

SHACL by example

RDF Validation tutorial

Jose Emilio Labra Gayo

WESO Research group
University of Oviedo, Spain

Eric Prud'hommeaux

World Wide Web Consortium
MIT, Cambridge, MA, USA

Harold Solbrig

Mayo Clinic, USA

Iovka Boneva

LINKS, INRIA & CNRS
University of Lille, France



SHACL

W3c Data Shapes WG deliverable

<https://www.w3.org/TR/shacl/>

Inspired by SPIN, OSLC & bits of ShEx

SPARQL based extension mechanism

RDF vocabulary

No human friendly syntax yet*

* A human friendly syntax inspired by ShEx is being considered by the WG



Some definitions about SHACL

Shape: collection of scopes, filters and constraints

Scopes: specify which nodes in the data graph must follow the shape

Filters: Further limit the scope nodes to those that satisfy the filter

Constraints: Determine how to validate a node

Shapes graph: an RDF graph that contains shapes

Data graph: an RDF graph that contains data to be validated

Example

```
prefix : <http://example.org/>
prefix sh: <http://www.w3.org/ns/shacl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>
```

```
:UserShape a sh:Shape ;
  sh:scopeNode :alice, :bob, :carol ;
  sh:property [
    sh:predicate schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:predicate schema:email ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:nodeKind sh:IRI ;
  ] .
```

UserShape
foaf:name :xsd:string
foaf:mbox :IRI

:alice schema:name "Alice Cooper" ;
 schema:email <mailto:alice@mail.org> .

:bob schema:firstName "Bob" ;
 schema:email <mailto:bob@mail.org> . 😞

:carol schema:name "Carol" ;
 schema:email "carol@mail.org" . 😞

Shapes graph

Data graph

Try it. RDFShape <http://goo.gl/FqXQpD>

Scopes

Scopes specify nodes that must be validated against the shape

Several types

Value	Description
scopeNode	Directly point to a node
scopeClass	All nodes that have a given type
scopeProperty	All nodes that have a given property
scope	General mechanism based on SPARQL



Scope node

Directly declare which nodes must validate the against the shape

```
:UserShape a sh:Shape ;  
  sh:scopeNode :alice, :bob, :carol ;  
  sh:property [  
    sh:predicate schema:name ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:datatype xsd:string ;  
  ] ;  
  sh:property [  
    sh:predicate schema:email ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:nodeKind sh:IRI ;  
  ] .
```

```
:alice schema:name "Alice Cooper" ;  
      schema:email <mailto:alice@mail.org> .  
  
:bob schema:givenName "Bob" ;  
      schema:email <mailto:bob@mail.org> .  
  
:carol schema:name "Carol" ;  
       schema:email "carol@mail.org" .
```

Scope class

Selects all nodes that have a given type

Looks for `rdf:type` declarations*

```
:UserShape a sh:Shape ;  
sh:scopeClass :User ;  
sh:property [  
    sh:predicate schema:name ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:datatype xsd:string ;  
] ;  
sh:property [  
    sh:predicate schema:email ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:nodeKind sh:IRI ;  
] .
```

```
:alice a :User;  
schema:name "Alice Cooper" ;  
schema:email <mailto:alice@mail.org> .  
  
:bob a :User;  
schema:givenName "Bob" ;  
schema:email <mailto:bob@mail.org> .  
  
:carol a :User ;  
schema:name "Carol" ;  
schema:email "carol@mail.org" .
```

* Also looks for `rdfs:subClassOf*/rdf:type` declarations



Implicit scope class

A shape with type sh:Shape and rdfs:Class is a scope class of itself

The scopeClass declaration is implicit

```
:User a sh:Shape, rdfs:Class ;
  sh:property [
    sh:predicate schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:predicate schema:email ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:nodeKind sh:IRI ;
  ] .
```

```
:alice a :User;
  schema:name "Alice Cooper" ;
  schema:email <mailto:alice@mail.org> .

:bob a :User;
  schema:givenName "Bob" ;
  schema:email <mailto:bob@mail.org> .

:carol a :User;
  schema:name "Carol" ;
  schema:email "carol@mail.org" .
```

Constraints

Types of constraints

Type	Description
Node constraints	Constraints about a given focus node
Property constraints	Constraints about a property and the values of that property for a node
Inverse property constraints	Constraints about a property and the inverse values of that property for a node
General constraints	General mechanism based on SPARQL



Node Constraints

Constraints about a focus node

```
:User a sh:Shape ;  
  sh:constraint [  
    sh:nodeKind sh:IRI ;  
  ] .
```

```
:alice a :User .  
  
<http://example.org/bob> a :User .  
  
_:1 a :User . 😞
```



Property constraints

Constraints about a given property and its values for the focus node

`sh:property` associates a shape with a property constraint

`sh:predicate` identifies the predicate

```
:User a sh:Shape ;  
  sh:property [  
    sh:predicate schema:email ;  
    sh:nodeKind sh:IRI  
  ] .
```

```
:alice a :User ;  
  schema:email <mailto:alice@mail.org> .  
  
:bob   a :User;  
  schema:email <mailto:bob@mail.org> .  
  
:carol a :User;  
  schema:email "carol@mail.org" .
```

Inverse property constraints

Constraints about a given property and its *inverse* values for the focus node

`sh:inverseProperty` associates shape with inverse property constraint
`sh:predicate` identifies the predicate

```
:User a sh:Shape, rdfs:Class ;  
  sh:inverseProperty [  
    sh:predicate schema:follows ;  
    sh:nodeKind sh:IRI ;  
  ] .
```

```
:alice a :User;  
  schema:follows :bob .  
  
:bob a :User .  
  ☹  
  
:carol a :User;  
  schema:follows :alice .  
  
_:1 schema:follows :bob .
```

Core constraint components

Type	Constraints
Cardinality	minCount, maxCount
Types of values	class, datatype, nodeKind, classIn, datatypeIn
Values	valueShape, in, hasValue
Range of values	minInclusive, maxInclusive minExclusive, maxExclusive
String based	minLength, maxLength, pattern, stem, uniqueLang
Logical constraints	not, and, or
Closed shapes	closed, ignoredProperties
Property pair constraints	equals, disjoint, lessThan, lessThanOrEquals
Non-validating constraints	name, value, defaultValue
Partitions	partition qualifiedValueShape, qualifiedMinCount, qualifiedMaxCount



Cardinality constraints

Constraint	Description
minCount	Restricts minimum number of triples involving the focus node and a given predicate. Default value: 0
maxCount	Restricts maximum number of triples involving the focus node and a given predicate. If not defined = unbounded

```
:User a sh:Shape ;  
  sh:property [  
    sh:predicate schema:follows ;  
    sh:minCount 2 ;  
    sh:maxCount 3 ;  
  ] .
```

```
:alice schema:follows :bob,  
          :carol .  
  
:bob   schema:follows :alice .   
  
:carol schema:follows :alice,  
        :bob,  
        :carol,  
        :dave . 
```

Try it. <http://goo.gl/9AwtFK>

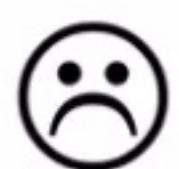


Datatypes of values

Constraint	Description
datatype	Restrict the datatype of all value nodes to a given value
datatypeIn	Restrict the datatype of all value nodes to a given list of values

```
:User a sh:Shape ;  
  sh:property [  
    sh:predicate schema:birthDate ;  
    sh:datatype xsd:date ;  
  ];  
  sh:property [  
    sh:predicate schema:jobTitle ;  
    sh:datatypeIn (  
      xsd:string  
      rdf:langString  
    )  
  ].
```

```
:alice schema:birthDate "1985-08-20"^^xsd:date;  
      schema:jobTitle "CEO", "Director"@es .  
  
:bob   schema:birthDate "2007-08-20"^^xsd:date;  
       schema:jobTitle :unknown .  
  
:carol schema:birthDate 1990 ;  
       schema:jobTitle "CTO" .
```



Try it: <http://goo.gl/eDwxsU>

Class of values

Constraint	Description
class	Verify that each node in an instance of some class It also allows instances of subclasses*
classIn	Verify that each node in an instance of some type in a list

(*) The notion of SHACL instance is different from RDFS
It is defined as rdfs:subClassOf*/rdf:type

```
:User a sh:Shape, rdfs:Class ;  
sh:property [  
    sh:predicat schema:follows ;  
    sh:class :User  
] .
```

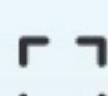
```
:Manager rdfs:subClassOf :User .  
  
:alice a :User;  
    schema:follows :bob .  
:bob a :Manager ;  
    schema:follows :alice .  
:carol a :User;  
    schema:follows :alice, :dave . 😞  
  
:dave a :Employee .
```

Kind of values

Constraint	Description
nodeKind	Possible values: BlankNode, IRI, Literal, BlankNodeOrIRI, BlankNodeOrLiteral, IRIOrLiteral

```
:User a sh:Shape, rdfs:Class ;  
  sh:property [  
    sh:predicate schema:name ;  
    sh:nodeKind sh:Literal ;  
  ] ;  
  sh:property [  
    sh:predicate schema:follows ;  
    sh:nodeKind sh:BlankNodeOrIRI  
  ] ;  
  sh:constraint [  
    sh:nodeKind sh:IRI  
  ] .
```

```
:alice a :User;  
  schema:name _:1 ;  
  schema:follows :bob .  
  
:bob a :User;  
  schema:name "Robert";  
  schema:follows [ schema:name "Dave" ] .  
  
:carol a :User;  
  schema:name "Carol" ;  
  schema:follows "Dave" .  
  
_:1 a :User .
```



Constraints on values

Constraint	Description
hasValue	Verifies that the focus node has a given value
in	Enumerates the value nodes that a property may have

```
:User a sh:Shape, rdfs:Class ;
  sh:property [
    sh:property schema:affiliation ;
    sh:hasValue :OurCompany ;
  ];
  sh:property [
    sh:property schema:gender ;
    sh:in (schema:Male schema:Female)
  ] .
```

```
:alice a :User;
  schema:affiliation :OurCompany ;
  schema:gender schema:Female .

:bob a :User;
  schema:affiliation :AnotherCompany ; 😞
  schema:gender schema:Male .

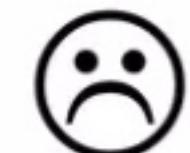
:carol a :User;
  schema:affiliation :OurCompany ;
  schema:gender schema:Unknown . 😞
```

Constraints on values with another shape

Constraint	Description
valueShape*	All values of a given property must have a given shape Recursion is not allowed in current SHACL

```
:User a sh:Shape, rdfs:Class ;  
  sh:property [  
    sh:predicat schema:worksFor ;  
    sh:valueShape :Company ;  
  ] .  
  
:Company a sh:Shape ;  
  sh:property [  
    sh:predicat schema:name ;  
    sh:datatype xsd:string ;  
  ] .
```

```
:alice a :User;  
  schema:worksFor :OurCompany .  
  
:bob   a :User;  
  schema:worksFor :Another .  
  
:OurCompany  
  schema:name "OurCompany" .  
  
:Another  
  schema:name 23 .
```

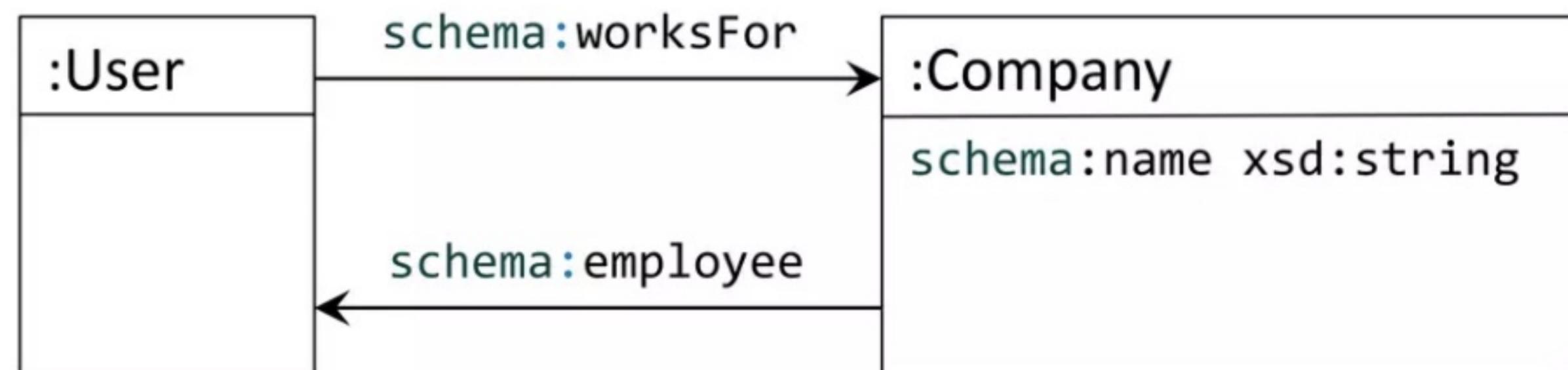


*recently renamed as sh:shape

Value shapes and recursion

Could we define cyclic data models as the following?

```
:User a sh:Shape ;  
  sh:property [  
    sh:predicate schema:worksFor ;  
    sh:valueShape :Company ;  
  ] .  
  
:Company a sh:Shape ;  
  sh:property [  
    sh:predicate schema:name ;  
    sh:datatype xsd:string ;  
  ] ;  
  sh:property [  
    sh:predicate schema:employee ;  
    sh:valueShape :User ;  
  ] .
```



```
:alice schema:worksFor :OneCompany .  
:bob   schema:worksFor :OneCompany .  
:carol  schema:worksFor :OneCompany .  
  
:OneCompany schema:name "One" ;  
            schema:employee :alice, :bob, :carol .
```

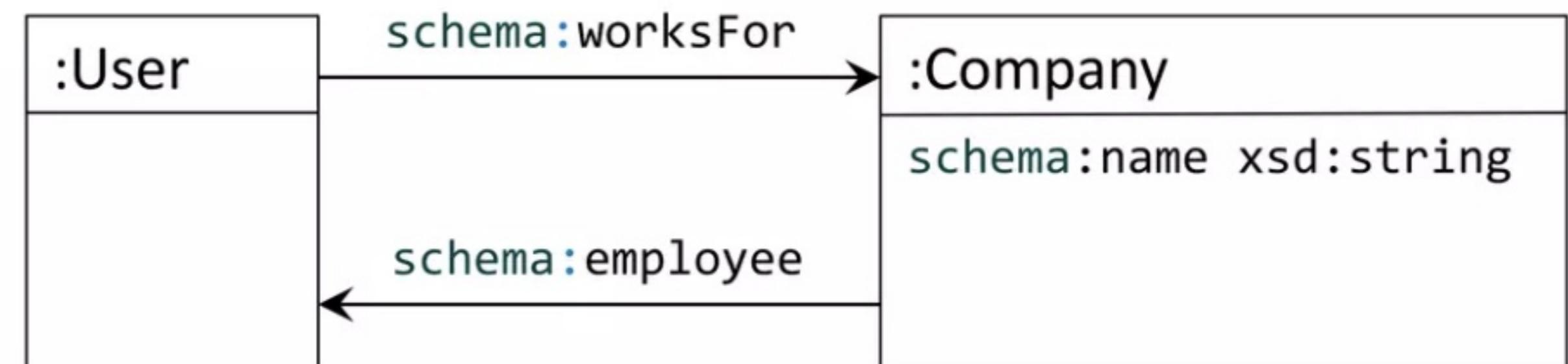
No, current SHACL specification doesn't allow this

Don't try it 😞

SHACL approach to avoid recursion

Add rdf:type arcs for every resource and use sh:class

```
:User a sh:Shape ;  
sh:property [  
    sh:predicate schema:worksFor ;  
    sh:class :Company ;  
] .  
  
:Company a sh:Shape ;  
sh:property [  
    sh:predicate schema:name ;  
    sh:datatype xsd:string ;  
] ;  
sh:property [  
    sh:predicate schema:employee ;  
    sh:class :User ;  
] .
```



```
:alice a :User ;  
      schema:worksFor :OneCompany .  
:bob   a :User ;  
      schema:worksFor :OneCompany .  
:carol a :User ;  
      schema:worksFor :Something .
```

:(frowny face)

```
:OneCompany a :Company ;  
      schema:name "One" ;  
      schema:employee :alice, :bob, :carol .
```

Try it: <http://goo.gl/wIVZJR>

Logical Operators

Constraint	Description
and	Conjunction of a list of shapes
or	Disjunction of a list of shapes
not	Negation of a shape



and

Default behavior

```
:User a sh:Shape ;  
  sh:constraint [  
    sh:and (  
      [ sh:property [  
        sh:predicate schema:name;  
        sh:minCount 1;  
      ]  
    ]  
    [ sh:property [  
      sh:predicate schema:affiliation;  
      sh:minCount 1;  
    ]  
  ]  
).
```

=

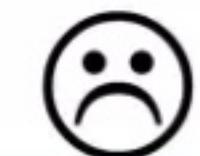
```
:User a sh:Shape ;  
  [ sh:property [  
    sh:predicate schema:name;  
    sh:minCount 1;  
  ]  
  ]  
  [ sh:property [  
    sh:predicate schema:affiliation;  
    sh:minCount 1;  
  ]  
.
```



Or

```
:User a sh:Shape ;  
sh:constraint [  
  sh:or (  
    [ sh:property [  
      sh:predicate foaf:name;  
      sh:minCount 1;  
    ]  
    ]  
    [ sh:property [  
      sh:predicate schema:name;  
      sh:minCount 1;  
    ]  
    ]  
  )  
].
```

```
:alice schema:name "Alice" .  
:bob   foaf:name "Robert" .  
:carol rdfs:label "Carol" .
```



Exclusive-or

There is no exclusive-or in SHACL



not

```
:NotFoaf
a sh:Shape ;
sh:constraint [
sh:not [
a sh:Shape ;
sh:property [
sh:predicate foaf:name ;
sh:minCount 1 ;
] ;
]
]
].
```

```
:alice schema:name "Alice" .
:bob   foaf:name "Robert" .
:carol rdfs:label "Carol" .
```



Value ranges

Constraint	Description
minInclusive	
maxInclusive	
minExclusive	
maxExclusive	

```
:Rating
a sh:Shape ;
sh:property [
  sh:predicate schema:ratingValue ;
  sh:minInclusive 1 ;
  sh:maxInclusive 1 ;
  sh:datatype xsd:integer
] .
```

```
:bad schema:ratingValue 1 .
:average schema:ratingValue 3 .
:veryGood schema:ratingValue 5 .
:zero schema:ratingValue 0 . ☹
```

Try it: <http://goo.gl/qnd66j>



String based constraints

Constraint	Description
minLength	Restricts the minimum string length on value nodes
maxLength	Restricts the maximum string length on value nodes
pattern	Checks if the string value matches a regular expression
stem	Checks if all value nodes are IRIs and the IRI starts with a given string value
uniqueLang	Checks that no pair of nodes use the same language tag

minLength/maxLength

Checks the string representation of the value

This cannot be applied to blank nodes

If minLength = 0, no restriction on string length

```
:User
  a sh:Shape ;
  sh:property [
    sh:predicate schema:name ;
    sh:minLength 4 ;
    sh:maxLength 10 ;
  ] .
```

```
:alice schema:name "Alice" .
:bob schema:name "Bob" .
:carol schema:name :Carol .
:strange schema:name _:strange .
```



Try it: <http://goo.gl/NrJl83>

pattern

Checks if the values matches a regular expression
It can be combined with sh:flags

```
:Product
a sh:Shape ;
sh:property [
    sh:predicate schema:productID ;
    sh:pattern "^\d{3,4}" ;
    sh:flags "i" ;
] .
```

```
:car    schema:productID "P2345" .
:bus    schema:productID "p567" .
:truck   schema:productID "P12" .
:bike   schema:productID "B123" .
```



Try it: <http://goo.gl/BsHpqu>

stem

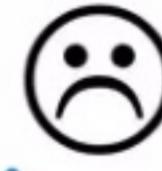
Checks if all value nodes are IRIs and those IRI start with a given string value

```
:W3People
  a sh:Shape ;
  sh:property [
    sh:predicate schema:url ;
    sh:stem "https://www.w3.org/People/" ]
.
```

```
:eric schema:url
      <https://www.w3.org/People/Eric> .

:timbl schema:url
      <https://www.w3.org/People/Berners-Lee> .

:alice schema:url
      <https://www.example.org/bob> .
```



Try it: <http://goo.gl/Krpao8>

uniqueLang

Checks that no pair of nodes use the same language tag

```
:Country
a sh:Shape ;
sh:property [
  sh:predicate schema:name ;
  sh:uniqueLang true
] .
```

```
:spain schema:name "Spain"@en,
       "España"@es .

:france schema:name "France"@en,
        "Francia"@es .

:usa    schema:name "USA"@en,
        "United States"@en.
```



Try it: <http://goo.gl/B1PNcO>

Property pair constraints

Constraint	Description
equals	The sets of values of both properties at a given focus node must be equal
disjoint	The sets of values of both properties at a given focus node must be different
lessThan	The values must be smaller than the values of another property
lessThanOrEquals	The values must be smaller or equal than the values of another property

```
:User a sh:Shape ;  
sh:property [  
    sh:predicat schema:givenName ;  
    sh>equals foaf:firstName  
];  
sh:property [  
    sh:predicat schema:givenName ;  
    sh:disjoint schema:lastName  
].
```

```
:alice schema:givenName "Alice";  
schema:lastName "Cooper";  
foaf:firstName "Alice".  
  
:bob schema:givenName "Bob";  
schema:lastName "Smith" ;  
foaf:firstName "Robert".  
  
:carol schema:givenName "Carol";  
schema:lastName "Carol" ;  
foaf:firstName "Carol".
```



Try it: <http://goo.gl/BFzMoz>

Closed shapes

Constraint	Description
closed	Valid resources must only have values for properties that appear in sh:property
ignoredProperties	Optional list of properties that are also permitted

```
:User a sh:Shape ;  
sh:constraint [  
    sh:closed true ;  
    sh:ignoredProperties ( rdf:type )  
] ;  
sh:property [  
    sh:predicate schema:givenName ;  
];  
sh:property [  
    sh:predicate schema:lastName ;  
] .
```

```
:alice schema:givenName "Alice";  
schema:lastName "Cooper" .  
  
:bob a :Employee ;  
schema:givenName "Bob";  
schema:lastName "Smith" .  
  
:carol schema:givenName "Carol";  
schema:lastName "King" ;  
rdfs:label "Carol" .
```



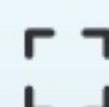
Try it: <http://goo.gl/wcW16o>

Non-validating constraints

Can be useful to annotate shapes or design UI forms

Constraint	Description
name	Provide human-readable labels for a property
description	Provide a description of a property
order	Relative order of the property
group	Group several constraints together

```
:User a sh:Shape ;
  sh:property [
    sh:property schema:url ;
    sh:name "URL";
    sh:description "User URL";
    sh:order 1
  ];
  sh:property [
    sh:property schema:name ;
    sh:name "Name";
    sh:description "User name";
    sh:order 2
  ].
```



Non-validating constraints

```
:User a sh:Shape ;  
  sh:property [ sh:predicate schema:url ;  
    sh:name "URL";  
    sh:group :userDetails  
  ];  
  sh:property [ sh:predicate schema:name ;  
    sh:name "Name"; sh:group :userDetails  
  ];  
  sh:property [ sh:predicate schema:address ;  
    sh:name "Address"; sh:group :location  
  ];  
  sh:property [ sh:predicate schema:country ;  
    sh:name "Country"; sh:group :location  
  ] .
```

```
:userDetails a sh:PropertyGroup ;  
  sh:order 0 ;  
  rdfs:label "User details" .  
  
:location a sh:PropertyGroup ;  
  sh:order 1 ;  
  rdfs:label "Location" .
```

An agent could generate a form like:

User details
URL: _____
Name: _____
Location
Address: _____
Country: _____

Partitions

partition, qualified value shape

TODO



Filters

Filters limit the nodes that are in scope to those that satisfy the filter

Similar to: "if <filter> then ..."

```
:User a sh:Shape ;  
sh:scopeClass schema:Person ;  
sh:filterShape [  
    a sh:Shape ; # Optional triple  
    sh:property [  
        sh:predicate schema:worksFor ;  
        sh:hasValue :OurCompany ;  
    ]  
] ;  
sh:property [  
    sh:predicate schema:url ;  
    sh:stem "http://ourcompany.org/" ;  
].
```

```
:alice a schema:Person ;  
schema:worksFor :OurCompany ;  
schema:url <http://ourcompany.org/alice> .  
  
:bob a schema:Person ;  
schema:worksFor :OurCompany ;  
schema:url <http://othercompany.org/bob> .  
  
:carol a schema:Person ;  
schema:worksFor :OtherCompany ;  
schema:url <http://othercompany.org/carol> .
```



Try it: <http://goo.gl/vadFMk>

SPARQL constraints

Constraints based on SPARQL code.

The query returns validation errors

Constraint	Description
SPARQLConstraint	Type of constraints that will be considered as SPARQL constraints
message	Message in case of error
sparql	SPARQL code to be executed
prefix	Declare reusable prefix

SPARQL constraints

Special variables are used to bind values between SHACL and SPARQL processors

Constraint	Description
\$this	Focus Node
\$shapesGraph	Can be used to query the shapes graph
\$currentShape	Current validated shape

SPARQL constraints

Mappings between result rows and error validation information

Constraint	Description
sh:focusNode	Value of \$this variable
sh:subject	Value of ?subject variable
sh:predicate	Value of ?predicate variable
sh:object	Value of ?object variable
sh:message	Value of ?message variable
sh:sourceConstraint	The constraint that was validated against
sh:sourceShape	The shape that was validated against
sh:severity	sh:ViolationError by default or the value of sh:severity



Extension mechanism

SHACL offers an extension mechanism based on SPARQL
In principle, it should be possible to add other mechanisms

```
<http://www.w3.org/2000/01/rdf-schema#> sh:prefix "rdfs" .  
  
:SpanishLabelsShape a sh:Shape ;  
sh:constraint [  
a sh:SPARQLConstraint ;  
sh:message "Values must be literals with Spanish language tag." ;  
sh:sparql """SELECT $this ($this AS ?subject)  
          (rdfs:label AS ?predicate)  
          (?value AS ?object)  
WHERE { $this rdfs:label ?value .  
      FILTER (!isLiteral(?value) || !langMatches(lang(?value), "es"))""";  
] .
```

Try it: <http://goo.gl/Wf1Lxn>

Some examples

Optional properties with fixed value



Other features

Several features are currently under discussion

- SPARQL scopes and SPARQL functions

- Extension mechanism for other languages

- Recursion

- User-friendly syntax