declares constraints based on regex string matching

```
ex:BelovedHasBirthYearShape
   a sh:NodeShape ;
   sh:targetObjectsOf ex:born ;
   sh:minLength 4 ;
   sh:maxLength 4 ;
   sh:pattern “(19|20)[0-9][0-9]”) ;
   sh:message “Beloved objects must have a birth year.” .
```

And be followed by two numbers between 0 and 9

Specific Values

sh:hasValue declares sh:path must have some specific value
sh:in declares sh:path must have a value from a specified list

```
ex:CrayonBoxShape
    a sh:NodeShape ;
    sh:targetObjectsOf ex:location_of;
    sh:property [sh:path ex:made_of; sh:hasValue ex:wax-7059 ] ;
    sh:property [sh:path ex:crayon_color; sh:in (“Red”, “Yellow”, “Green”) ] .
```

Again with Documentation Support

```
ex:NameTagShape
    a sh:NodeShape ;
    sh:targetClass ex:NameTag
    sh:property [
        sh:path ex:tags ;
        sh:Class ex:Person;
        sh:minCount 1 ;
        sh:maxCount 1 ; ] ;
```

```
SELECT ?nameTag ?person
WHERE {
    ?nameTag a ex:NameTag ;
        ex:tags ?person .
            }
```

You can read off SPARQL queries from SHACL shapes, which may support documentation

Again with Documentation Support

```
ex:NameTagShape
    a sh:NodeShape ;
    sh:targetClass ex:NameTag
    sh:property [
        sh:path ex:tags ;
        sh:Class ex:Person;
        sh:minCount 1 ;
        sh:maxCount 1 ; ] ;
```

```
SELECT ?nameTag ?person
WHERE {
    ?nameTag a ex:NameTag ;
        ex:tags ?person .
        }
```

You can read off SPARQL queries from SHACL shapes, which may support documentation

Again with Documentation Support

```
ex:NameTagShape
    a sh:NodeShape ;
    sh:targetClass ex:NameTag
    sh:property [
        sh:path ex:tags ;
        sh:Class ex:Person;
        sh:minCount 1 ;
        sh:maxCount 1 ; ] ;
```

$\longrightarrow$

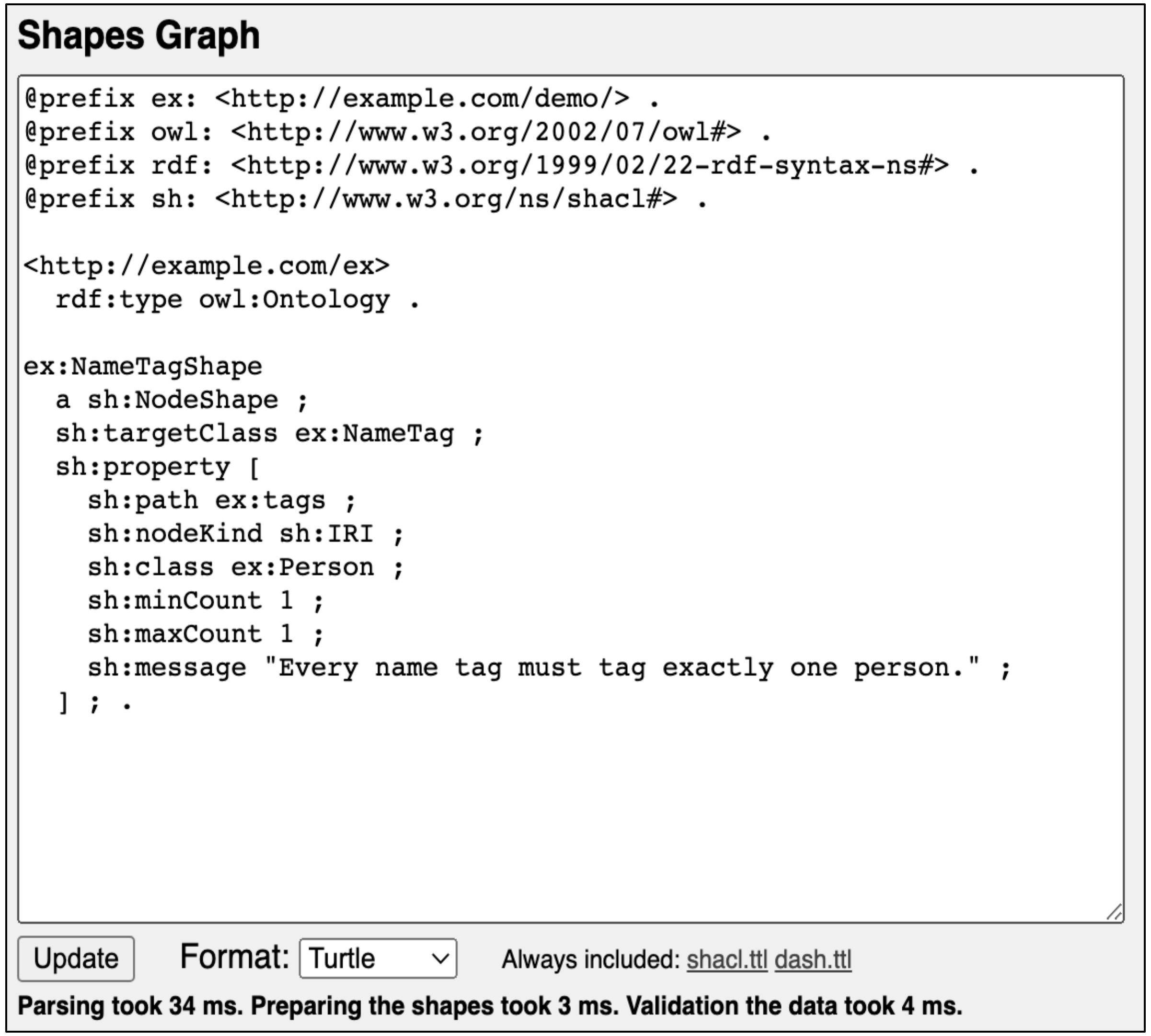
```
SELECT ?nameTag ?person
WHERE {
    ?nameTag a ex:NameTag ;
        ex:tags ?person .
        }
```

You can read off SPARQL queries from SHACL shapes, which may support documentation

Pipeline Tools

- User friendly tools exist for generating shapes graphs from ontologies:
  - Astrea - https://astrea.linkeddata.es/
  - Sparna – https://shacl-play.sparna.fr/play/validate

- Tools exist for running validation:
  - PySHACL - https://pypi.org/project/pyshacl/0.9.5/
  - SHACL Playground – https://shacl.org/playground/
  - Topbraid Composer – https://github.com/TopQuadrant/shacl
  - Protege SHACL plugin - https://github.com/fekaputra/shacl-plugin

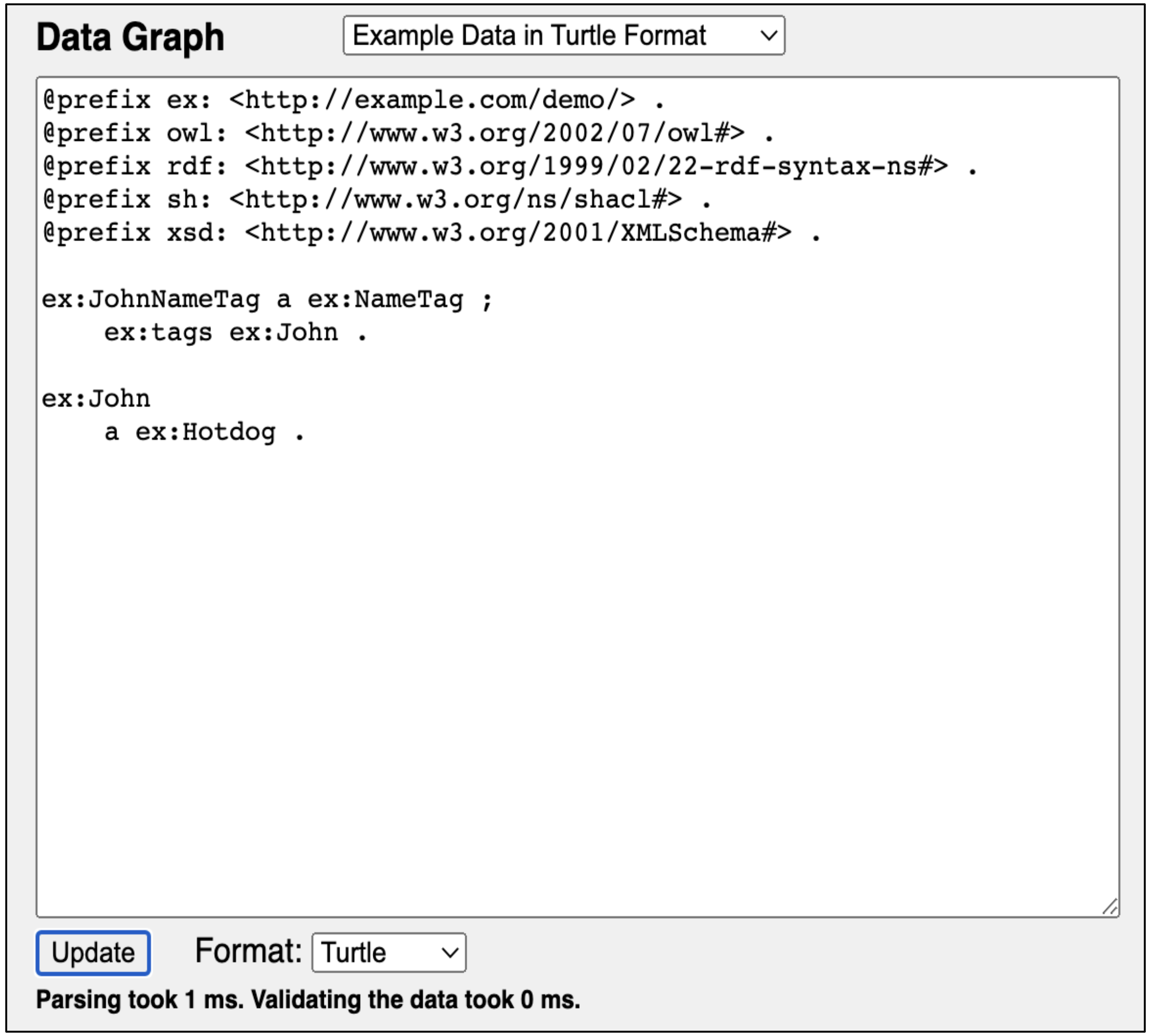

Shapes Graph

```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

<http://example.com/ex>
  rdf:type owl:Ontology .

ex:NameTagShape
  a sh:NodeShape ;
  sh:targetClass ex:NameTag ;
  sh:property [
    sh:path ex:tags ;
    sh:nodeKind sh:IRI ;
    sh:class ex:Person ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:message "Every name tag must tag exactly one person." ;
  ] ; .
```

Parsing took 34 ms. Preparing the shapes took 3 ms. Validation the data took 4 ms.


Data Graph

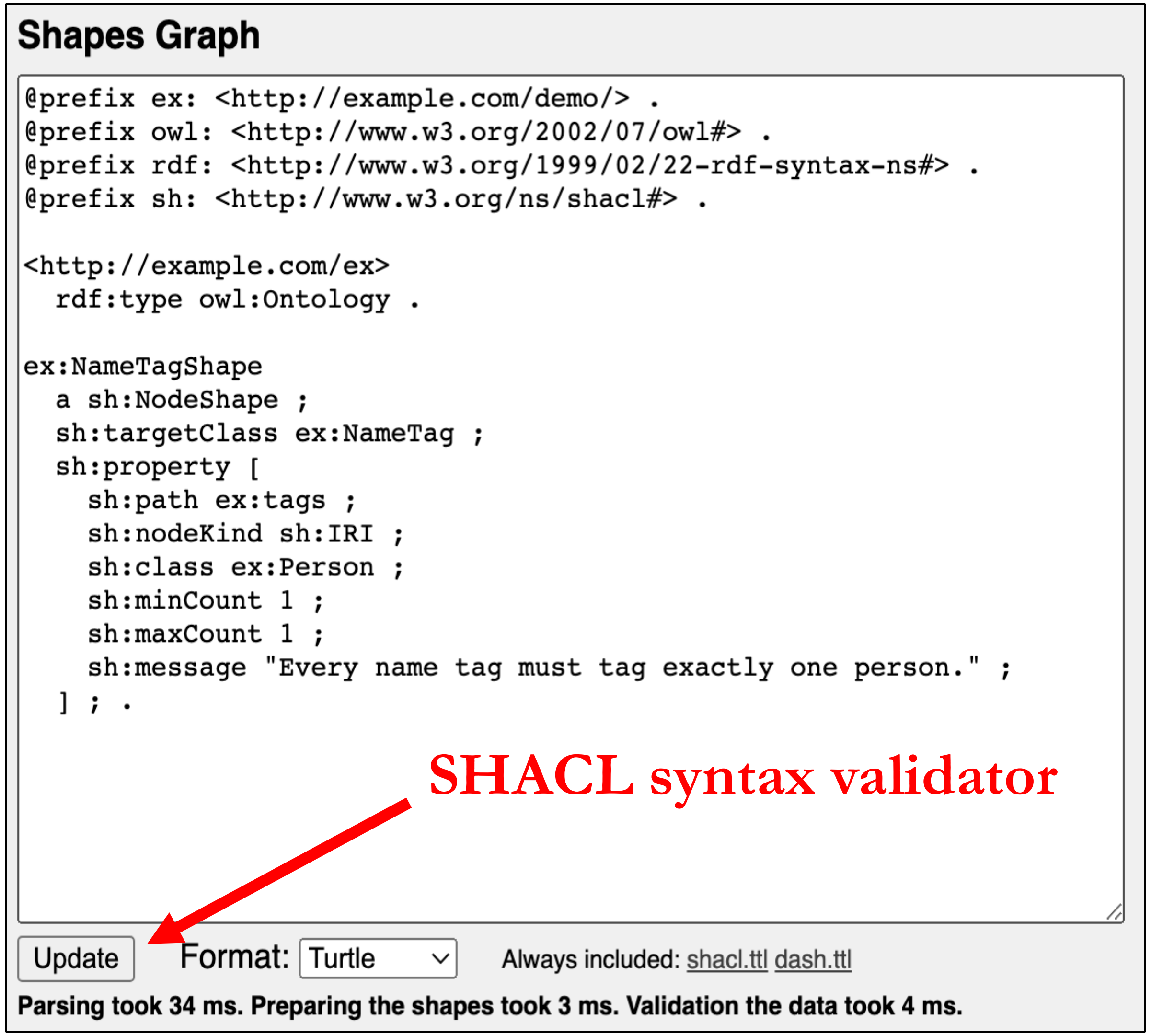
```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:JohnNameTag a ex:NameTag ;
    ex:tags ex:John .

ex:John
    a ex:Hotdog .
```

Parsing took 1 ms. Validating the data took 0 ms.

SHACL Playground – https://shacl.org/playground/



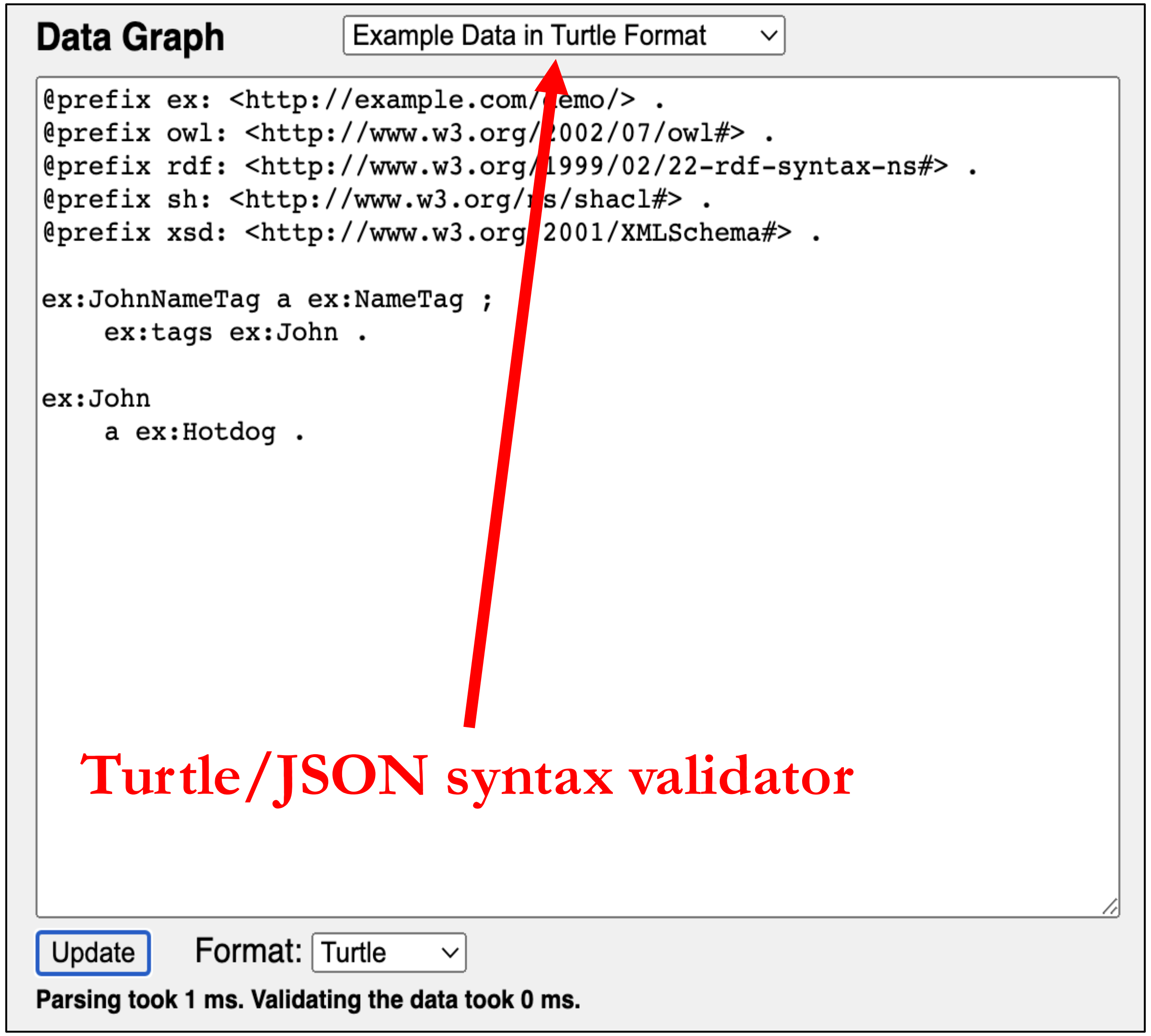
Shapes Graph

```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

<http://example.com/ex>
  rdf:type owl:Ontology .

ex:NameTagShape
  a sh:NodeShape ;
  sh:targetClass ex:NameTag ;
  sh:property [
    sh:path ex:tags ;
    sh:nodeKind sh:IRI ;
    sh:class ex:Person ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:message "Every name tag must tag exactly one person." ;
  ] ; .
```

Parsing took 34 ms. Preparing the shapes took 3 ms. Validation the data took 4 ms.



Data Graph

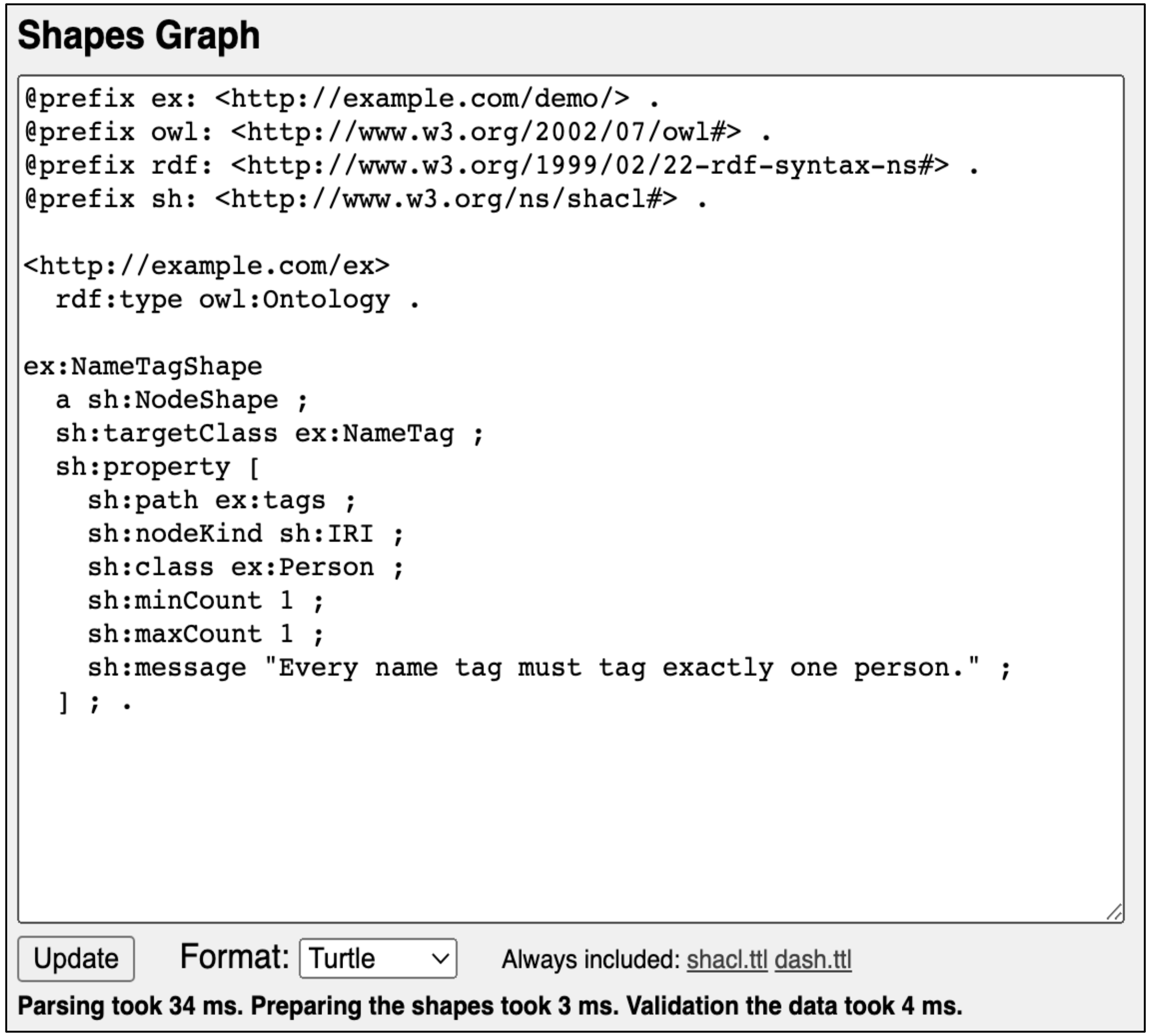
```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:JohnNameTag a ex:NameTag ;
    ex:tags ex:John .

ex:John
    a ex:Hotdog .
```

Parsing took 1 ms. Validating the data took 0 ms.

SHACL Playground – https://shacl.org/playground/



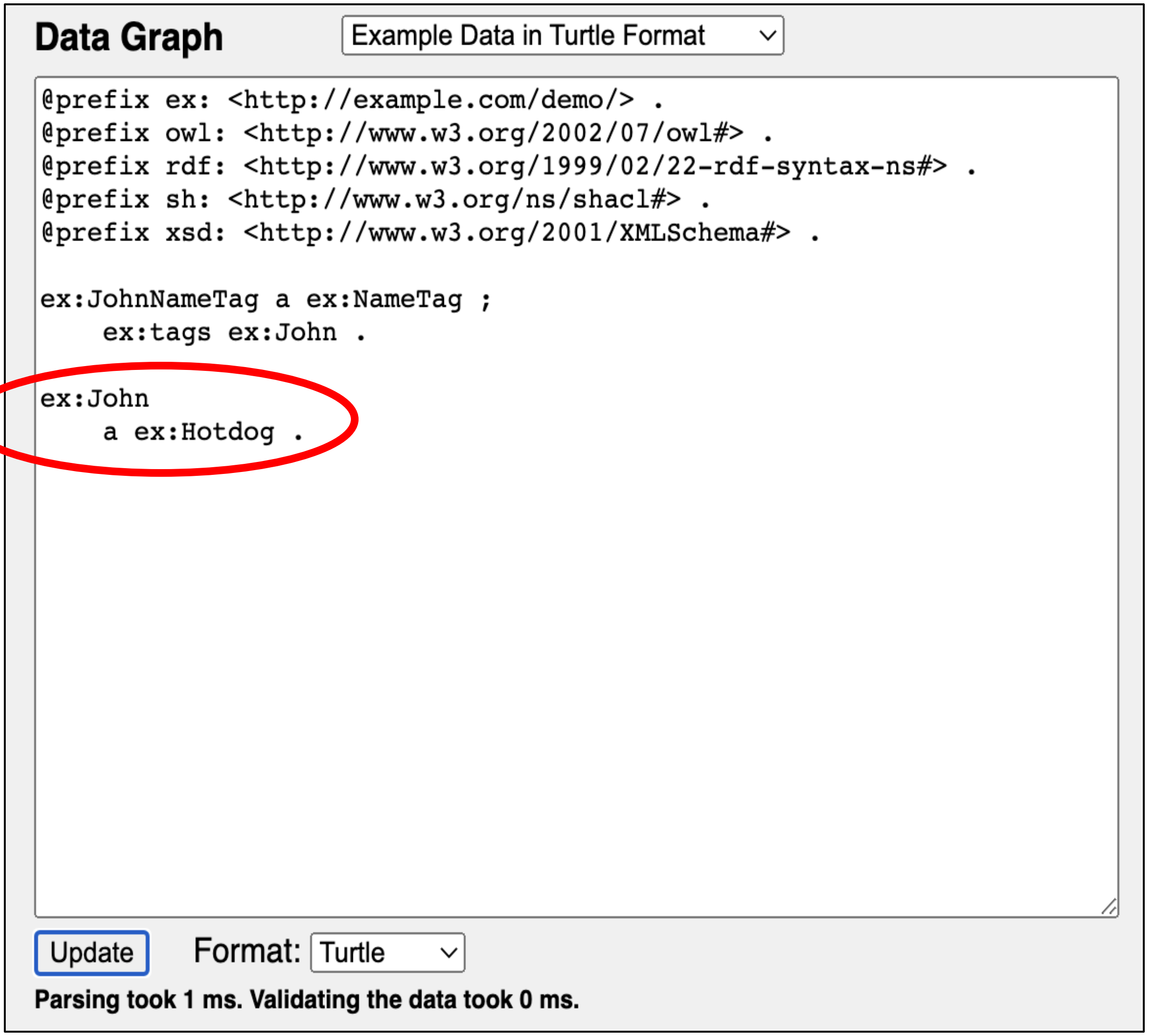
Shapes Graph

```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

<http://example.com/ex>
  rdf:type owl:Ontology .

ex:NameTagShape
  a sh:NodeShape ;
  sh:targetClass ex:NameTag ;
  sh:property [
    sh:path ex:tags ;
    sh:nodeKind sh:IRI ;
    sh:class ex:Person ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:message "Every name tag must tag exactly one person." ;
  ] ; .
```

Parsing took 34 ms. Preparing the shapes took 3 ms. Validation the data took 4 ms.



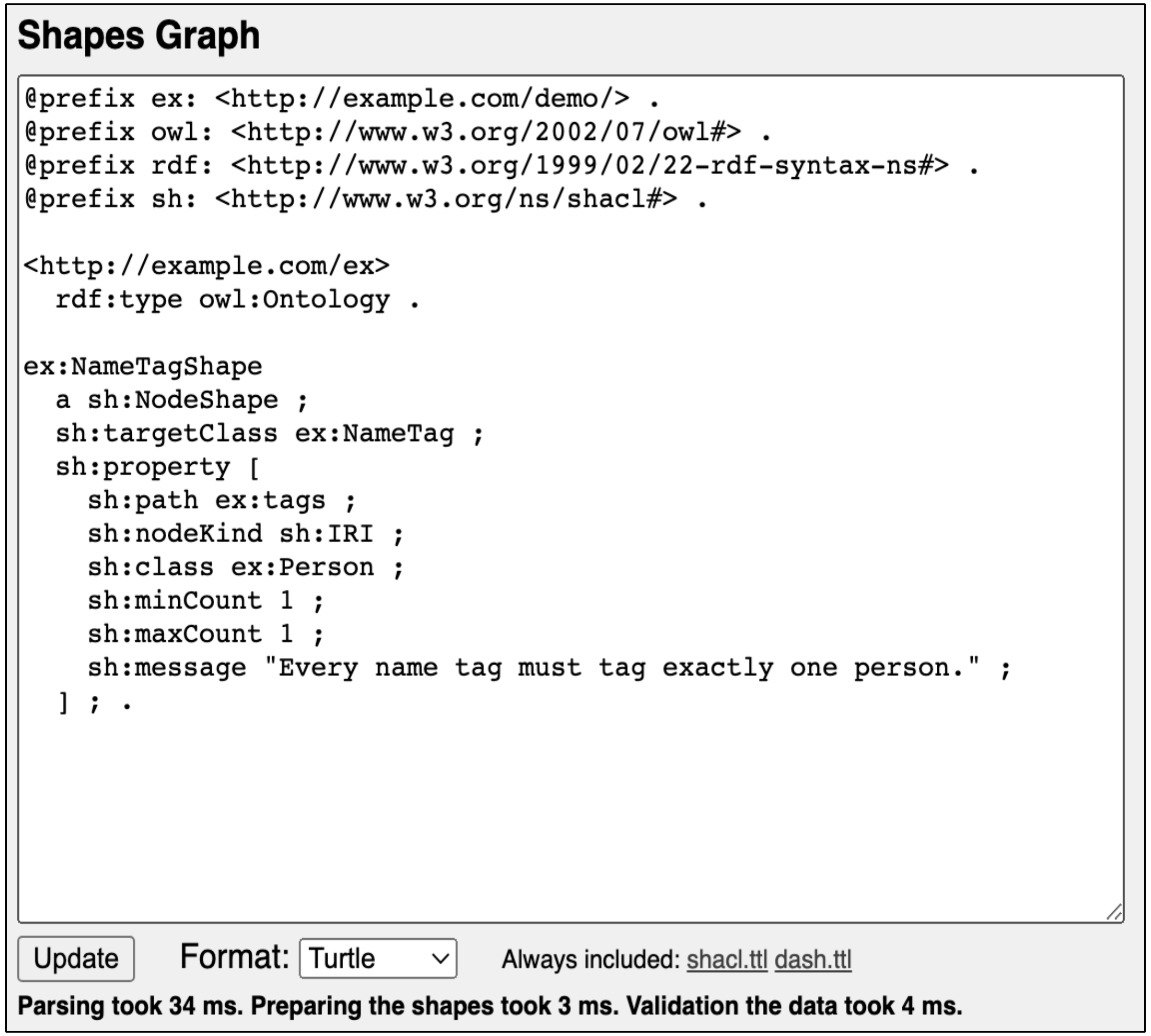
Data Graph

```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:JohnNameTag a ex:NameTag ;
    ex:tags ex:John .

ex:John
    a ex:Hotdog .
```

Parsing took 1 ms. Validating the data took 0 ms.



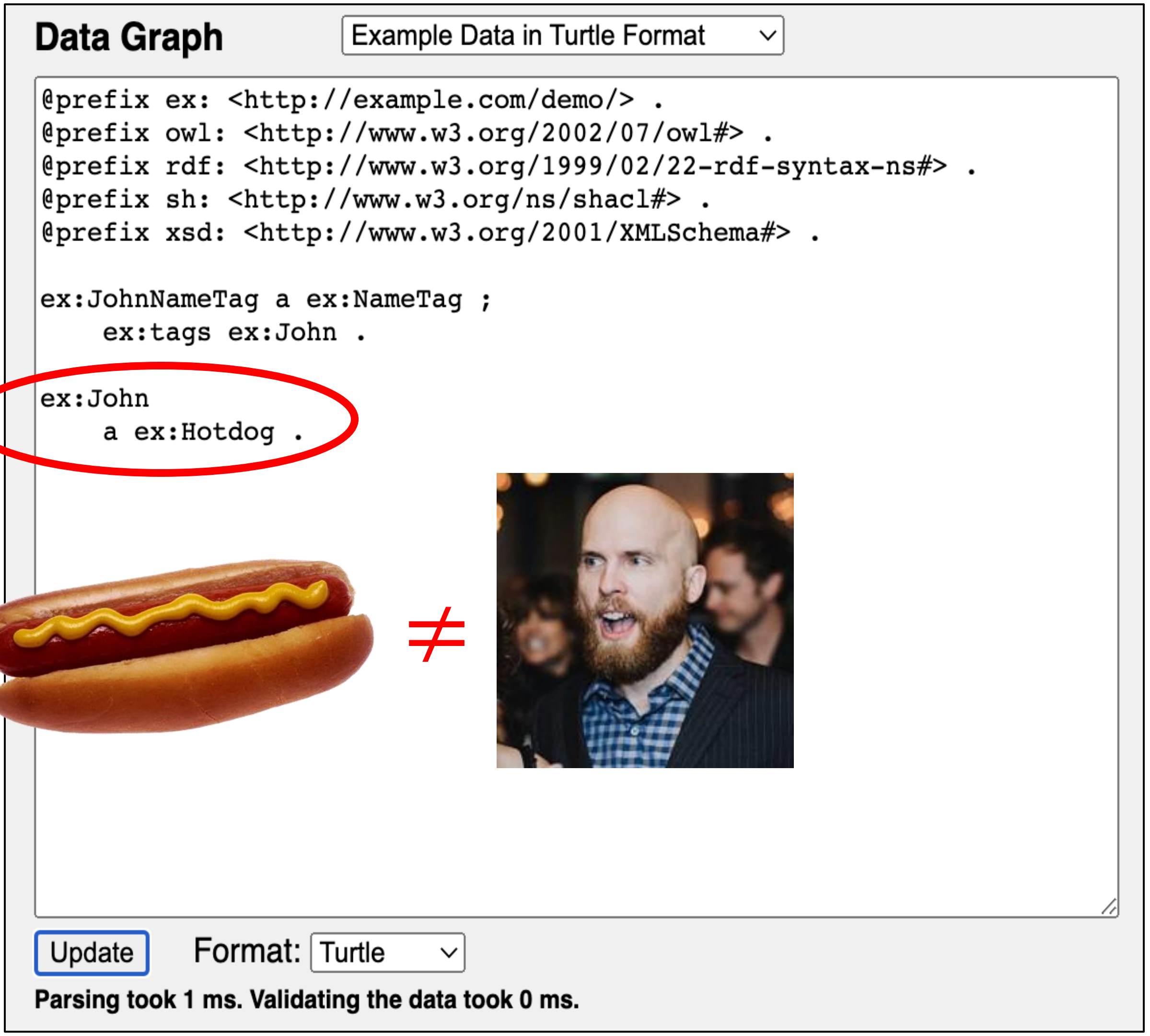
Shapes Graph

```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

<http://example.com/ex>
  rdf:type owl:Ontology .

ex:NameTagShape
  a sh:NodeShape ;
  sh:targetClass ex:NameTag ;
  sh:property [
    sh:path ex:tags ;
    sh:nodeKind sh:IRI ;
    sh:class ex:Person ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:message "Every name tag must tag exactly one person." ;
  ] ; .
```

Parsing took 34 ms. Preparing the shapes took 3 ms. Validation the data took 4 ms.



Data Graph

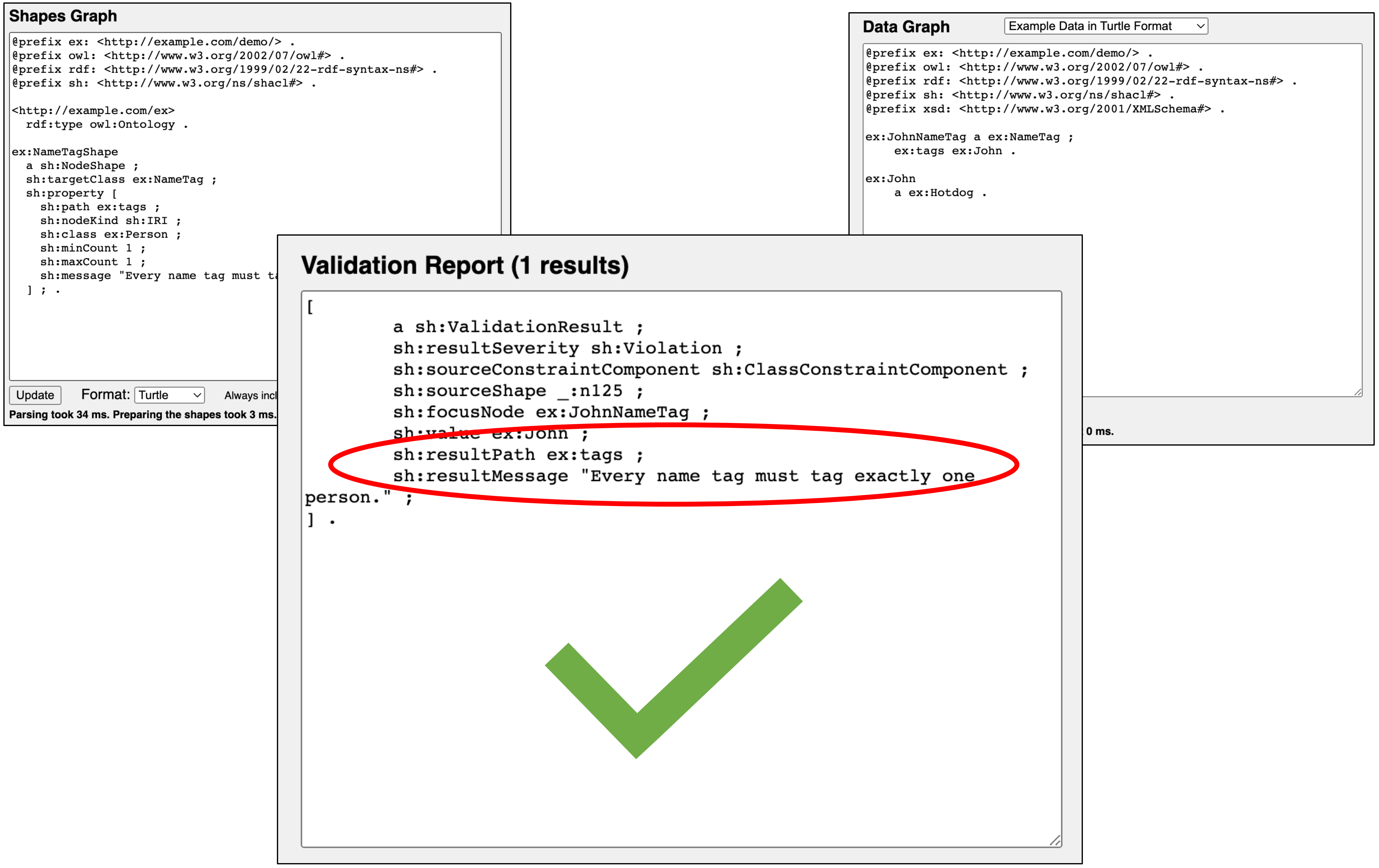
```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:JohnNameTag a ex:NameTag ;
    ex:tags ex:John .

ex:John
    a ex:Hotdog .
```

$\neq$

Parsing took 1 ms. Validating the data took 0 ms.



Shapes Graph

```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

<http://example.com/ex>
  rdf:type owl:Ontology .

ex:NameTagShape
  a sh:NodeShape ;
  sh:targetClass ex:NameTag ;
  sh:property [
    sh:path ex:tags ;
    sh:nodeKind sh:IRI ;
    sh:class ex:Person ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:message "Every name tag must t
  ] ; .
```

Parsing took 34 ms. Preparing the shapes took 3 ms.

Data Graph

```
@prefix ex: <http://example.com/demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:JohnNameTag a ex:NameTag ;
    ex:tags ex:John .

ex:John
    a ex:Hotdog .
```

0 ms.

Validation Report (1 results)

```
[
        a sh:ValidationResult ;
        sh:resultSeverity sh:Violation ;
        sh:sourceConstraintComponent sh:ClassConstraintComponent ;
        sh:sourceShape _:n125 ;
        sh:focusNode ex:JohnNameTag ;
        sh:value ex:John ;
        sh:resultPath ex:tags ;
        sh:resultMessage "Every name tag must tag exactly one
person." ;
] .
```

Integrating SHACL

- Use an existing tool to auto-generate shapes file for ontology, e.g. astrea rest API
- Bundle ontology build deliverable with shapes file and integrated validation tool, e.g. PySHACL
- Use validation reports to revise ontology, improve documentation, extract useful SPARQL queries

Integrating SHACL

- For example, users tagging data using terms from an ontology may generate a knowledge graph

- Which can be validated against a shapes file bundled with the ontology, with the shapes perhaps in the ontology file itself

- Erroneous tagging can be detected early by SHACL violation reports, e.g. generated by a user commit, end of day build check, etc.

Outline

- Data Validation
- Motivating SHACL
- SHACL Features
- Exercises

Easy Exercise

Please gather in groups and create a SHACL constraint reflecting the following:

All library books must have exactly one title and exactly one publication year later than 1900

Review the W3C documentation as guidance and validate your shape using one or more online validators

Easy Exercise

Please gather in groups and create a SHACL constraint reflecting the following:

Every student in a university has a surname, is over 18, and enrolled in at least one course; professors teach at least one course

Review the W3C documentation as guidance and validate your shape using one or more online validators

Easy Exercise

Please gather in groups and create a SHACL constraint reflecting the following:

Products have an English name, positive decimal price, category, and are optionally discounted; if discounted, the category must be "sale", else it's either new or used

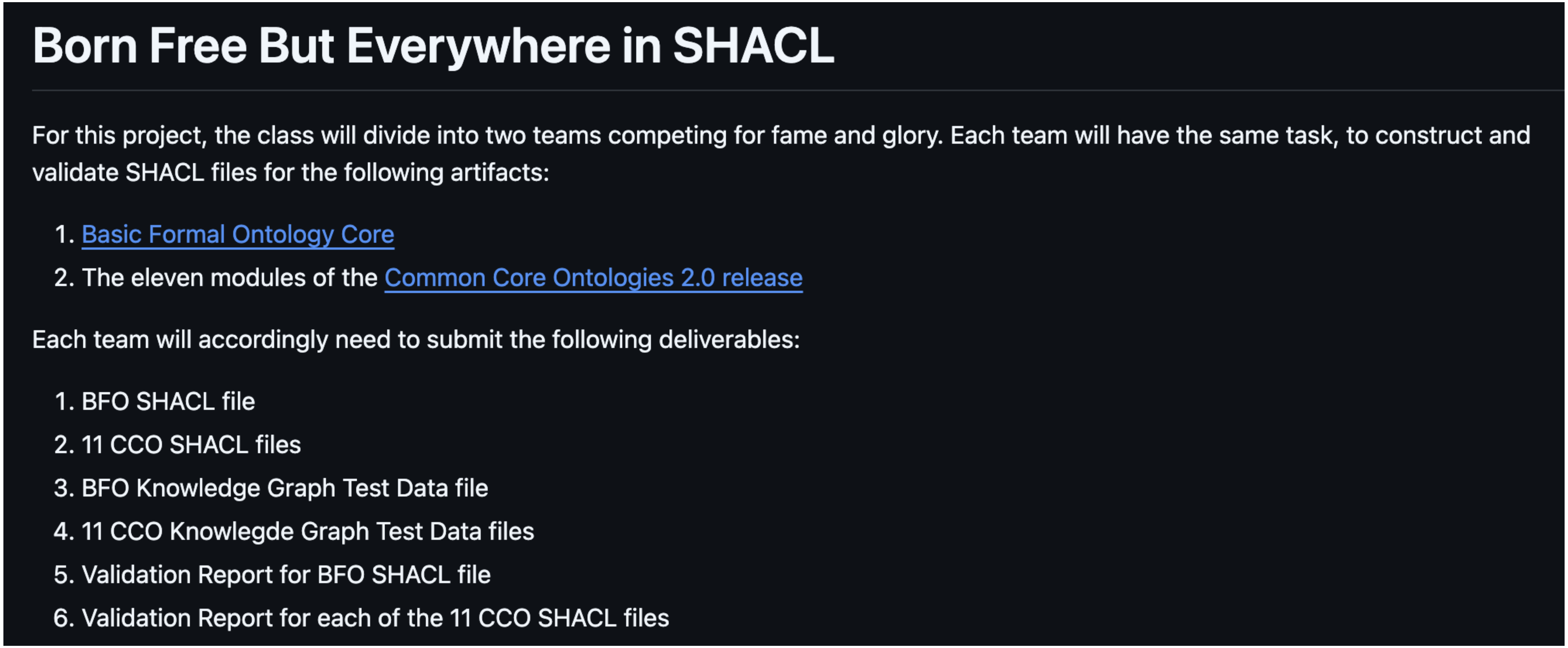
Review the W3C documentation as guidance and validate your shape using one or more online validators

Easy Exercise

Please gather in groups and create a SHACL constraint reflecting the following:

A foot has at most five toes, exactly one of which is a big toe, and all of them are disjoint from each other

Review the W3C documentation as guidance and validate your shape using one or more online validators



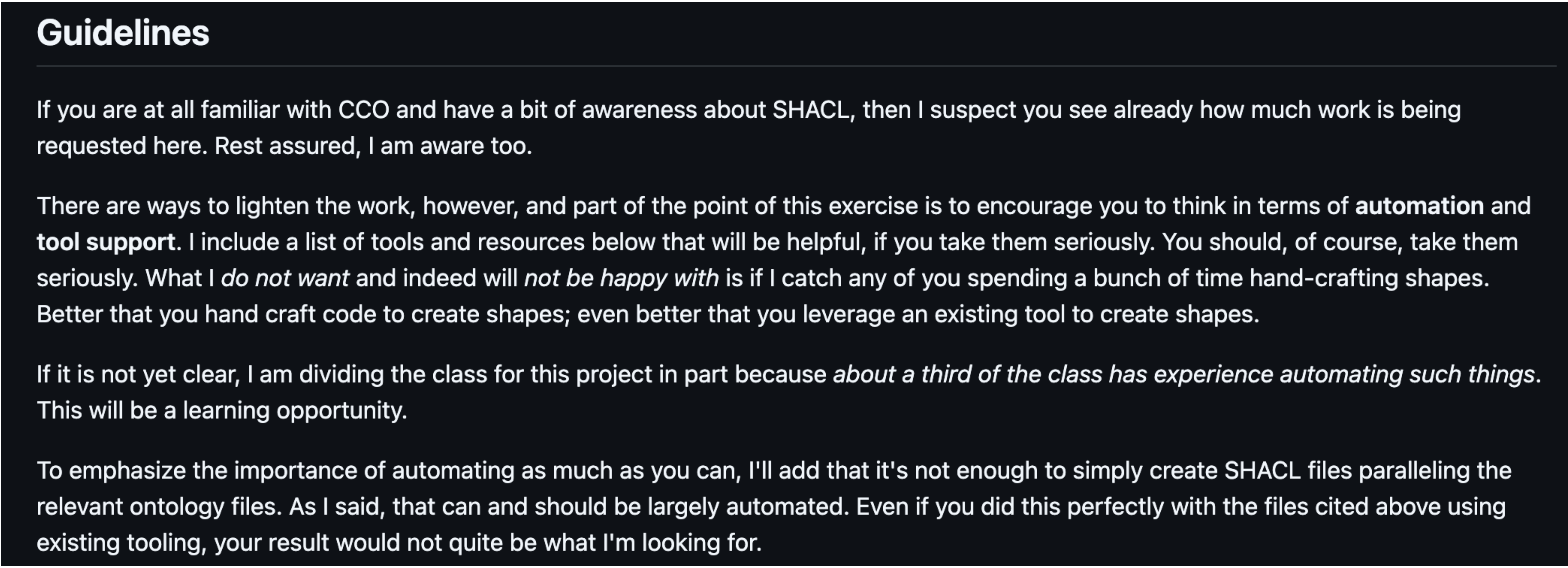
Born Free But Everywhere in SHACL

For this project, the class will divide into two teams competing for fame and glory. Each team will have the same task, to construct and validate SHACL files for the following artifacts:

1. Basic Formal Ontology Core
2. The eleven modules of the Common Core Ontologies 2.0 release

Each team will accordingly need to submit the following deliverables:

1. BFO SHACL file
2. 11 CCO SHACL files
3. BFO Knowledge Graph Test Data file
4. 11 CCO Knowlegde Graph Test Data files
5. Validation Report for BFO SHACL file
6. Validation Report for each of the 11 CCO SHACL files



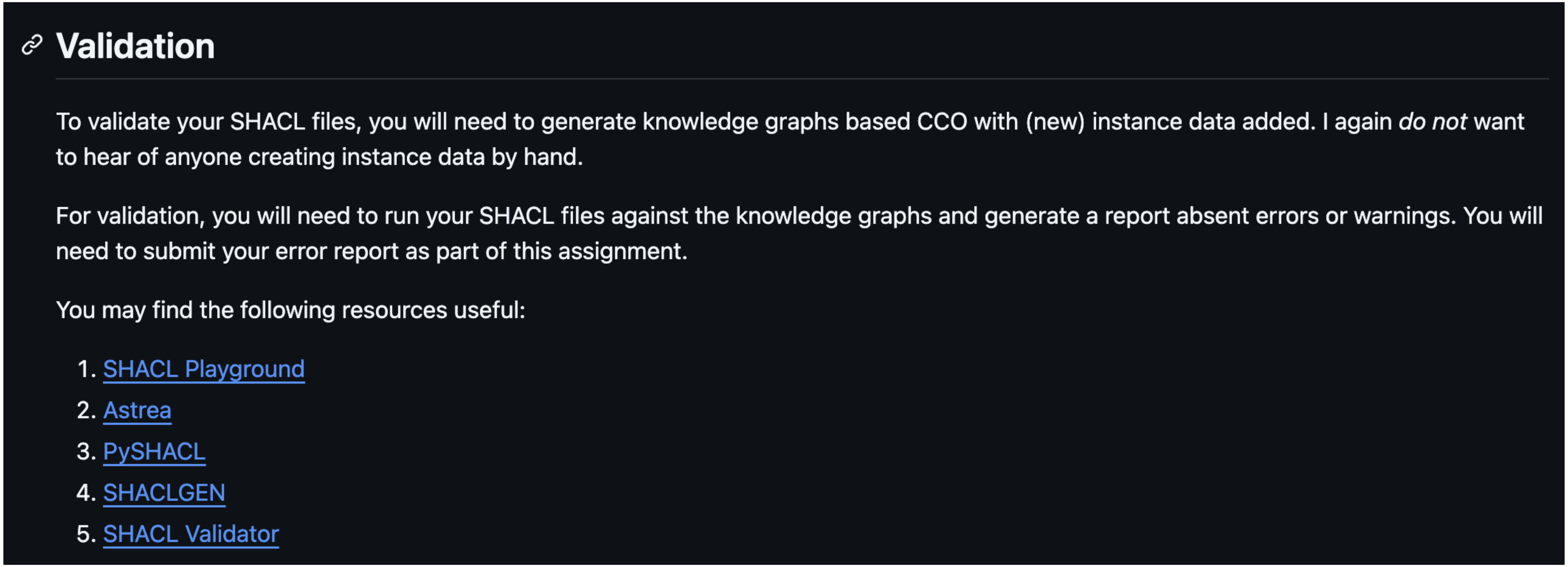
Guidelines

If you are at all familiar with CCO and have a bit of awareness about SHACL, then I suspect you see already how much work is being requested here. Rest assured, I am aware too.

There are ways to lighten the work, however, and part of the point of this exercise is to encourage you to think in terms of **automation** and **tool support**. I include a list of tools and resources below that will be helpful, if you take them seriously. You should, of course, take them seriously. What I *do not want* and indeed will *not be happy with* is if I catch any of you spending a bunch of time hand-crafting shapes. Better that you hand craft code to create shapes; even better that you leverage an existing tool to create shapes.

If it is not yet clear, I am dividing the class for this project in part because *about a third of the class has experience automating such things*. This will be a learning opportunity.

To emphasize the importance of automating as much as you can, I'll add that it's not enough to simply create SHACL files paralleling the relevant ontology files. As I said, that can and should be largely automated. Even if you did this perfectly with the files cited above using existing tooling, your result would not quite be what I'm looking for.



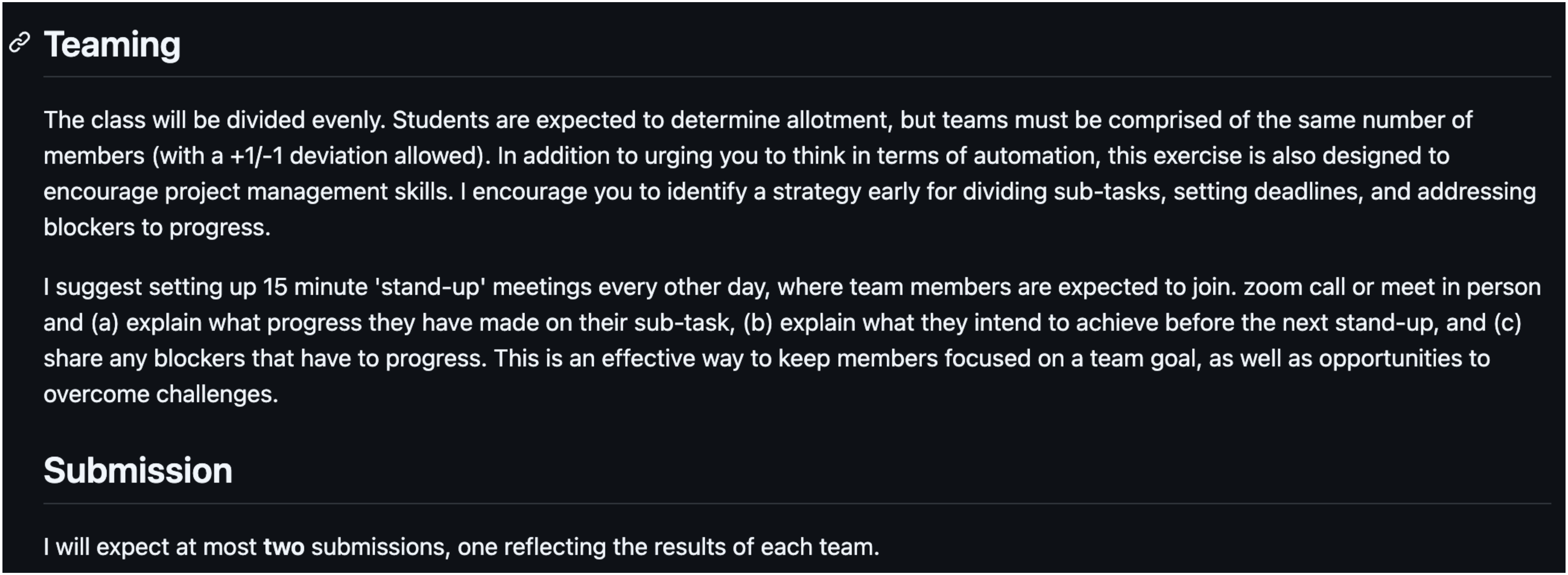
Validation

To validate your SHACL files, you will need to generate knowledge graphs based CCO with (new) instance data added. I again *do not* want to hear of anyone creating instance data by hand.

For validation, you will need to run your SHACL files against the knowledge graphs and generate a report absent errors or warnings. You will need to submit your error report as part of this assignment.

You may find the following resources useful:

1. SHACL Playground
2. Astrea
3. PySHACL
4. SHACLGEN
5. SHACL Validator



Teaming

The class will be divided evenly. Students are expected to determine allotment, but teams must be comprised of the same number of members (with a +1/-1 deviation allowed). In addition to urging you to think in terms of automation, this exercise is also designed to encourage project management skills. I encourage you to identify a strategy early for dividing sub-tasks, setting deadlines, and addressing blockers to progress.

I suggest setting up 15 minute 'stand-up' meetings every other day, where team members are expected to join. zoom call or meet in person and (a) explain what progress they have made on their sub-task, (b) explain what they intend to achieve before the next stand-up, and (c) share any blockers that have to progress. This is an effective way to keep members focused on a team goal, as well as opportunities to overcome challenges.

Submission

I will expect at most **two** submissions, one reflecting the results of each team.