JSONPath

June 15th (Tuesday), 09:00-11:00 UTC
(11:00–13:00 CEST, 02:00–04:00 PDT)

#97

Addresses issue #84 (terminology)

Most suggested changes were picked for #101

Remaining issue: Values vs. Literals

JSON RFC (RFC 8259)

Describes **JSON text**.

Discusses **JSON values**,
but never really what is different between
the value itself and its representation

`10, 1e1, 100e-1, 10.0, 1.0e1, 1.00e1, 10.00`
are all the same **number**
in different notation (“number **literal**”)

Literals in JSONPath

JSONPath syntax denotes **operators**, punctuation, and **values**

We need to explain where JSONPath literals for values are
the same as in JSON (e.g., numbers, false, true, null),
and where we have our own literal syntax (strings).

JSONPath strings (the **values**):
sequence of one or more Unicode codepoints.

JSON string literals

JSON string literals:
Always double-quoted.
No C0 characters.
Can backslash-escape `" \ / b f n r t` and `uXXXX`; nothing else.

```
> a = "\"\\'\""
> console.log(a)
"\'"
> JSON.parse(a)
Uncaught SyntaxError: Unexpected token ' in JSON at position 2
```

JavaScript string literals: way more permissive. Irrelevant.

JSONPath string literals (#97)

JSONPath string literals:
almost, but not entirely unlike JSON string literals.

— allow outer single quotes and their escaping?

— what else?

What else?

Any other literals that differ from their JSON counterparts?

Any literals for data types that aren't in JSON?

— JSONPath nodes:
  we already have the »$« literal/expression/…
— JSONPath regexps?
— What else?

#99 (#98)

#98: Stefan's selector PR
#99: Fixes so that it builds; branch name ≠ `main`

Discussion happens under both PRs (sorry about that)
#99 discussion mostly editorial; can ignore today

#98 surfaces undecided questions

#98 really has the **expression language** as a prerequisite.

How powerful should the expression language be?

— should it have arithmetic operations (`+ - * / %`)?
— should it have function calls?
— should there be literals (or constructor notations) for structured values?

What is our stance to implicit conversions?
(Emerging consensus was: No implicit conversions.)
What about conversion to Boolean ("truthy")?

Example: Comparison with structured values

Should comparison with structured values (e.g., @.foo == [1, 2]) be supported?

If it is not supported, should this silently fail or the attempt cause a syntax error (in #99, it causes a syntax error, but then the text says something else).

— Data types: can we even write and pass around `[1, 2]`?

Stefan's topics: Terminology

RFC 8259 on types of JSON values:

> JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).

So, with respect to types imported from JSON, we have
- primitive (as opposed to atomic/simple), and
- structured (as opposed to complex/container).

(Note that "container" is useful to name the container itself, as opposed to including what's in there.)

Stefan's topics: Selectors

— Fundamental point is the alternate wording in text … :
  — … selects a value.
  — … selects a node.

Both is possible with JSONPath, but we should stick with one throughout the text.

(cabo:
Clearly, we are selecting nodes, which contain their values.)

— Is a dot-member name allowed to start with a DIGIT?

(cabo:
related: what does `.1` applied to `[1, 2, 3]` mean?)

— Are unions allowed to contain wildcards, descendant-selectors and filter-selectors?

— Do we need to specify the order of evaluation of the descendant-selector?

— Discuss the proposed lean backward compatible syntax `[?<expr>]` in contrast to original `[?(<expr>)]`.

— Discuss proposal of `in-op` operator.

— Should comparison with structured values be allowed?

(cabo:
This first requires defining literal and/or constructor syntax for structured values.)

— What's the opinion about supporting functions?

Related to this is:
How to access a specific member in `{...,"key":5,...}`
via filter expression `[?@...]`?