Shape Expressions Language 2.next

The Shape Expressions (ShEx) language describes RDF nodes and graph structures. A node constraint describes an RDF node (IRI, blank node or literal) and a shape describes the triples involving nodes in an RDF graph. These descriptions identify predicates and their associated cardinalities and datatypes. ShEx shapes can be used to communicate data structures associated with some process or interface, generate or validate data, or drive user interfaces.

This document defines the ShEx language. See the [Shape Expressions Primer](../primer/index.html) for a non-normative description of ShEx.

RDF, Schema, Shape Expressions, Structure Definition, Structural Validation

This is an editor's draft of the Shape Expressions specification. ShEx 2.x differs significantly from the W3C ShEx Submission. The [July 2017 publication](http://shex.io/shex-semantics-20170713/) included a [definition of validation](#validation) which implied infinite recursion. This version explicitly includes recursion checks. No tests changed as a result of this and no implementations or applications are known to have been affected.

If you wish to make comments regarding this document, please raise them as GitHub issues. There are separate interfaces for [specification](https://github.com/shexSpec/spec/issues), [language](https://github.com/shexSpec/shex/issues) and [test](https://github.com/shexSpec/shexTest/issues) issues. Only send comments to [public-shex@w3.org](mailto:public-shex@w3.org) ([subscribe](mailto:public-shex-request@w3.org?subject=subscribe), [archives](https://lists.w3.org/Archives/Public/public-shex/)) if you are unable to raise issues on GitHub. All comments are welcome.

## Conformance

Conformance criteria are relevant to authors and authoring tool implementers. As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

All ShEx documents MUST conform to the [Schema Requirements](#schema-requirements). Additional constraints for the specific types of ShEx documents (ShExC, ShExJ, and ShExR) follow:

* A ShExC document is a UTF-8 document which conforms to the grammar described in resulting in a valid ShExJ document.
* A ShExJ document is a valid JSON-LD document [[!JSON-LD]], and conforms to the ShExJ syntax, as described in .
* JSON documents can be interpreted as ShExJ by following the normative statements in Section 4.8 Interpreting JSON as JSON-LD in [[!JSON-LD]].
* A ShExR RDF document complies with this specification if it conforms to the schema in [ShExR.shex](#shexr).

## Introduction

The Shape Expressions (ShEx) language provides a structural schema for RDF data. This can be used to document APIs or datasets, aid in development of API-conformant messages, minimize defensive programming, guide user interfaces, or anything else that involves a machine-readable description of data organization and typing requirements.

ShEx describes RDF graph [[RDF11-CONCEPTS]] structures as sets of potentially connected Shapes. These constrain the triples involving nodes in an RDF graph. Node Constraints constrain RDF nodes by constraining their node kind (IRI, blank node or Literal), enumerating permissible values in value sets, specifying their datatype, and constraining value ranges of Literals. Additionally, they constrain lexical forms of Literals, IRIs and labeled blank nodes. Shape Expressions schemas share blank nodes with the constrained RDF graphs in the same way that graphs in RDF datasets [[!rdf11-concepts]] share blank nodes.

ShEx can be represented in JSON structures (ShExJ) or a compact syntax (ShExC). The compact syntax is intended for human consumption; the JSON structure for machine processing. This document defines ShEx in terms of ShExJ and includes a [section on the ShEx Compact Syntax (ShEx)](#shexc).

## Definitions, Acronyms, and Abbreviations

### Definitions

Shape expressions are defined using terms from RDF semantics [[!rdf11-mt]]:

* Node: one of IRI, blank node, Literal
* Graph: a set of Triples of (subject, predicate, object)

The following functions access the elements of an RDF graph G containing a node n:

* arcsOut(G, n) is the set of triples in a graph G with subject n.
* predicatesOut(G, n) is the set of predicates in arcsOut(G, n).
* arcsIn(G, n) is the set of triples in a graph G with object n.
* predicatesIn(G, n) is the set of predicates in arcsIn(G, n).
* neigh(G, n) is the neighbourhood of the node n in the graph G.  
  neigh(G, n) = arcsOut(G, n) ∪ arcsIn(G, n).
* predicates(G, n) is the set of predicates in neigh(G, n).  
  predicates(G, n) = predicatesOut(G, n) ∪ predicatesIn(G, n).
* def(Sch, label) is the decl.shapeExpr where decl.label = label. Sch must have exactly one def(Sch, label).

Consider the RDF graph G represented in Turtle:

PREFIX ex: http://schema.example/#  
PREFIX inst: http://inst.example/#  
PREFIX foaf: http://xmlns.com/foaf/  
PREFIX xsd: http://www.w3.org/2001/XMLSchema#  
  
inst:Issue1  
 ex:state ex:unassigned ;  
 ex:reportedBy \_:User2 .  
  
\_:User2  
 foaf:name "Bob Smith" ;  
 foaf:mbox <mailto:bob@example.org> .

There are two arcs out of \_:User2; arcsOut(G, \_:User2):

\_:User2 foaf:name "Bob Smith" .  
 \_:User2 foaf:mbox <mailto:bob@example.org> .

There is one arc into \_:User2; arcsIn(G, \_:User2):

inst:Issue1 ex:reportedBy \_:User2 .

There are three arcs in the neighbourhood of \_:User2 set, neigh(G, \_:User2):

\_:User2 foaf:name "Bob Smith" .  
 \_:User2 foaf:mbox <mailto:bob@example.org> .  
 inst:Issue1 ex:reportedBy \_:User2 .