ShExStatements: Simplifying Shape Expressions for Wikidata

John Samuel
john.samuel@cpe.fr
CPE Lyon, LIRIS UMR-CNRS 5205, Université de Lyon
Villeurbanne, France

ABSTRACT

Wikidata recently supported entity schemas based on shape expressions (ShEx). They play an important role in the validation of items belonging to a multitude of domains on Wikidata. However, the number of entity schemas created by the contributors is relatively low compared to the number of WikiProjects. The past couple of years have seen attempts at simplifying the shape expressions and building tools for creating them. In this article, ShExStatements is presented with the goal of simplifying writing the shape expressions for Wikidata.

CCS CONCEPTS

• **Software and its engineering** → **Software libraries and repositories**; • **Information systems** → **Web data description languages**; Wikis.

KEYWORDS

Wikidata, Shape Expressions, Data validation

ACM Reference Format:
John Samuel. 2021. ShExStatements: Simplifying Shape Expressions for Wikidata. In *Companion Proceedings of the Web Conference 2021 (WWW '21 Companion), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3442442.3452349

1 INTRODUCTION

Entity schemas based on ShEx (Shape Expressions) [10, 13, 15] were recently introduced on Wikidata [17]. One of the main advantages of Shape Expressions is that they can be used for RDF validation [7, 14, 16]. Several tools and scripts currently exist that can be used to visualize and validate a subset of data on Wikidata using ShEx. One such tool is shex.js [4], which let the Wikidata contributors easily check entities against any particular schema. A SPARQL query is used to select a subset of relevant data from Wikidata and the validation is run on this prefetched data. Thus users can both test and explore the current state of the data related to the SPARQL query. They may propose new modifications to the entity schema or even correct the data items.

In the case of Wikidata, WikiProjects are used to identify and discuss relevant properties for the items to a particular domain. For example, WikiProject Informatics[1] identifies properties for software, hardware, programming languages, file sytems, algorithms, etc. The number of Wikiprojects is an interesting indicator for measuring the use of entity schemas since WikiProjects are managed by dedicated contributors interested in a particular domain. At the time of writing, there are only less than 300 shape expressions[2] on Wikidata. This number is quite low compared to the number of WikiProjects on Wikipedia [9] and Wikidata [6].

Therefore any tool for shape expressions must take into consideration such WikiProjects and propose ways to integrate the information present in these tools to build shape expressions. One possible approach is to propose a smaller subset of shape expressions that can be used to build simple shape expressions in a manner that closely resembles some of the existing templates[3]. These simple expressions can take into account the WikiProjects for validating whether the items belonging to a given domain have all the necessary statements. Considering the multilingual nature, another important aspect is to let the communities describe relevant domains in their local languages. ShExStatements[4] [12] was developed to answer these requirements. It was developed in a manner similar to the QuickStatements[5] and OpenRefine[6] [3] to ensure a simpler interface using tabular formats or CSV files.

In this article ShExStatements [12] is presented, explaining how a tabular format or a CSV file format was developed for simplifying writing shape expressions, especially for the new comers. In section 2, state of the art is presented. Taking an example, the grammar of ShExStatements is described in section 3. Section 4 presents the development and use of ShExStatements. Section 6 concludes the article.

2 RELATED WORKS

Several WikiProjects[7] are currently available on Wikidata related to open government data, culture, history, sports, birds, agriculture, tourism, etc. Some of these WikiProjects take into consideration the infoboxes of Wikipedia[8] belonging to different languages to identify the different properties used to describe the objects belonging to a certain class. These infobox properties are then mapped to appropriate Wikidata properties. WikiProjects, therefore, play an important role in identifying the key Wikidata properties. However, WikiProjects alone cannot be used to automatically validate the existing Wikidata items.

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.
WWW '21 Companion, April 19–23, 2021, Ljubljana, Slovenia
© 2021 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY 4.0 License.
ACM ISBN 978-1-4503-8313-4/21/04.
https://doi.org/10.1145/3442442.3452349

[1]https://www.wikidata.org/wiki/Wikidata:WikiProject_Informatics
[2]https://www.wikidata.org/wiki/User:HakanIST/EntitySchemaList
[3]https://www.wikidata.org/wiki/Template:List_of_properties/Row
[4]https://shexstatements.toolforge.org/
[5]https://quickstatements.toolforge.org/
[6]https://openrefine.org/
[7]https://www.wikidata.org/wiki/Wikidata:WikiProjects

Though Wikidata supports property constraints[8], their usage is limited to specifying how properties can be used. Wikidata items use multiple properties and schemas are needed to describe and validate the items belonging to different classes. This is very important in multilingual and multi-domain context. Therefore, validation of RDF [7, 14, 16] is important to ensure data present in a semantic knowledge base is following the proposed ontology or schema.

Several tools have been proposed that take into consideration the expressivity [15] of shape expressions. These tools can be classified in the way shape expressions can be created. The first approach is to automatically generate schema expressions from existing RDF data. Designer [1], *Wikidata Shape Expressions Inference*[9], and sheXer[10] are some examples. Visual interfaces have also been suggested to understand, modify and create new shape expressions. YASHE[11] and ShExAuthor[12] are examples of some visual tools for creating Shape Expressions. Finally, there are approaches that propose a smaller subset of the ShEx language. Shex-Lite[13] [2], ShExML [5], and ShExStatements belong to this category. ShExML is a language developed to integrate multiple heterogeneous data sources. Shex-Lite is meant to be an independent language, maintaining compatibility with ShEx, and can be used to generate object models in object-oriented programming languages. ShExStatements, on the other hand, is a language developed to generate ShEx from CSV files and tabular formats.

3 SHEXSTATEMENTS

To explain the grammar of ShExStatements, an example is given below in Figure 6. It describes the ShExStatements of a human language on Wikidata.

```
wd,<http://www.wikidata.org/entity/>,,,
wdt,<http://www.wikidata.org/prop/direct/>,,,

@language,wdt:P31,wd:Q34770,,# instance of a language
@language,wdt:P1705,LITERAL,,# native name
@language,wdt:P17,.,+,# spoken in country
@language,wdt:P2989,.,+,# grammatical cases
@language,wdt:P282,.,+,# writing system
@language,wdt:P1098,.,+,# speakers
@language,wdt:P1999,.,*,# UNESCO language status
@language,wdt:P2341,.,+,# indigenous to
```

Figure 1: ShExStatements of a human language on Wikidata

There are two parts, separated by a blank line. This is also shown in Figure 2. The first part consists of prefixes, i.e., the namespaces that are going to be used in the second part. The prefixes in this example include *wd* and *wdt*.

In the second part, there are eight statements. Each statement starts with a node [15], i.e., a string starting with *@*. In this example, *@language* is a node.

[8]https://www.wikidata.org/wiki/Help:Property_constraints_portal
[9]https://wd-shex-infer.toolforge.org/
[10]http://shexer.weso.es/
[11]https://github.com/weso/YASHE
[12]https://github.com/weso/shex-author
[13]https://github.com/weso/shex-lite

The second part has a simple syntax, with 5 columns, with the values separated by a separator (,).

(1) Node name
(2) Property
(3) Allowed values
(4) Cardinality (optional)
(5) Comments (optional)

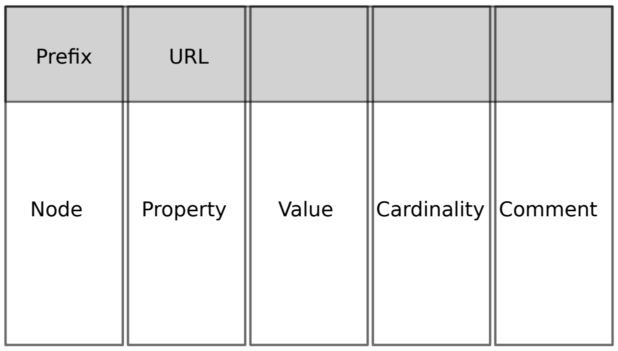


Figure 2: ShExStatements example with its two parts. First part is used for specifying prefixes and the second part for statements

If these five columns are present in the CSV file, column 1 is used for specifying the node name, column 2 for specifying the property value, column 3 for possible values, column 4 for cardinality (+,), and column 5 for comments. Comments start with #. Columns 1, 2, 3 are mandatory. Column 3 can be a special value like . (period to say 'any' value). Columns 3,4 and 5 are empty for prefixes.

Consider the first statement in the second part. It states that a language must be an instance of (*wdt:P31*) a language (*wd:Q34770*). The fourth value, cardinality is intenionally left blank. The fifth value starts with a # indicating a comment.

Cardinality can be any one of the following values

(1) * : zero or more values
(2) ? : zero or one
(3) + : one or more values
(4) m : m number of values
(5) m,n : any number of values between m and n (including m and n).

Take the fifth statement that states that a language can have one or more writing systems, hence the use of + in the fourth column.

But the third column can also be another node. A ShExStatements file can also use delimiters like vertical bar (|) or semicolons (;). The following example in Figure 3 shows these two cases.

This example is a ShExStatements of a TV series. The first statement describes that a TV series is an instance of *wd:Q5398426* (television series). The second statement states that a TV series has zero or more genres *wdt:P136*. However, to describe a genre, we need additional statements. The third statement describes a genre to be

```
@tvseries|wdt:P31|wd:Q5398426||# instance of a tvseries
@tvseries|wdt:P136|@genre|*|# genre
@genre|wdt:P31|wd:Q201658,wd:Q15961987|#instance of genre
```

Figure 3: ShExStatements of a TV series on Wikidata

Table 1: Significance of different terms

No.	Name	Symbol(s)
1.	SEP	| ;
2.	COLON	:
3.	CARET	∧
4.	STAR	∗
5.	PLUS	+
6.	QUESTIONMARK	?
7.	PERIOD	.

an instance of *wd:Q201658* (film genre) or *wd:Q15961987* (television genre). This statement is interesting since it demonstrates the use of different separators. The above example uses vertical bar (|) for separating the columns. The multiple possible values in the third column are separated by comma (,).

Now, the grammar of ShExStatements can be formalized.

A simplified version of the grammar of ShExStatements is given below. For the complete grammar (for example, optional comments, shape constraints, import statements, etc.), the readers can take a look at *shexstatementsparser.py* in [12].

ShExStatements consists of one or more statements, often preceded by prefix statements. There may exist blank lines (*NEWLINE*) between the statements.

Listing 1: ShExStatements: statements

```
statements : NEWLINE
           | statement
           | statement statements
           | prefixes statement statements
```

A statement may be one of the following use cases:

(1) a node with a property and an associated property value. For example, a language has a property value of *wd:Q34770* for the property *wdt:P31*.
(2) a node with a property and multiple associated values separated by a comma. For example, a genre can have any one value from *wd:Q201658, wd:Q15961987.* for the property *wdt:P31*.
(3) a node with a property and an associated property value, along with the cardinality. For example, a language has a property value of . (period) with the cardinality + for the property *wdt:P282* (writing system).

These are detailed below in the grammar. Table 1 can be used as a reference for understanding the terms in upper case.

Listing 2: ShExStatements: statement

```
statement :
  nodeproperty propertyvalue SEP comment
 | nodeproperty delimseparatedlist
          SEP comment
 | nodeproperty propertyvalue SEP
          cardinality SEP comment
```

As described above, in this article, we have given a grammar that shows a statement must have a comment. However, a comment can be omitted.

A *nodeproperty* is a combination of node and property, separated by SEP(|).

Listing 3: ShExStatements: node property

```
nodeproperty : node SEP prop SEP
```

There can be one or more prefixes.

Listing 4: ShExStatements: prefixes

```
prefixes : prefix
     | prefix prefixes
```

A prefix consists of two string separated by SEP.

Listing 5: ShExStatements: prefix

```
prefix : STRING SEP STRING
```

A *propertyvalue* in the third column may be a value, a node, a type, or a special term (e.g., LITERAL above).

Listing 6: ShExStatements: property value

```
propertyvalue : value
              | node
              | type
              | specialterm
```

A *node* consists of a word starting with @ (*NODENAME*), possibly separated by a colon from another word.

Listing 7: ShExStatements: node

```
node : NODENAME
  | NODENAME COLON STRING
```

Special terms include period(.) or types like LITERAL, IRI, BNode, etc.

Listing 8: ShExStatements: special term

```
specialterm : PERIOD
            | NODEKIND
```

As described above, cardinality values include *, ? etc.

Listing 9: ShExStatements: cardinality

```
cardinality : PLUS
            | STAR
            | QUESTIONMARK
            | NUMBER
            | NUMBER COMMA
            | NUMBER COMMA NUMBER
```

To specify types other than LITERAL, we need a special case to distinguish values from types.

Take, for example, in the example given below, we want to specify that a painting must have creation date of type xsd:string. Unlike values, this is a special case. Here we do not know any possible value, but we know the type of those values.

```
@painting,wdt:P571,@@xsd:dateTime,#date of creation
```

Figure 4: ShExStatements of a painting on Wikidata

A *type* consists of a word starting with @@ (*TYPESTRING*), possibly separated by a colon from another word.

Listing 10: ShExStatements: type

```
type  :  TYPESTRING
      |  TYPESTRING  COLON  STRING
```

A value is a non-whitespace string which has no characters like @ in the beginning.

Listing 11: ShExStatements: value

```
value  :  STRING
```

A comment is a string starting with #.

Listing 12: ShExStatements: comment

```
comment  :  COMMENT
```

A *prop* is just a *value* or *value* followed by ∧. This is interesting to specify cases, where we wish to specify that the statement with the given property must not hold.

Listing 13: ShExStatements: property

```
prop  :  value
     |  CARET  value
```

A *commaseparatedvaluelist* is a list of values separated by a comma.

Listing 14: ShExStatements: list of comma separated values

```
commaseparatedvaluelist  :
     value  COMMA  value
    |  value  COMMA  commaseparatedvaluelist
```

A *commaseparatedtypelist* is a list of types separated by a comma.

Listing 15: ShExStatements: list of comma separated types

```
commaseparatedtypelist  :  type  COMMA  type
    |  type  COMMA  commaseparatedtypelist
```

A *delimseparatedlist* is a list of types separated by a comma or a list of values separated by a comma.

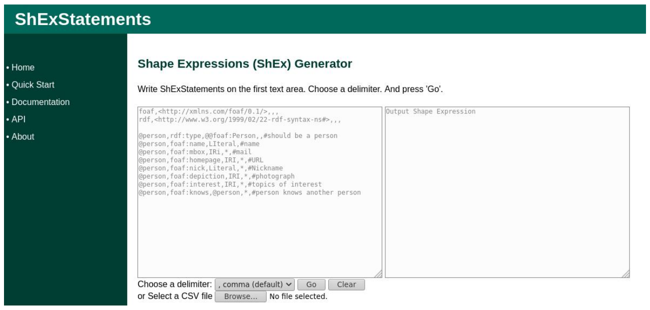
Listing 16: ShExStatements: list of comma separated types or values

```
delimseparatedlist  :
      commaseparatedtypelist
    |  commaseparatedvaluelist
```

4 DEVELOPMENT

ShExStatements is developed in Python and has multiple interfaces. It can be executed from the command line. There is also a web interface as shown in Figure 5 and an API that allows users to generate shape expressions from CSV files.

It uses the library *ply*[14] for writing the grammar as described above and the parser for parsing CSV files or input. The web interface is built using *Flask*[15] and *pyshex*[16] is used to generate ShExj[17] from ShExStatements.



Write ShExStatements on the first text area. Choose a delimiter. And press 'Go'.

```
foaf,<http://xmlns.com/foaf/0.1/>,,,
rdf,<http://www.w3.org/1999/02/22-rdf-syntax-ns#>,,,

@person,rdf:type,@@foaf:Person,,#should be a person
@person,foaf:name,LIteral,#name
@person,foaf:mbox,IRi,*,#mail
@person,foaf:homepage,IRI,*,#URL
@person,foaf:nick,Literal,*,#Nickname
@person,foaf:depiction,IRI,*,#photograph
@person,foaf:interest,IRI,*,#topics of interest
@person,foaf:knows,@person,*,#person knows another person
```

Figure 5: Web interface for ShExStatements

ShEx generation

ShExStatements is also available on Python package index[18] and therefore can be installed using *pip*. Once ShExStatements is installed, run the following command with the above example written in a file (for example, *language.csv*). This file contains an example description of a language on Wikidata and uses comma as a delimiter to separate the values.

```
$ ./shexstatements.sh language.csv
```

ShExStatements will generate the following Shape Expression (ShEx). It is also possible to use *shexstatements* in Python programs. The method *generate_shex_from_csv* takes as input a CSV file containing shexstatements and a delimiter. In this example, we use "," as a delimiter.

Listing 17: ShExStatements: Using Python library

```
from shexstatements.shexfromcsv import CSV

shex = CSV.generate_shex_from_csv(
           "language.csv",
           delim=",")
print(shex)
```

ShExStatements has also a public API that can be easily accessed both on a local installation as well as on the public interface. It has one operation that takes as input a JSON array with two elements as given below:

- delimiter

[14]https://pypi.org/project/ply/
[15]https://pypi.org/project/Flask/
[16]https://pypi.org/project/PyShEx/
[17]https://shexspec.github.io/primer/ShExJ
[18]https://pypi.org/project/shexstatements/

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
start = @<language>
<language> {
  wdt:P31 [ wd:Q34770  ] ;# instance of a language
  wdt:P1705 LITERAL ;# native name
  wdt:P17 .+ ;# spoken in country
  wdt:P2989 .+ ;# grammatical cases
  wdt:P282 .+ ;# writing system
  wdt:P1098 .+ ;# speakers
  wdt:P1999 .* ;# UNESCO language status
  wdt:P2341 .+ ;# indigenous to
}
```

Figure 6: Shape expression of a human language on Wikidata

- CSV (every line should be terminated by newline character)

Following is the way to call the ShExStatements API on a local machine:

```
$ curl -s http://127.0.0.1:5000/ -X POST -H \
    "Accept: application/json" \
    --data  @examples/api/tvseries.json \
    |sed 's/\\n/\n/g'
```

Figure 7: ShExStatements API on a local machine

A small excerpt of *examples/api/tvseries.json* is given below:

```
[
"|",
"wdt|<http://www.wikidata.org/prop/direct/>|||\n
\n
@tvseries|wdt:P136|@genre|*|# genre\n"
]
```

Figure 8: JSON excerpt of TV series for API calls

It returns a JSON array with one element containing the ShEx (shape expression).

5 RESULTS

ShExStatements is also available on Toolforge[19] along with a detailed documentation[20]. A number of shape expressions were created during COVID-19 Biohackathon April 5-11 2020[21] using ShExStatements. For example, pandemic (EntitySchema:E184[22]), hospital, preprint, lockdown, etc. The primary goal was to identify the key properties for these entities, which could later be improved and extended. During this hackathon, the possibility of using such simple shape expressions for selecting data from Wikidata was also discussed.

[19]https://shexstatements.toolforge.org/

[20]https://shexstatements.readthedocs.io/en/latest/

[21]https://github.com/virtual-biohackathons/covid-19-bh20/wiki/Wikidata

[22]https://www.wikidata.org/wiki/EntitySchema:E184

Wikidata is a multilingual knowledge base. One of the main objectives of ShExStatements is to ensure its use by multilingual users. ShExStatements mainly makes use of symbols and positions for specifying prefixes or even imports. EntitySchema:E210[23] is one such example which was generated from *examples/hospital.csv* in [12]. Other ShExStatements related to Biohackathon can also be found in the folder *examples/* [12].

Limitations and Future Works

Currently, ShExStatements only works with CSV files or tabular formats. Future works include supporting formats like Office Open XML format and Excel files. User evaluation tests are required to understand the challenges associated with writing shape expressions. Even though ShExStatements was tested mainly for data on Wikidata and for creating new entity schemas, it can also be used for generating ShEx for other RDF data sources. This needs to be further explored.

Another possible major work is to integrate ShExStatements in such a manner that users can directly create new entity schemas from the ShExStatements application. Currently, users need to manually copy the generated shape expression from ShExStatements and then create a new entity schema on Wikidata. ShExStatements applications can also be integrated with other works that support tabular formats for generating shape expressions. Finally, WikiProjects can also play an important role in the greater use of shape expressions for data validation. Contributors can develop simple shape expressions and link them to the appropriate WikiProject page.

6 CONCLUSION

Validation of data is important, especially for Wikidata considering the multilingual and multi-domain nature of the knowledge base. The recently introduced shape expressions (ShEx) is a major step in this direction. To promote its use, more tools may be required. Tabular formats, especially CSV files are commonly used file formats by Wikidata contributors while using tools like OpenRefine and QuickStatements. In this article, ShExStatements tool was presented to simplify writing shape expressions using CSV files. A subset of ShEx was used for building ShExStatements. With a command-line interface, a Python library, and a web interface, ShExStatements provide a wide variety of ways to generate shape expressions using this simpler subset.

ACKNOWLEDGMENTS

The author would like to thank the contributors of Wikidata and ShExStatements, especially the organizers and participants of Wiki Techstorm 2019 [11] for their feedback.

REFERENCES

[1] Iovka Boneva, Jérémie Dusart, Daniel Fernández-Álvarez, and José Emilio Labra Gayo. 2019. Shape Designer for ShEx and SHACL constraints. In *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 26-30, 2019 (CEUR Workshop Proceedings, Vol. 2456)*, Mari Carmen Suárez-Figueroa, Gong Cheng, Anna Lisa Gentile, Christophe Guéret, C. Maria Keet, and Abraham Bernstein (Eds.). CEUR-WS.org, 269–272. http://ceur-ws.org/Vol-2456/paper70.pdf

[23]https://www.wikidata.org/wiki/EntitySchema:E210

[2] Guillermo Facundo Colunga, Alejandro González Hevia, Emilio Rubiera Azcona, and José Emilio Labra Gayo. 2020. ShEx-Lite: Automatic Generation of Domain Object Models from a Shape Expressions Subset Language. In *Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020), Globally online, November 1-6, 2020 (UTC) (CEUR Workshop Proceedings, Vol. 2721)*, Kerry L. Taylor, Rafael S. Gonçalves, Freddy Lécué, and Jun Yan (Eds.). CEUR-WS.org, 148–152. http://ceur-ws.org/Vol-2721/paper536.pdf

[3] Antonin Delpeuch. 2020. A survey of OpenRefine reconciliation services. In *Proceedings of the 15th International Workshop on Ontology Matching co-located with the 19th International Semantic Web Conference (ISWC 2020), Virtual conference (originally planned to be in Athens, Greece), November 2, 2020 (CEUR Workshop Proceedings, Vol. 2788)*, Pavel Shvaiko, Jérôme Euzenat, Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, and Cássia Trojahn (Eds.). CEUR-WS.org, 82–86. http://ceur-ws.org/Vol-2788/om2020_STpaper3.pdf

[4] Eric Prud'hommeaux, tombaker, Glenna, Jose Emilio Labra Gayo, mrolympia, andrawaag, Lucas Werkmeister, and David Booth. 2018. shex.js - Javascript implementation of Shape Expressions. https://doi.org/10.5281/zenodo.1213693

[5] Herminio García-González, Iovka Boneva, Slawek Staworko, José Emilio Labra Gayo, and Juan Manuel Cueva Lovelle. 2020. ShExML: improving the usability of heterogeneous data mapping languages for first-time users. *PeerJ Comput. Sci.* 6 (2020), e318. https://doi.org/10.7717/peerj-cs.318

[6] Timothy Kanke. 2018. Preliminary Exploration of Knowledge Curation Activities in Wikidata WikiProjects. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries, JCDL 2018, Fort Worth, TX, USA, June 03-07, 2018*, Jiangping Chen, Marcos André Gonçalves, Jeff M. Allen, Edward A. Fox, Min-Yen Kan, and Vivien Petras (Eds.). ACM, 349–350. https://doi.org/10.1145/3197026.3203878

[7] Jose Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitris Kontokostas. 2017. *Validating RDF Data*. Synthesis Lectures on the Semantic Web: Theory and Technology, Vol. 7. Morgan & Claypool Publishers LLC. 1–328 pages. https://doi.org/10.2200/s00786ed1v01y201707wbe016

[8] Dustin Lange, Christoph Böhm, and Felix Naumann. 2010. Extracting structured information from Wikipedia articles to populate infoboxes. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, Jimmy Huang, Nick Koudas, Gareth J. F. Jones, Xindong Wu, Kevyn Collins-Thompson, and Aijun An (Eds.). ACM, 1661–1664. https://doi.org/10.1145/1871437.1871698

[9] Edward L. Platt and Daniel M. Romero. 2018. Network Structure, Efficiency, and Performance in WikiProjects. In *Proceedings of the Twelfth International Conference on Web and Social Media, ICWSM 2018, Stanford, California, USA, June 25-28, 2018*. AAAI Press, 251–260. https://aaai.org/ocs/index.php/ICWSM/ICWSM18/paper/view/17901

[10] Eric Prud'hommeaux, José Emilio Labra Gayo, and Harold R. Solbrig. 2014. Shape expressions: an RDF validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014*, Harald Sack, Agata Filipowska, Jens Lehmann, and Sebastian Hellmann (Eds.). ACM, 32–40. https://doi.org/10.1145/2660517.2660523

[11] John Samuel. 2020. Wikimedian John Samuel and his experiences at Wiki Techstorm 2019. https://diff.wikimedia.org/2020/01/07/wikimedian-john-samuel-and-his-experiences-at-wiki-techstorm-2019/

[12] John Samuel, Tom Baker, Nishad Thalhath, and Eric Prud'hommeaux. 2020. ShExStatements: v0.7. https://doi.org/10.5281/ZENODO.3723870

[13] ShEx. [n.d.]. ShEx - Shape Expressions. http://shex.io/

[14] Harold R. Solbrig, Eric Prud'hommeaux, Grahame Grieve, Lloyd McKenzie, Joshua C. Mandel, Deepak K. Sharma, and Guoqian Jiang. 2017. Modeling and validating HL7 FHIR profiles using semantic web Shape Expressions (ShEx). *Journal of Biomedical Informatics* 67 (2017), 90 – 100. https://doi.org/10.1016/j.jbi.2017.02.009

[15] Slawek Staworko, Iovka Boneva, José Emilio Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, and Harold R. Solbrig. 2015. Complexity and Expressiveness of ShEx for RDF. In *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium (LIPIcs, Vol. 31)*, Marcelo Arenas and Martín Ugarte (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 195–211. https://doi.org/10.4230/LIPIcs.ICDT.2015.195

[16] Katherine Thornton, Harold Solbrig, Gregory S. Stupp, José Emilio Labra Gayo, Daniel Mietchen, Eric Prud'hommeaux, and Andra Waagmeester. 2019. Using Shape Expressions (ShEx) to Share RDF Data Models and to Guide Curation with Rigorous Validation. In *The Semantic Web - 16th International Conference, ESWC 2019, Portorož, Slovenia, June 2-6, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11503)*, Pascal Hitzler, Miriam Fernández, Krzysztof Janowicz, Amrapali Zaveri, Alasdair J. G. Gray, Vanessa López, Armin Haller, and Karl Hammar (Eds.). Springer, 606–620. https://doi.org/10.1007/978-3-030-21348-0_39

[17] Denny Vrandecic. 2012. Wikidata: a new platform for collaborative data collection. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab (Eds.). ACM, 1063–1064. https://doi.org/10.1145/2187980.2188242