tapshex

Release 0.2.3

Tom Baker

Feb 17, 2023

CONTENTS

tapshex is a package and command-line utility for reading and interpreting CSV Files formatted according to the DCTAP Model and expressing them as schemas in the Shape Expressions language (ShEx). This package builds on classes and functions defined in a related package, dctap, and extends them with features that are explicitly aligned with ShEx. The command-line tool provided by **tapshex** is very similar to the dctap tool provided by a related package, dctap-python.

The **tapshex** documentation extensively links to, and builds on, the documentation for dctap where interested users will find a general description of the DCTAP Model and a DCTAP Glossary. The **tapshex** documentation features, in addition, a *ShEx Glossary*.

This package is under development at https://github.com/tombaker/tapshex, where issues or questions can be posted in its issue tracker.

CHAPTER

ONE

ABOUT DCTAP-PYTHON

CHAPTER

TWO

DCTAP ELEMENTS

In the DCTAP Model, a Shape groups a set of Statement Templates, each of which describes one type of Statement in Instance Data about a specified Entity. Each of these two components (Shapes and Statement Templates) has its own (extensible) set of DCTAP Elements:

2.1 Statement Constraint Elements

Removed from this section:

- mandatory / repeatable
- MinInclusive / MaxInclusive
- MinLength / MaxLength

This section:

- propertyID / propertyLabel
- valueNodeType
- valueDataType
- valueConstraint / valueConstraintType
  - Value constraints with no value constraint types
  - Value constraint types with no value constraints
  - Built-in value constraint types
    - Picklist
    - Pattern
    - IRIStem
    - LanguageTag
  - Custom value constraint types
- valueShape
- note

2.2 Shape Elements

There are two Shape elements. If the shapeID element is not used in a given DCTAP instance, it will be assigned a default value (which can be customized in the config file - see Default Shape Identifier).

- shapeID / shapeLabel

CHAPTER

THREE

CONFIGURATION

dctap has built-in default config settings (see defaults.py). By generating and editing a config file (see Initialize a config file), the following defaults can be tweaked:

- Default Shape Identifier [dctap]
- Namespace Prefix Mappings [dctap]
- Extra Elements [dctap]
  - Extra statement template elements [dctap]
  - Extra shape elements [dctap]
- dctap:config/list_elements/index [dctap]
  - dctap:config/list_item_separator/index [dctap]
- Element Aliases [dctap]

Modify:

- dctap:config/list_elements/valueNodeType/index

CHAPTER

FOUR

DCTAP AND CSV

The DC Tabular Application Profiles (DCTAP) model adheres to RFC 4108, the Common Format and MIME Type for Comma-Separated Values (CSV) Files. The CSV format consists of exactly one two-dimensional grid of rows and columns, so DCTAP is limited to things that can readily be encoded in two dimensions.

Modern spreadsheet programs, such as Excel and Google Sheets, transcend these limitations by supporting multiple grids in separate tabs, with hyperlinks across tabs — features which support drop-down menus and the construction of simple tabular databases — though at the cost of using non-standard file formats. Implementations of DCTAP in the wild have already leveraged such features to good effect, as "extensions" to DCTAP, but for **tapshex**, such extensions are out of scope.

As implemented in **dctap-python**, the base DCTAP model can be extended for use with extra elements defined specifically to support Shape Expressions language (ShEx).

CHAPTER

FIVE

DCTAP AND SHEX

The DCTAP model is intended to be usable in a wide range of implementation scenarios, even with technologies not based on RDF. So as not to clash with assumptions underlying such implementations, DCTAP was intentionally defined with the weak semantics. allowing implementers the freedom to define their own interpretations.

The DCTAP model, for example, does not articulate an overly specific view on how shapes relate to data. ShEx, in contrast, does have a strong view. In ShEx, the triple constraints of a shape are evaluated against all of the triples in an RDF graph that touch a given data node (the "focus node"), as determined by a "shape map" that specifies the set of focus nodes either by query or by enumeration. ShEx is also very precise about whether shapes, and their statement constraints, are by default considered "open" or "closed", questions that the DCTAP model deliberately leaves out of scope.

The **tapshex** module supports a subset of ShEx that is expressive enough for the simplest use cases but falls far short of ShEx in its expressivity. One might use DCTAP with ShEx extensions in the following ways:

- If the subset of ShEx features is sufficient, one might use DCTAP as an alternative to the standard syntaxes for ShEx — the compact syntax (ShExC), JSON syntax (ShExJ), and the less frequently used RDF syntax (ShExR).
- DCTAP can serve as an on-ramp to ShEx itself, and specifically to the ShExC syntax — a starter syntax for users new to ShEx who may find the spreadsheet format more approachable or easier to grok than the Turtle-like ShExC syntax, where users need not know where ShExC expects curly, pointy, or square brackets, semi-colons, or at-signs, or in what order node and cardinality constraints are placed in a triple constraint.
- DCTAP can serve as a tool for learning ShExC syntax.
- One might use DCTAP for rapidly prototyping ShEx schemas, then gradually leave DCTAP behind and continue on with the more expressive ShEx.

As discussed in the pages below, the choice between DCTAP and ShEx may be determined by specific requirements for expressivity.

CHAPTER

SIX

SHEX FEATURES NOT SUPPORTED BY DCTAP

The Shape Expressions language (ShEx) is a sophisticated language for describing RDF graph structures, with a well-defined grammar and a level of expressivity beyond what is attainable in a two-dimensional table.

Many ShEx features are not supported in the base DCTAP model, notably:

6.1 Reusable expressions

In a ShEx schema, one can assign a name to a frequently used set of node constraints (eg, to say: "integer between 1970 and 2022", then reference that name in the context of multiple triple constraints. A triple expression can also be defined and labeled just once, then referenced in multiple shapes (eg, if the subjects of "user" and "employee" shapes are both expected to have names and email addresses).

In a DCTAP instance, constraints are repeated in every statement template to which they apply, and statement templates are repeated in every shape to which they apply. In other words, a ShEx schema can be "DRY", while DCTAP instances are "WET" ("Write Every Time"). Of course, DCTAP can serve as a stepping stone to ShEx, where a wet schema, once generated, can be dried.

6.2 Combinations with AND, OR, or NOT

In ShEx, triple constraints can express choices — for example, to say that a person must be described either with `foaf:name` or with the combination `foaf:givenName` and `foaf:familyName`.

6.3 Nested shapes

In DCTAP, each shape is assigned an identifier and referenced with that identifier.

In ShExC syntax,, if a shape is only needed by one other shape, that shape can be embedded, anonymously, within the other.

6.4 Inverse triple constraints

In DCTAP, statement templates only ever describe "outgoing triples" — statements about the subject of a given shape. In ShEx, however, inverse triple constraints can also describe "incoming triples" — statements in which the focus node is the object.

6.5 Negative triple constraints

In ShEx, a schema can describe triples that *must not* appear in the data.

6.6 Imported schemas

A ShEx schema can reference shapes and triple expressions in schemas that are "imported".

Other features in the pipeline for the next release of the ShEx semantics include:

- **inheritance**: defining a shape expression as an extension or restriction of another shape
- **abstract shapes**: flagging a shape such that only shapes inheriting from that shape may satisfy the shape (analogously to object-oriented programming, where abstract classes may not be instantiated, only used for deriving non-abstract classes)

CHAPTER

SEVEN

DCTAP EXTENSIONS IN SUPPORT OF SHEX

7.1 Shape element: CLOSED

In ShEx, shapes are matched against outgoing arcs from a given focus node in terms of the predicates mentioned in the triple constraints, and any other outgoing arcs — triples that do not have the mentioned properties as their predicates — are by default simply ignored. However, a given shape can be "closed" (by using the ShExC keyword `CLOSED`), which means that *all* outgoing arcs from the given focus node must match the given set of triple constraints.

In other words, a closed shape will pass validation only if all of its outgoing arcs match *and* there are no other outgoing arcs from that node in the data. -

7.2 Shape element: EXTRA

In ShEx,, once a property is "mentioned" in a triple constraint, that mention "closes the property". As explained in the ShEx Primer: "By default, for an RDF data node to match that shape, every outgoing arc from that node that uses a mentioned predicate must match a triple constrain in the shape".

This interpretation coincides with what "property mentions" are commonly understood to mean in the context of Dublin Core-style application profiles.

ShEx also allows a given predicate to be "opened", such that any number of additional arcs using that predicate can be accepted. For example, if shape has two triple constraints that mention the predicate `rdf:type` — one constrained to class A as its value and one constrained to class B, then an outgoing type arc with a value of class C would by default fail validation. However, if that predicate is flagged in the shape as an "extra" property by use of the ShExC keyword `EXTRA`, the outgoing type arc with a value of class C would pass validation. This behavior of ShEx can be supported by adding `EXTRA` as an extension element.

7.3 Literal constraining facets

ShEx supports the use of XML Schema constraining facets, which include constraints on numeric values and constraints on strings. While **tapshex** could be further extended to support this entire set of facets, it already supports a subset:

- `MinInclusive` / `MaxInclusive`, which are used to define a range within which a numeric value must fall.
- `MinLength` / `MaxLength`, which are used to define a minimum and maximum length of a string value.

The string contraining facet `pattern` is already supported by dctap by using a regular expression with **valueConstraint** and the keyword "pattern" with **valueConstraintType**.

7.4 Node type: NONLITERAL

The **dctap** module supports, by default, keywords for the three types of node as defined in the RDF model: IRI, BNODE, and LITERAL.

Like ShEx, **tapshex** supports, in addition, the keyword NONLITERAL, as an alias for "IRI or BNODE".

In both **dctap** and **tapshex**, keywords are case-insensitive, with the difference that in **dctap**, they are normalized to lowercase and in **tapshex** to uppercase.

CHAPTER

EIGHT

SHEX GLOSSARY

This glossary defines terms that are specific to ShEx; definitions in double quotes are taken verbatim from the Shape Expressions Primer. See the DCTAP Glossary for definitions of **Application Profile**, **Blank Node**, **Compact IRI**, **CSV File**, **Datatype**, **DCTAP Element**, **DCTAP CSV File**, **Description**, **Entity**, **Instance Data**, **IRI**, **Language Tag**, **Literal**, **Property**, **Predicate Constraint**, **Shape**, **Statement**, **Statement Template**, **URI**, **Value**, **Value Constraint**, and **Vocabulary**.

Constraint
: "A restriction on the set of permissible nodes or triples."

Focus Node
: "An RDF data node of interest that is examined during validation."

Shape Map
: "A collection of pairs of RDF data nodes and ShEx shapes that is used for invoking validation and reporting results."

Shape (ShEx)
: "A triple expression against which a focus node is tested to see whether all incoming or outgoing arcs, which match predicates in the triple expression, have the appropriate cardinality and values."

ShExC
: "A compact syntax meant for human eyes and fingers."

ShExJ
: "A JSON-LD representation meant for machine processing."

ShExR
: "The RDF interpretation of ShExJ expressible in any RDF syntax."

ShEx Schema
: "A collection of shape expressions."

Triple Constraint
: "A constraint with a given predicate which is tested against RDF triples with that predicate and the focus node as subject (or object, for Inverse Triple Constraints)."

Triple Expression
: "A collection of triple constraints which may be combined in groups or choices."

Value (ShEx)
: "A shorthand designation for the RDF node at the opposite end of an RDF data type from a focus node. In typical usage, this is the object of a triple or, for inverse triple constraints, its subject."

INDEX