JSONPath standardization

Almost, but not entirely unlike XPath for JSON

Discussion slot at DISPATCH WG meeting @ IETF108

JSON is a data representation language

- JSON is the premier format for representing tree-shaped data for interchange
- Often, there is a need to identify elements or subtrees in such a tree, without transferring the whole tree
- XML has XPath, a complex, Turing-equivalent query language
- JSON has JSONPath, proposed in 2007, but never standardized

JSONPath examples

| XPath | JSONPath | Result | | | | |
|----------------------|-------------------------------------|--|--|--|--|--|
| /store/book/author | <pre>\$.store.book[*].author</pre> | the authors of all books in the store | | | | |
| //author | \$author | all authors | | | | |
| /store/* | \$.store.* | all things in store, which are some books and a red bicycle. | | | | |
| /store//price | \$.storeprice | the price of everything in the store. | | | | |
| //book[3] | \$book[2] | the third book | | | | |
| //book[last()] | \$book[(@.length-1)] \$book[-1:] | the last book in order. | | | | |
| //book[position()<3] | \$book[0,1] \$book[:2] | the first two books | | | | |
| //book[isbn] | \$book[?(@.isbn)] | filter all books with isbn number | | | | |
| //book[price<10] | \$book[?(@.price<10)] | filter all books cheapier than 10 | | | | |
| //* | \$* | all Elements in XML document. All members of JSON structure. | | | | |

```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the
Century",
        "price": 8.95
       "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
    "bicycle": {
      "color": "red",
      "price": 19.95
```

Aren't there other ways to do this?

Sure:

- RFC 6901, JSON Pointer, very similar in idea, but different in syntax, and limited towards pointing into a single place in a known structure
- XPath extensions for JSON all the complexity for very little functionality
- (Insert your favorite query language here)
- But the question is less "is JSONPath always the best tool for the job" it is being used a lot, and it would benefit from a common standard

Why do we want this now?

- Well, many of us have wanted this for a while... (Current occasion for me: JSONPath fits some IoT discovery well)
- Now the stars seem to have aligned to make this possible:
 - The original JSONPath author is interested in getting this done
 - An amazing project has started documenting implementation deviations

• So let's do it, like we did RFC 6901 earlier.

Why this isn't trivial

- JSONPath was defined in 2007, and implemented many times since
- JSONPath left expressions/filters to an "underlying scripting language"
 - JSONPath implementations have used their implementation language or a synthetic language patterned after common usage
 - These are mostly close, but not identical
- Lots of details differ
 - Fortunately, there is https://cburgmer.github.io/json-path-comparison/ (225 test cases against 37 implementations, "Proposal A" in the making)

| JSONPath Comparison | | | | | | | | | | • | | | | | | Objective CC Kotlin JavaScrip LavaScrip Golang EHang Cpp Cpp | dotNET Scala Scala Rust Ruby Raku Python |
|---|--|---|--|---|---|--|---|--|--|---|---|--|-------------------------------|--------------------------|---|--|--|
| See how JSONPath is implemented across different languages. See the FAQ for why we are doing this | s and where the results come fron | Cpp Cbjure | Erlang Elixir | Golang | Java JavaScript | PHP Objective- C C Kotlin | Python Perl | Scala Rust Ruby Raku | dotNET | | | | | | | remo jani jani jani jani jani filo Goes com.apywa com.apywa com.jaywa com.jaywa com.jithut json json githut | Prop Manate Jsonp Jsonp jsonp jsonp jsonp jsonp jsonp jsonp |
| | | ExJsonPath jsoncons json- path json- glib JSONPath.sh | jsonsice i github.com- PaesslerAG- jsonpath i ejsonpath warpath | github.com- wmware- labs-yami- jsonpath github.com- ajson github.com- ofiveagle- jsonpath github.com- | com.jayway.jsonp com.github.jsur jsonpath- plus jsonpath Goessner jsonpath | galbar- jsonpath flow- jsonpath Goessner 2 SMJJSONPatt | jsonpath- rw jsonpath JSON- Path remorhaz- jsonpath | jsonpath jsonpath_lib jsonpath jsonpath JSON- Path | Proposal A Manatee.Json JsonPathLib | | # Dot notation with | vildcard after recursive | descent ⁴ | | \$* | path y pa | e.Json > athLib > NET X sttr_lib > path X pa |
| # Array slice # Array slice on exact match | \$[1:3] \$[0:5] | | | | fer oath | | | | | | # Dot notation with | | descent on null value ar | ay ⁴ | \$* \$* | V X V V V E V V E V V V | x |
| # Array slice on exact match # Array slice on non overlapping array # Array slice on object | \$[7:10] \$[1:3] | e | e | e | V V V V V V V V V V e X V V V X | v v v v | V V V | v v v v v . | v v v v | | # Dot notation without | ut root | | | key \$[?(@.key)] | | |
| # Array slice on partially overlapping array # Array slice with large number for end # Array slice with large number for end and negative step | \$[1:10] \$[2:113667776004] \$[2:-113667776004:-1] | ✓ e × ✓ ✓ | e 🗸 🗸 🗸 | e v v | e | × | ✓ e ✓ ✓ | e | e 🗸 🗙 🗸 | | # Filter expression a | | rildcard after recursive de | scent ⁴ | \$*[?(@.id>2)] \$[?(@.id=2)] | | e • • • • • • • • • • • • • |
| # Array slice with large number for start # Array slice with large number for start # Array slice with large number for start end negative step | \$[-113667776004:2] \$[113667776004:2:-1] | • e e • • | e & & & & & | e s | e • • • • • • | • e • • | e 5 5 | • • • • • • • | e • • • | | # Filter expression v | | | | \$[?(@.key+50==100)] \$[?(@.key>42 && @.key<44)] | * | e • • • • • • • • • • • • • |
| # Array slice with negative start and end and range of -1 # Array slice with negative start and end and range of 0 | \$[-4:-5] \$[-4:-4] | | e | × v v | VVVVVVVVVV | v v v v | V V V | v x x x v v . | v v v v | | # Filter expression v | ith boolean and operat | or and value false | | \$[?(@.key>0 && false)] \$[?(@.key>0 && true)] | | e • • • • • • • • • • • • • • • • • |
| # Array slice with negative start and end and range of 1 # Array slice with negative start and positive end and range of -1 # Array slice with negative start and positive end and range of 0 | \$[-4:-3] \$[-4:1] \$[-4:2] | | e | v v v | X | v x v v v | V X V | v v v x v v | v v v v | | # Filter expression v | ith boolean and operat | r | | \$[?(@.key>43 @.key<43)] | | e s |
| # Array slice with negative start and positive end and range of 1 # Array slice with negative step | \$[-4:3] \$[3:0:-2] | e x x v x | e | × v v | · · · · · · | v x v v v | v × v v | v x v x v . | v v v v | | | ith boolean or operato ith boolean or operato | | | | | |
| # Array slice with negative step on partially overlapping array # Array slice with negative step and start greater than end # Array slice with negative step only | \$[7:3:-1] \$[0:3:-2] \$[::-2] | • e • • e | e • • • • | e 😘 😘 | e • • • • • • | • e e • | No. 10 No. 10 | e • • • | e e e e | | # Filter expression v | | d current object literal | | \$[?(@['key']==42)] \$[?(@['@key']==42)] | V V E V X V E V <td>e</td> | e |
| # Array slice with open end # Array slice with open end and negative step | \$[1:] \$[3::-1] | | e | V V V | e • • • • • • | v v v v | V V V | v v v v v . | | | | ith bracket notation wi ith bracket notation wi | | | \$[?(@[1]=='b')] \$[?(@[1]=='b')] | * * * * * * * * * * * * * * * * * * * | |
| # Array slice with open start # Array slice with open start and end # Array slice with open start and end on object | \$[:2] \$[:] \$[:1 | ✓ X e ✓ ✓ | e | v v v | × | v x v v v | ✓ e ✓ ✓ | V X V | | / | | ith bracket notation wi | th number on object | | \$[?(@[1]=='b')] \$[?(@)] | 1, 1 2 1, 1 1 1 2 2 2 2 2 2 2 | |
| # Array slice with open start and end on object # Array slice with open start and end and step empty # Array slice with open start and negative step | \$[::] \$[:2:-1] | ✓ X e ✓ ✓ | e 🗸 🗴 🗸 🗸 | e 🗸 🗸 | x | v x v v v | v v | | ~ | | V | 10 | perators | | \$[?(@.a && (@.b @.c))] \$[?(@.a && @.b @.c)] | | |
| # Array slice with positive start and negative end and range of -1 # Array slice with positive start and negative end and range of 0 | \$[3:-4] \$[3:-3] | x | e | × | × | \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ | V | | | | | | | | | | e • • • • • • • • • • • • • • • • • • • |
| # Array slice with positive start and negative end and range of 1 # Array slice with range of -1 # Array slice with range of 0 | \$[3:-2] \$[2:1] \$[0:0] | e | e 🗸 🗸 🗸 | v v v | X | v v v | е | × | ~ | ✓ | | × | | | | 1. 1. e 1. 1. 1. 1. 1. | e , , |
| # Array slice with range of 1 # Array slice with start -1 and open end | \$[0:1] \$[-1:] | \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ | e 🗸 🗸 🗸 | × v v | V V V V V V V V V V V V V V V V V V V | v v v | | | | | | | | | \$[?(@.key==43)] | 1 | e |
| # Array slice with start -2 and open end # Array slice with start large negative number and open end on short array # Array slice with sten | \$[-2:] \$[-4:] \$[0:3:2] | X | e | e v v | V V V X V V V V V V V V V V V V V V V V | × × × | | | | | | | | | \$[?(@.id==2)] | 1. < | e , · · · · · · · · · · · · · · · · · · |
| # Array slice with step 0 # Array slice with step 0 | \$[0:3:0] \$[0:3:1] | ✓ e X ✓ ✓ •• e • • • • ✓ e X ✓ ✓ | e • • • • • | e e • | e e e • • • • • • • • • • • • • • • • • | V | V | V | V | V | V | V | V | | \$[?(@.d==["v1","v2"])] \$[?(@[0:1]==[1])] | * | |
| # Array slice with step and leading zeros # Array slice with step but end not aligned | \$[010:024:010] \$[0:4:2] | x | e | e | V V X X X X X | | 1 | • | • | 1 | , | • | , | | \$[?(@.*==[1,2])] \$[?(@.d==['v1','v2'])] | 1. < | |
| # Array slice with step empty # Array slice with step only # Bracket notation | \$[1:3:] \$[::2] \$['key'] | <pre></pre> | e 🗸 🗴 🗸 | e v v | X | | | | | | | | | | \$[?((@.key<44)==false)] \$[?(@.key==false)] | | |
| # Bracket notation on object without key # Bracket notation after recursive descent 4 | \$['missing'] \$[0] | e | e 🗸 🗸 x e | e v v | e | | _ | 1 | 1 | 1 | | × | × | 1 | \$[?(@.key==null)] \$[?(@[0:1]==1)] | | |
| # Bracket notation with NFC path on NFD key # Bracket notation with dot Description: | \$['ü'] \$['two.some'] | e | | | e | • | - | · • | • | • | - | ^ | ^ | • | \$[?(@[*]==2)] \$[?(@.*==2)] | | e • • • • • • • • • • • • |
| # Bracket notation with empty path # Bracket notation with empty string | \$["key"] \$[] \$[''] | e × e ✓ ✓ | e 🗸 🗸 e | e | ✓ × ✓ | | | | | | | | | | [?(@.key==-0.123e2)] [?(@.key==010)] | • • • • • • • • • • • • • • • • • • • | e • • • • • • • • • • • • • |
| # Bracket notation with empty string doubled quoted # Bracket notation with negative number on short array | \$[] \$[-2] | | | e • • • • | | | | | | | | | | | (@.d=={"k":"v"})] | | e • • • • • • • • • • • • • • • • • • • |
| # Bracket notation with number # Bracket notation with number on object # Bracket notation with number on short array | \$[2] \$[0] \$[1] | e • • • • • • | s s s s s | e • • | e • | * | A | , A | * | - 74 | - 74 | - 74 | - 74 | - 74 | (@.key=="value")] @.key=="hi@example.com")] | ** ** ** ** ** ** ** ** ** ** ** ** ** | |
| # Bracket notation with number on string # Bracket notation with number after dot notation with wildcard on nested arrays with different length | \$[0] th \$.*[1] | | e e e e | e • • | e • • | | | | | | | | | | 0.key=="some.value")] | V V E V V X V V V V V X V V V V V X V <td></td> | |
| # Bracket notation with number -1 # Bracket notation with number -1 on empty array | \$[-1] \$[-1] | e × ✓ ✓ ✓ ✓ e ✓ ✓ | e 🗸 🗸 e e | e v v | e v | | | | _ | _ | | | | | .key==true)] ms[?(@.key==\$.value)] | | |
| # Bracket notation with number 0 # Bracket notation with quoted array slice literal # Bracket notation with quoted closing bracket literal | \$[,:,] \$[,:,] \$[o] | e v e v v | e 🗸 🗸 🗸 | e v v | × | | е | | | | е | - 🗡 | × | е | .key>42)] .key>=42)] | | |
| # Bracket notation with quoted current object literal # Bracket notation with quoted dot literal | \$['e'] \$['e'] | e | e | e v v | × | | | | | | | | | | 0.d in [2, 3])] 2 in @.d)] | 1, 1, e 1, 1, 1, e e e e 1, 1, 1, e e e e | |
| # Bracket notation with quoted dot wildcard # Bracket notation with quoted double quote literal # Bracket notation with quoted escaped backslash | \$[,.,] \$[,.,] | e | e | | × | | | | | | | | | | [@.key<42)] (@.key<=42)] | 1, 1, e 1, e 1, e 1 e 1 1, 1, e 1 e 1 e | e • • • • e • • • e • • • • |
| # Bracket notation with quoted escaped single quote # Bracket notation with quoted number on object | \$[.6.] \$[./] | | e • e • • | e • • | | · 🛰 | * | * | е | * | * | * | * | * | ?(@.key*2==100)] [?(!(@.key==42))] | • • • • • • • • • • • • • • • • • • • | e • • • • • • • • • • • • • • |
| # Bracket notation with quoted root literal # Bracket notation with quoted special characters combined # Bracket notation with quoted string and unescaped single quote | \$['\$'] \$[':@."\$,*\'\\'] \$['single'quote'] | | e • e • e | e • • • | | | | | | | | | | | \$[?(!(@.key<42))] | • • • • • • • • • • • • • • • • • • • | e • • • • • • • • • • • • • • • • |
| # Bracket notation with quoted union literal # Bracket notation with quoted wildcard literal # Bracket notation with quoted wildcard literal # | \$[.*.] \$[.*.] | | e | e v v | | | | | | | | | | | \$[?(@.key!=42)] \$[?(@.name=~/hello.*/)] | | e • • • • • • • • • • • • • • • • • • • |
| # Bracket notation with quoted wildcard literal on object without key # Bracket notation with string including dot wildcard | \$['ni.*'] \$['two'.'some'] | X V e V | e | | × | | , | , | , | , | , | , | , | , | | | |
| # Bracket notation with two literals separated by dot # Bracket notation with two literals separated by dot without quotes # Bracket notation with wildcard on array | \$[two.some] \$[*] | X X e X ✓ X ✓ e X ✓ e ✓ ✓ X | e 🗸 🗸 e e | | X | V | ~ | ~ | ~ | ~ | ~ | ~ | ~ | V | \$[?(@.key=42)] \$[?(@.a[?(@.price>10)])] | V V E V E V E X E V V X X X V V V V E X V V V V V V V V V | |
| # Bracket notation with wildcard on empty array # Bracket notation with wildcard on empty object | \$[*] \$[*] | e | v v v v | e v v | V V V V V V V V V V | | | | | | | | | | \$[?(@.address.city=='Berlin')] \$[?(@.key-50==-100)] | | |
| # Bracket notation with wildcard on null value array # Bracket notation with wildcard on object # Bracket notation with wildcard after array slice | \$[*] \$[0:2][*] | VXXVVXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX<l< td=""><td>×</td><td>e v v</td><td>V V V V V V V V V V V V V V V V V V V</td><td>× ×</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><th>\$[?(1==1)] \$[?(@.key===42)]</th><td>1. <</td><td></td></l<> | × | e v v | V V V V V V V V V V V V V V V V V V V | × × | | | | | | | | | \$[?(1==1)] \$[?(@.key===42)] | 1. < | |
| # Bracket notation with wildcard after dot notation after bracket notation with wildcard # Bracket notation with wildcard after recursive descent 4 | \$[*].bar[*] \$[*] | <pre></pre> | × | e 🗸 🗸 | V V V V V V V V V V | | | ./ | ./ | ./ | ./ | ./ | | | \$[?(@.key)] \$[?(@.id)] | | |
| # Bracket notation without quotes # Dot bracket notation # Dot bracket notation with double quotes | \$[key] \$.['key'] \$.["key"] | • • e • • | e • • • • | e • • | | | _ ^ | V | • | | | · • | V | | [?(false)] 11)] | | |
| # Dot bracket notation without quotes # Dot notation | \$.[key] \$.key | X | e 🗸 🗸 🗴 X | e × ✓ | X | | | | | | | | | | | | e • • • • • • • • • • • • • • • • • • • |
| # Dot notation on array # Dot notation on array value # Dot notation on array with containing object matching key | \$.key \$.key \$.id | e | v v v v v | v v v | e | V V V V V V V V V V V V V V V V V V V | V V V V | | | | | | | | | | e • • • • • • • • • • • • • • • • • • • |
| # Dot notation on array with containing object matering key # Dot notation on empty object value # Dot notation on null value | \$.key \$.key | e | v v v v v | V V V | | V V X V X V X V V V V V V V V V V V V V | V V V V | V V | / | 1 | | ulter dot notation 4 | | | | 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1 | |
| # Dot notation on object without key # Dot notation after array slice | \$.missing \$[0:2].key \$. [1].key | e | e 🗸 🗸 🗸 | × v v | e | V V V V V V V V V V V V V V V V V V V | V V V V V | V V V V V V V V V V V V V V V V V V V | V | | # 0 | arter doculotation * | | | | | x x v v v v v v v v v v v |
| # Dot notation after bracket notation after recursive descent. ⁴ # Dot notation after bracket notation with wildcard # Dot notation after bracket notation with wildcard on one matching | \$[1].key \$[*].a \$[*].a | e X e V X | v x v X v v v v v | e | V V V V X V V V V V V X V V V V V | v v v v | v v v | <pre></pre> | v v v v | | # Root on scalar # Root on scalar fals | | | | | V X V V V V X V e V X V V V V X V <td>e × ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td> | e × ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ |
| # Dot notation after bracket notation with wildcard on some matching # Dot notation after filter expression | \$[*].a \$[?(@.id==42)].name | x | v v v v v e v v v | × | e | v v v v | v v v | v v v x v v | v v v v | | # Root on scalar true # Script expression | | | | | * * * * * * * * * * * * * * * * * * * | |
| # Dot notation after recursive descent * # Dot notation after recursive descent after dot notation * # Dot notation after union | \$key \$.storeprice \$[0,2].key | X X V V V X X X X X X X X X X X X X X X | X | e | V V V V V V V V V V V | V V V V V V V V V V V V V V V V V V V | V V V V V V V V V V V V V V V V V V V | | V V V V V V V V V V V V V V V V V V V | | # Union # Union with filter | | | | | e e e e e e e e e | |
| # Dot notation after union with keys # Dot notation with dash | \$['one','three'].key \$.key-dash | × | √ √ × × e | V V V | X | v v v v | V V V | × × | ✓ ✓ × ✓ | | # Union with keys # Union with keys or | object without key | | | \$['ka \$['missin | <pre></pre> | |
| # Dot notation with double quotes # Dot notation with double quotes after recursive descent 4 # Dot notation with empty path | \$."key" \$"key" \$. | • • • e | • • e • • | e • • | • • • • • • • • • • • • • • • • • • • | | | | | | # Union with keys at | | | | \$[:]['c','d'] \$[0]['c','d'] | V X E X V X V X V X V X V X V X V X V X V X V X V X V X V X V X V X V X | |
| # Dot notation with empty path # Dot notation with key named in # Dot notation with key named length | \$. \$.in \$.length | × | v v v v v | v v v | | v v v v | V V V | v v v v v . | v v v v | # Union with keys after dot notation with wildcard # Union with numbers in decreasing order | | | | \$.*['c','d'] \$[4,1] | | | |
| # Dot notation with key named length on array # Dot notation with key named null | \$.length \$.null | e | √ √ √ e e √ √ √ √ √ | × × ✓ ✓ | e × × × v v | V V V X X V V V | V V V V V V V V | VEVVVVVV | v v x v v v v v | | # Union with slice at | d number | | | \$[1:3,4] \$[0,1] | | |
| # Dot notation with key named true # Dot notation with key root literal # Dot notation with non ASCII key | \$.true \$.\$ \$.属性 | e • • • • | | e • • | V V | | • e • • | | | | # Union with wildcar | | | | \$[*,1] | | |
| # Dot notation with number # Dot notation with number on object | \$.2 \$.2 | e • • • • • • • × × | * * * * * * * * * * * * * * * * * * * | • • • • • • • • • • • • • • • • • • • | e • • • • • • • • • • • • • • • • • • • | • • • • • • • • • • • • • • • • • • • | • • • • • · · · · · · · · · · · · · · · | * * * * * * * * * * * * * * * * * * * | • • • • • • • | | Explanation | | | | | | |
| # Dot notation with number -1 # Dot notation with single quotes # Dot notation with single quotes after recursive descent 4 | \$1 \$.'key' \$'key' | • • • • • • • | • • e • • | to the second | e % % % * % | e e e e e | No. No. No. No. | | e sa sa sa sa | | | is implementation mat not match the conser | ches the consensus of rensus. | sults. | | | |
| # Dot notation with single quotes after recursive descent * # Dot notation with single quotes and dot # Dot notation with wildcard on array | \$. *key \$. *some.key' \$.* | e | • • • • • | e e • | * * * * * * * * * * * * * * * * * * * | er er er er er | | | e e e e | | • ■and ■ : no clea | | ne implementations, but | ∎indicates a majority (| and possible future consensus). | | |
| # Dot notation with wildcard on empty array # Dot notation with wildcard on empty object | \$.* \$.* | e X | e | e | V V V V V V V V | <pre></pre> | V V V V V V V V | ✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓<l< td=""><td></td><td></td><td> ¹ This implement </td><td>tion returns a single va</td><td></td><td></td><th>of an array of a single value). matches.</th><td></td><td></td></l<> | | | ¹ This implement | tion returns a single va | | | of an array of a single value). matches. | | |
| # Dot notation with wildcard on object # Dot notation with wildcard after dot notation after dot notation with wildcard # Dot notation with wildcard after dot notation with wildcard on nested arrays | \$.* \$.*.bar.* \$.*.* | | v v v x | e | <pre></pre> | v v v v | ✓ ✓ ✓ × | v v x v v x : | × | | | | | | eturn a single match results in no match. tions might apply different and even non-deter | For comparison the results are sorted into a canonical order. | |
| # Dot notation with wildcard after dot notation with wildcard on nested arrays | ** ** | e x v v v | * * * * * | e | | <u> </u> | , , , , , x | v v x v v x ; | v 4 4 4 | | | | , | • | | | |

WG needs to decide on direction

- Find lowest common denominator and standardize that
 - Won't cover that many real-world examples, no big benefit
- Find the gaps and start filling them all by dumping in more rubble
 - High complexity of the result, bugs will be plenty
- Define a middle ground, filling gaps neatly where existing usage abounds
 - The right thing, but probably needs some more detailed guidance
 - Needs input from implementers and users
 - Janus approach: look both back and forward

Where do we want to do this?

- New Working Group (?)
- Revive JSON Working Group specifically for this
- Do this in CBOR working group because that is alive, has the same generic data model, and actually needs a tree query language as well (and probably would promise to standardize JSONPath for JSON first)
- Stuff this into a new HTTPAPI working group, because JSONPath often is used over HTTP (as is everything else)
- → For the DISPATCH WG to decide!

Please DISPATCH!