TopQuadrant™

SHACL Tutorial: Getting Started

Updated for the SHACL W3C Recommendation and TopBraid 5.3

What is SHACL?

SHACL (Shapes Constraint Language) is a data modeling language that has been developed by a W3C Working Group. As more people work with data coming from a variety of sources, especially for data integration projects, SHACL gives them a way to describe the "shapes" of the data that they're working with so that applications can take better advantage of that data. In addition to describing which properties go with which classes (for example to drive user input forms), SHACL lets you define constraints on data that, when used by your applications, can make it easier to improve the quality of your data with standardized models instead of procedural code.

This tutorial is written for people who would like to get an overview of SHACL. You will learn how to define some shapes and constraints on those shapes using a few of the properties defined by the SHACL specification. If you do a quick skim through the specification's table of contents, you will get an idea of the broad vocabulary that SHACL offers to let you describe your data and constraints. This tutorial uses TopBraid Composer. If you want to see how to perform the same tasks using web browser and TopBraid EDG, switch to the SHACL in EDG tutorial.

You can use SHACL to define classes together with constraints on their properties. SHACL comes with several built-in types of constraints such as cardinality (minCount/maxCount), value type and allowed values, but it is also possible to define more complex kinds of constraints for almost arbitrary validation conditions. Using these constraints you can communicate the intended format of your data to other people and tools. SHACL validation tools can verify whether your data fulfills the constraints described by your data model, similar to how XML Schema or JSON Schema are being used.

While languages like XML Schema are limited to tree structures, SHACL is based on RDF and supports the validation of graph-based and object-oriented data. Since SHACL is based on RDF, it may also serve as a schema language for JSON via its JSON-LD dialect. The examples in this Tutorial are using the RDF Turtle notation. We also show screenshots of **TopBraid**

Composer, for those who would like to follow along with a graphical editing tool.

Getting Started

A SHACL file is an RDF graph, which means that it comprises of RDF triples, which each consist of a subject, a predicate and an object. The nodes in a SHACL graph can have a URI, which means that any class, shape or constraint can have a global identifier that can be referenced from other SHACL graphs and shared on the Web as Linked Data.

Let's start by creating a new SHACL file with a classical data model about Persons and family relationships. This following discussion could also be applied to an existing RDFS or OWL ontology – SHACL can be incrementally applied to existing data models.

Assuming you have TopBraid Composer 5.3 or higher installed, create a new project (**File > New > Project…** named **example.org**) and then use **File > New > SHACL Shapes File** as shown:

TBC-Create-Shapes-File

The Turtle source code so far is merely defining a few namespaces and an **owl:Ontology** element to capture information about the graph/file itself. It also links to the DASH Data Shapes Library, which extends the SHACL Core Vocabulary, via **owl:imports**, so that SHACL tools know which library to use:

```
# baseURI: http://example.org/family
# imports: http://datashapes.org/dash
# prefix: family

@prefix dash: <http://datashapes.org/dash#> .
@prefix family: <http://example.org/family#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.org/family>
  rdf:type owl:Ontology ;
  owl:imports <http://datashapes.org/dash> ;
.
```

Back in TopBraid Composer, we can **close the Properties panel** to gain some screen real estate. Also, switch to the **Shapes View** on the left hand side. It should now look (roughly) as follows:

SHACL Tutorial: TBC New File

Classes, Shapes and Property Constraints

In RDF data models, a **Class** is a collection of instances with similar properties. Similar to object-oriented systems, SHACL class definitions can define attributes and relationships. **Shapes** are similar to classes: they also

define shared characteristics of a group of nodes and properties that belong together. The main difference however is that shapes are not directly instantiated, but serve as a collector of constraints for validation purposes. Another difference is that shapes can apply to literal nodes, too. Shapes are also useful when we cannot rely on type relationships in the data to determine which constraints apply.

Let's look at an example. We want to create a class **family:Person** and specify that each Person must have exactly one last name and one or more first names. In TopBraid, we click **Create Shape Class...** and fill in the dialog as follows:

TBC-Create-Person-Class

Note that this new class has two types. In addition to being a normal **rdfs:Class**, it is also declared to be a **sh:NodeShape**. This is a convenient pattern to attach property constraints to a class, although there are also other design patterns to relate classes with the shapes of their instances. When you look at the form of the new class, you can see two types of constraints:

- **sh:property**: Links a shape/class with the associated property declarations, for example the first name of a Person.
- **sh:sparql**: Arbitrary constraints expressed in SPARQL.

In addition to those, a shape may declare constraints that are directly about the target nodes of the shape itself. Examples of this include properties such as sh:closed, sh:and, sh:or and sh:not.

Datatype Property Constraints

In order to state that every instance of **family:Person** must have exactly one last name, we can click the context menu behind **sh:property** and select **Create Property constraint...**, and fill in the dialog as follows:

SHACL Tutorial TBC Create Property Constraint lastName

The resulting source code now contains a class definition with one property constraint. Note that the property constraint is represented by an anonymous "blank" node, as shown by the [...] angular brackets. If you prefer to use URIs for these nodes instead, use **Convert to URI node** from the context menu behind the property constraint.

```
family:Person
    a rdfs:Class, sh:NodeShape ;
    rdfs:label "Person" ;
    rdfs:subClassOf rdfs:Resource ;
    sh:property [
        sh:path family:lastName ;
        sh:name "last name" ;
```

```
        sh:description "A Person's last name (aka family name)." ;
        sh:datatype xsd:string ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
    ] ;
.
```

The shape above has introduced the use of some properties defined by the SHACL specification:

- **sh:path**: Points at the URI of the property that is being restricted. Alternative, it may point at a path expression, which would allow you to constrain values that may be several "hops" away from the starting point.
- **sh:minCount** and **sh:maxCount**: Can specify the minimum and maximum number of allowed values for the property, aka the cardinality. If left unspecified, the cardinality is zero-to-many.
- **sh:datatype**: Specifies that all values must be literals with the given type. Common values are **xsd:string**, **xsd:boolean**, **xsd:integer**, **xsd:decimal**, **xsd:date** and **xsd:dateTime**. Use **rdf:langString** for strings with a language tag, and **rdf:HTML** for strings including HTML markup.

In addition, property constraints may define human-readable labels via **sh:name** and descriptions of the property via **sh:description**, both potentially in multiple languages.

We can repeat the same mechanism to declare **family:firstName** – the only difference is that we define the cardinality to be **One or more [1..*]**. Once we have done this, the resulting class definition looks as follows in Composer:

TBC-Person-Form1

In the screenshot above, the details of the lastName property are expanded. This can be achieved by moving the mouse over the property icon, and clicking the + button.

Let's now use this class and shape definition for some validation task. We can use TopBraid to create an instance of the class **family:Person**, e.g. on the **Instances** panel. An example instance could look like the following:

SHACL-Tutorial-TBC-Instance-with-violations

The source code of the Person instance above is the following:

```
family:JohnDoe
    rdf:type family:Person ;
    family:lastName "Doe" ;
.
```

This instance violates the **sh:minCount** constraint at **family:firstName**. In TopBraid, you can activate the "exclamation mark" **Display constraint violation warnings** button in the tool bar to highlight incorrect or missing

values on the form. You can also display a clickable list of all constraint violations in the corresponding **SHACL Validation** panel at the bottom of the screenshot above. Adding a first name will make the errors disappear.

Relationship Property Constraints

SHACL property constraints may also define relationships between resources. For example, a Person may have a mother and a father, which are again instances of Person. Let's create the **family:mother** relationship and leave the father as an exercise. In TopBraid Composer, navigate back to the Person class and pick **Create Property constraint...** in the drop down behind **sh:property**, then fill in the dialog as follows:

[SHACL-Tutorial-TBC-Create-Property-Constraint-mother](#)

The resulting source code now has an additional constraint on the mother property:

```
family:Person
    ...
    sh:property [
        sh:path family:mother ;
        sh:name "mother" ;
        sh:description "A Person's mother." ;
        sh:maxCount 1 ;
        sh:class family:Person ;
        sh:nodeKind sh:IRI ;
    ] ;
.
```

This introduces two new SHACL system properties:

- **sh:nodeKind**: Can be used to specify that all values of a property must be Literals, IRIs or Blank nodes. Here, we have selected **sh:IRI** to make sure that all values are resources with an IRI (that is, a URI that allows an internationalized character set), and not an anonymous node that cannot be linked to from other graphs.
- **sh:class**: Specifies the type (rdf:type) of all values. In other words, all values of mother must be instances of family:Person, or a subclass thereof.

In order to make this example more interesting, let's create another class **family:Gender** with two instances **family:male** and **family:female**.

```
family:Gender
    rdf:type rdfs:Class ;
    rdfs:label "Gender" ;
    rdfs:subClassOf rdfs:Resource ;
.
family:female
    rdf:type family:Gender ;
    rdfs:label "female" ;
.
family:male
    rdf:type family:Gender ;
    rdfs:label "male" ;
.
```

Next, create a new property **family:gender** that points from the Person class to the Gender class:

TBC-Person-Form-with-gender

Now we are ready to explore pure Shapes that are not (also) classes. Let's assume we want to specify that all values of **family:mother** must be female. To do this, we create a new **sh:NodeShape** called **family:FemaleShape** with a constraint on the **family:gender** property. In TopBraid Composer's Shapes view, click on **Create Shape...**, leading to the following dialog:

TBC-Create-FemaleShape

This takes us to the form for **family:FemaleShape**, where we can add a property constraint on **family:gender**. This time we select **Add empty row** in the context menu behind **sh:property** on the form. Then we enter **family:gender** under **sh:path**, and select **family:female** using **Add existing value...** next to **sh:hasValue**, leading to the following definition:

SHACL-Tutorial-TBC-FemaleShape

The source code of the complete definition of this shape looks as follows:

```
family:FemaleShape
    rdf:type sh:NodeShape ;
    rdfs:label "Female shape" ;
    sh:property [
        sh:hasValue family:female ;
        sh:path family:gender ;
    ] ;
.
```

The **sh:hasValue** system property represents a constraint that one of the values of the given predicate must be **family:female**. (As a side effect this also implies that at least one value must exist).
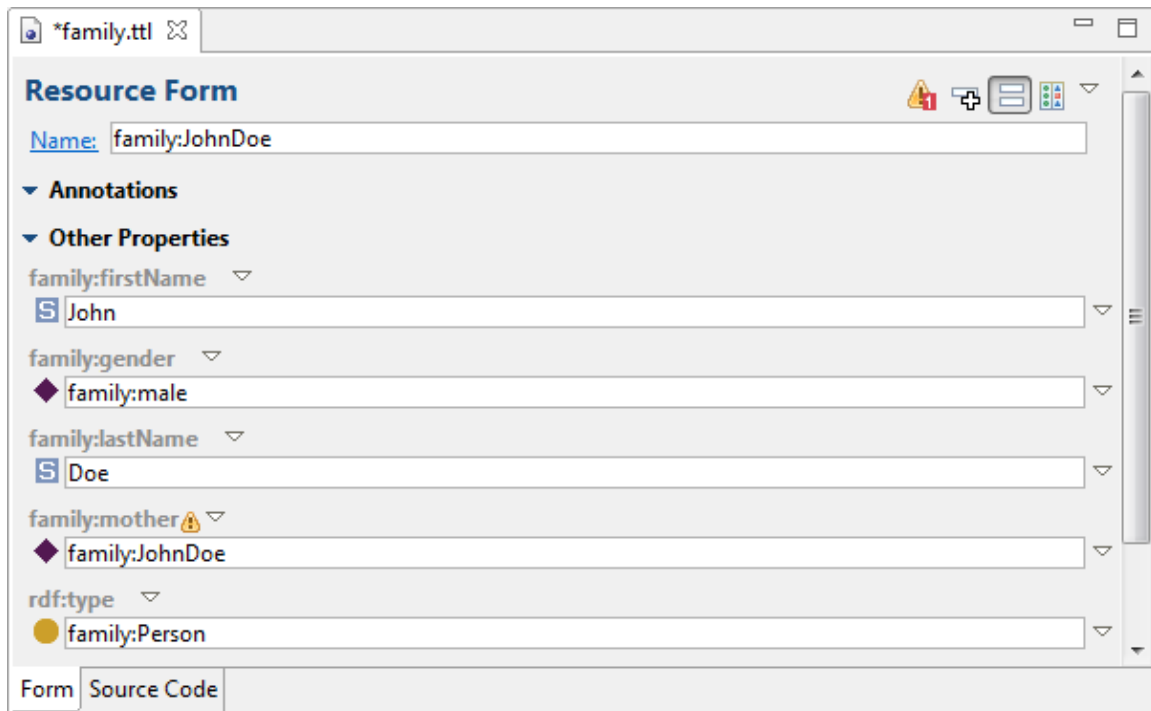
We can now go back to the Person class and add a **sh:node** constraint to the **family:mother** property. To do that in TopBraid Composer, open the "nested" form for the property constraint using a right-click so that all relevant properties including **sh:node** appear. (Alternatively, edit the Source Code tab directly). Finally, add **family:FemaleShape** to the values of **sh:node**, leading to the following situation:

SHACL-Tutorial-TBC-shape-constraint

The meaning of **sh:node** is that all values of the property must fulfill the constraints defined by the given **sh:NodeShape**. This is very different from **sh:class** in that it actually "walks" into the values of **family:mother** to (recursively) check that they all have the female gender. Such shapes can become quite complex and may include further nested shapes. SHACL also

has a syntax to express logical combinations such as AND, OR and NOT for complex scenarios.

Anyway, we can now verify that our constraint actually works by creating a simple constraint violation. We can make John Doe his own mother. This should trigger a violation because the mother must be female:

Further Topics

There would be much more to say about SHACL in this tutorial. Here is a list of some notable features for future editions and further reading:

SHACL Tutorial: SPARQL-based Constraints provide the ultimate expressive power to define almost arbitrary conditional checks.

Targets (sh:target) are a general mechanism for SHACL to select the nodes to which a given Shape applies to. A common pattern is to associate a Shape with a class (e.g. via **sh:targetClass**) but this mechanism can also be used to associate a shape with all subjects that have a given property, thus defining **"global" property constraints**.

User-defined constraint components are a macro facility in SHACL, allowing advanced users to prepare new high-level modeling elements to describe constraints. The SHACL Core constraints such as **sh:minCount** are all defined using its own extension mechanism.

Functions are another macro facility to define new SPARQL functions that encapsulate another SPARQL query similar to stored procedures. This can

lead to dramatically more maintainable queries based on reusable building blocks.

This concludes our crash course into SHACL for now. Stay tuned for future updates.

Privacy Statement   Legal   Terms of Use

Copyright 2020 TopQuadrant, Inc. All Rights Reserved.