SHACL

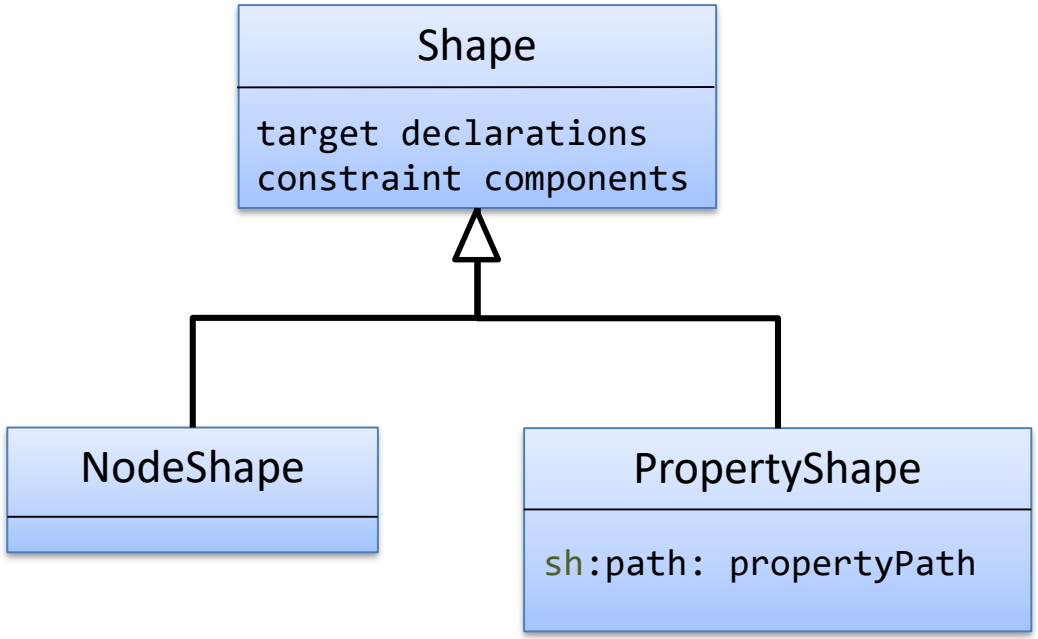
Créditos: Jose Emilio Labra Gayo, Eric Prud'hommeaux
Harold Solbrig, IovkaBoneva

Node and property shapes

2 types of shapes:

NodeShape: constraints about shapes of nodes

PropertyShapes: constraints on property path values of a node



NodeShape

PropertyShape

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:nodeKind sh:IRI ;
   sh:property :hasName,
               :hasEmail .
:hasName sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string .
. . .
```

Node Shapes

Constraints about a focus node

```
:UserShape  a sh:NodeShape ;
      sh:nodeKind      sh:IRI ;
      sh:targetClass   :User .
```

```
:alice a :User .

<http://example.org/bob> a :User .

_:1 a :User .
```

☹

Try it: https://goo.gl/mMPxkM

Property shapes

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

- `sh:property` associates a shape with a property shape
- `sh:path` identifies the path

```
:User a sh:NodeShape ;
   sh:property [
      sh:path     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" .
```

☹

Repeated parameter

Each value of the parameter declares a different constraint

```
:UserShape a          sh:NodeShape;
           sh:class foaf:Person ;
           sh:class schema:Person .
```

```
:alice a schema:Person, foaf:Person .

:bob a schema:Person .
```

☹

SHACL Core constraint components

Type	Constraints
Cardinality	`minCount, maxCount`
Types of values	`class, datatype, nodeKind`
Values	`node, in, hasValue, property`
Range of values	`minInclusive, maxInclusive` `minExclusive, maxExclusive`
String based	`minLength, maxLength, pattern`
Language based	`languageIn, uniqueLang`
Logical constraints	`not, and, or, xone`
Closed shapes	`closed, ignoredProperties`
Property pair constraints	`equals, disjoint, lessThan, lessThanOrEquals`
Non-validating constraints	`name, description, order, group`
Qualified shapes	`qualifiedValueShape, qualifiedValueShapesDisjoint` `qualifiedMinCount, qualifiedMaxCount`

See later

Human friendly messages

Message declares the message that will appear in the validation report in case of violation

```
:UserShape a sh:NodeShape ;
   sh:targetClass :User ;
   sh:property [
    sh:path       schema:name ;
    sh:minCount   1 ;
    sh:message    "Where is the name?"
  ] .
```

```
:bob    a :User ;
        schema:alias "Bob" .
```

☹

```
:report a :ValidationReport ;
 sh:conforms false ;
 sh:result [ a sh:ValidationResult ;
  sh:resultSeverity            sh:Violation ;
  sh:sourceConstraintComponent sh:MinCountConstraintComponent ;
  sh:sourceShape               ... ;
  sh:focusNode                 :bob ;
  sh:resultPath                schema:name ;
   sh:resultMessage            "Where is the name?" ;
].
```

Severities

Declare the level of the violation

3 predefined levels: Violation (default), Warning, Info

```
:UserShape a sh:NodeShape ;
 sh:targetClass :User ;
 sh:property [
    sh:path     schema:name ;
    sh:datatype xsd:string ;
    sh:severity sh:Warning
  ] .
```

```
:bob    a :User ;
        schema:alias "Bob" .
```

```
:report a :ValidationReport ;
 sh:conforms false ;
 sh:result [ a sh:ValidationResult ;
  sh:resultSeverity      sh:Warning ;
  sh:sourceConstraintComponent sh:MinCountConstraintComponent ;
  sh:sourceShape               ... ;
  sh:focusNode                 :bob ;
  sh:resultPath                schema:name ;
  sh:resultMessage             "MinCount Error" ;
].
```

Target declarations

Targets specify nodes that must be validated against the shape

Several types

Value	Description
`targetNode`	Directly point to a node
`targetClass`	All nodes that have a given type
`targetSubjectsOf`	All nodes that are subjects of some predicate
`targetObjectsOf`	All nodes that are objects of some predicate

Target node

Directly declare which nodes must validate the against the shape

```
:UserShape a sh:NodeShape ;
   sh:targetNode :alice, :bob, :carol ;
   sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
   sh:path 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" .
```

Target class

Selects all nodes that have a given class

Looks for rdf:type declarations*

```
:UserShape a sh:NodeShape ;
 sh:targetClass :User ;
 sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
 ] ;
 sh:property [
   sh:path 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 class target

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

The targetClass declaration is implicit

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
     sh:path schema:name ;
     sh:minCount 1;
     sh:maxCount 1;
     sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path 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" .
```

Core constraint components

Type	Constraints
Cardinality	`minCount, maxCount`
Types of values	`datatype, class, nodeKind`
Values	`node, in, hasValue`
Range of values	`minInclusive, maxInclusive` `minExclusive, maxExclusive`
String based	`minLength, maxLength, pattern, stem, uniqueLang`
Logical constraints	`not, and, or, xone`
Closed shapes	`closed, ignoredProperties`
Property pair constraints	`equals, disjoint, lessThan, lessThanOrEquals`
Non-validating constraints	`name, value, defaultValue`
Qualified shapes	`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:NodeShape ;
   sh:property [
    sh:path      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

```
:User a sh:NodeShape ;
   sh:property [
    sh:path     schema:birthDate ;
    sh:datatype xsd:date ;
  ] .
```

```
:alice schema:birthDate "1985-08-20"^^xsd:date .

:bob   schema:birthDate "Unknown"^^xsd:date .

:carol schema:birthDate 1990 .
```

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

Kind of values

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

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
   sh:path     schema:name ;
   sh:nodeKind sh:Literal ;
  ];
  sh:property [
   sh:path     schema:follows ;
   sh:nodeKind sh:BlankNodeOrIRI
  ];
  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:NodeShape, rdfs:Class ;
  sh:property [
   sh:path      schema:affiliation ;
   sh:hasValue :OurCompany ;
  ];
  sh:property [
   sh:path 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
node	All values of a given property must have a given shape Recursion is not allowed in current SHACL

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
   sh:path schema:worksFor ;
   sh:node :Company ;
  ] .

:Company a sh:Shape ;
  sh:property [
   sh:path     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 .
```

☹

Logical Operators

Constraint	Description
`and`	Conjunction of a list of shapes
`or`	Disjunction of a list of shapes
`not`	Negation of a shape
`xone`	Exactly one (similar XOR for 2 arguments)

and

Default behavior

```
:User a sh:NodeShape ;
  sh:and (
   [ sh:property [
       sh:path     schema:name;
       sh:minCount 1;
     ]
   ]
   [ sh:property [
       sh:path     schema:affiliation;
       sh:minCount 1;
     ]
   ]
  ) .
```

≡

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

or

```
:User a sh:NodeShape ;
  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" .
```

☹

not

```
:NotFoaf a sh:NodeShape ;
 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" .
```

☹

Exactly one

```
:UserShape a sh:NodeShape ;
  sh:targetClass :User ;
  sh:xone (
   [ sh:property [
      sh:path      foaf:name;
      sh:minCount 1;
     ]
   ]
   [ sh:property [
      sh:path      schema:name;
      sh:minCount 1;
     ]
   ]
  ) .
```

```
:alice a :User ;                #Passes as :User
       schema:name "Alice" .

:bob   a :User ;                #Passes as :User
       foaf:name   "Robert" .

:carol a :User ;                #Fails as :User
       foaf:name   "Carol";
       schema:name "Carol" .

:dave  a :User ;                #Fails as :User
       rdfs:label  "Dave" .
```

Value ranges

Constraint	Description
minInclusive	<=
maxInclusive	>=
minExclusive	<
maxExclusive	>

```
:Rating a sh:NodeShape ;
 sh:property [
   sh:path          schema:ratingValue ;
   sh:minInclusive 1 ;
   sh:maxInclusive 5 ;
   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

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:NodeShape ;
 sh:property [
   sh:path       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:NodeShape ;
 sh:property [
   sh:path    schema:productID ;
   sh:pattern "^P\\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

Language based constraints

Constraint	Description
`languageIn`	Declares the allowed languages of a literal
`uniqueLang`	Specifies that no pair of nodes can have the same language tag

languageIn

Specifies the allowed language that a literal can have

```
:ProductShape a sh:NodeShape;
  sh:targetClass :Product ;
  sh:property [
    sh:path       rdfs:label ;
    sh:languageIn ("es" "en" "fr")
] .
```

```
:p234 a :Product ;
  rdfs:label "jamón"@es, "ham"@en .

:p235 a :Product ;
  rdfs:label "milk"@en .

:p236 a :Product ;
  rdfs:label "Käse"@de .

:p237 a :Product ;
  rdfs:label "patatas"@es ,
             "kartofeln"@de .
```

☹

☹

uniqueLang

Checks that no pair of nodes use the same language tag

```
:CountryShape a sh:NodeShape ;
  sh:targetClass :Country ;
  sh:property [
    sh:path       skos:prefLabel ;
    sh:uniqueLang true
] .
```

```
:spain  a :Country;
 skos:prefLabel "Spain"@en,
                "España"@es .

:france a :Country;
  skos:prefLabel "France",
                 "France"@en,
                 "Francia"@es .

:italy  a :Country .

:usa    a :Country;
        skos:prefLabel "USA"@en,
                       "United States"@en.
```

☹

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:NodeShape ;
 sh:property [
  sh:path    schema:givenName ;
  sh:equals foaf:firstName
];
 sh:property [
  sh:path       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:NodeShape ;
  sh:closed true ;
  sh:ignoredProperties ( rdf:type ) ;
  sh:property [
    sh:path schema:givenName ;
  ];
  sh:property [
    sh:path 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