JSONPath - XPath for JSON

A frequently emphasized advantage of XML is the availability of plenty tools to analyse, transform and selectively extract data out of XML documents. XPath is one of these powerful tools.

It's time to wonder, if there is a need for something like XPath4JSON and what are the problems it can solve.

- Data may be interactively found and extracted out of JSON structures on the client without special scripting.
- JSON data requested by the client can be reduced to the relevant parts on the server, such minimizing the bandwidth usage of the server response.

If we agree, that a tool for picking parts out of a JSON structure at hand does make sense, some questions come up. How should it do its job? How do JSONPath expressions look like?

Due to the fact, that JSON is a natural representation of data for the C family of programming languages, the chances are high, that the particular language has native syntax elements to access a JSON structure.

The following XPath expression

```
/store/book[1]/title
```

would look like

```
x.store.book[0].title
```

or

```
x['store']['book'][0]['title']
```

in Javascript, Python and PHP with a variable `x` holding the JSON structure. Here we observe, that the particular language usually has a fundamental XPath feature already built in.

The JSONPath tool in question should …

- be naturally based on those language characteristics.
- cover only essential parts of XPath 1.0.
- be lightweight in code size and memory consumption.
- be runtime efficient.

# JSONPath expressions

JSONPath expressions always refer to a JSON structure in the same way as XPath expression are used in combination with an XML document. Since a JSON structure is usually anonymous and doesn't necessarily have a "root member object" JSONPath assumes the abstract name `$` assigned to the outer level object.

JSONPath expressions can use the *dot*–notation

```
$.store.book[0].title
```

or the *bracket*–notation

```
$['store']['book'][0]['title']
```

for input pathes. Internal or output pathes will always be converted to the more general *bracket*–notation.

JSONPath allows the *wildcard* symbol * for member names and array indices. It borrows the *descendant* operator '..' from E4X and the *array slice syntax* proposal `[start:end:step]` from ECMASCRIPT 4.

Expressions of the underlying scripting language `(<expr>)` can be used as an alternative to explicit names or indices as in

```
$.store.book[(@.length-1)].title
```

using the symbol '@' for the current object. Filter expressions are supported via the syntax `?(<boolean expr>)` as in

```
$.store.book[?(@.price < 10)].title
```

Here is a complete overview and a side by side comparison of the JSONPath syntax elements with its XPath counterparts.

XPath	JSONPath	Description
/	$	the root object/element
.	@	the current object/element
/	. or []	child operator
..	n/a	parent operator
//	..	recursive descent. JSONPath borrows this syntax from E4X.
*	*	wildcard. All objects/elements regardless their names.
@	n/a	attribute access. JSON structures don't have attributes.

[]	[]	subscript operator. XPath uses it to iterate over element collections and for predicates. In Javascript and JSON it is the native array operator.
|	[,]	Union operator in XPath results in a combination of node sets. JSONPath allows alternate names or array indices as a set.
n/a	[start:end:step]	array slice operator borrowed from ES4.
[]	?()	applies a filter (script) expression.
n/a	()	script expression, using the underlying script engine.
()	n/a	grouping in Xpath

XPath has a lot more to offer (Location pathes in not abbreviated syntax, operators and functions) than listed here. Moreover there is a remarkable difference how the subscript operator works in Xpath and JSONPath.

- Square brackets in XPath expressions always operate on the *node set* resulting from the previous path fragment. Indices always start by 1.
- With JSONPath square brackets operate on the *object* or *array* addressed by the previous path fragment. Indices always start by 0.

# JSONPath examples

Let's practice JSONPath expressions by some more examples. We start with a simple JSON structure built after an XML example representing a bookstore (original XML file).

```
{ "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
    }
  }
}
```

XPath	JSONPath	Result
`/store/book/author`	`$.store.book[*].author`	the authors of all books in the store
`//author`	`$..author`	all authors
`/store/*`	`$.store.*`	all things in store, which are some books and a red bicycle.
`/store//price`	`$.store..price`	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.

# JSONPath implementation

JSONPath is implemented in Javascript for clientside usage and ported over to PHP for use on the server.

Usage

All you need to do is downloading either of the files

- jsonpath.js
- jsonpath.php

include it in your program and use the simple API consisting of one single function.

```
jsonPath(obj, expr [, args])
```

parameters:

`obj (object|array)`:
Object representing the JSON structure.

`expr (string)`:
JSONPath expression string.

`args (object|undefined)`:
Object controlling path evaluation and output. Currently only one member is supported.

`args.resultType ("VALUE"|"PATH")`:
causes the result to be either matching values *(default)* or normalized path expressions.

return value:

`(array|false)`:
Array holding either values or normalized path expressions matching the input path expression, which can be used for lazy evaluation. `false` in case of no match.

Javascript Example:

```
var o = { /*...*/ },  // the 'store' JSON object from above
    res1 = jsonPath(o, "$..author").toJSONString(),
    res2 = jsonPath(o, "$..author", {resultType:"PATH"}).toJSONString();
```

PHP example:

We need here to convert the JSON string to a PHP array first. I am using Michal Migurski's JSON parser for that.

```
require_once('json.php');      // JSON parser
require_once('jsonpath.php');  // JSONPath evaluator

$json = '{ ... }';  // JSON structure from above

$parser = new Services_JSON(SERVICES_JSON_LOOSE_TYPE);
$o = $parser->decode($json);
$match1 = jsonPath($o, "$..author");
$match2 = jsonPath($o, "$..author", array("resultType" => "PATH"));
$res1 = $parser->encode($match1);
$res2 = $parser->encode($match2);
```

results

Both *Javascript* and *PHP* example result in the following JSON arrays (as strings):

```
res1:
[ "Nigel Rees",
  "Evelyn Waugh",
  "Herman Melville",
  "J. R. R. Tolkien"
]
res2:
[ "$['store']['book'][0]['author']",
  "$['store']['book'][1]['author']",
  "$['store']['book'][2]['author']",
  "$['store']['book'][3]['author']"
]
```

Please note, that the return value of `jsonPath` is an array, which is also a valid JSON structure. So you might want to apply `jsonPath` to the resulting structure again or use one of your favorite array methods as `sort` with it.

# Issues

- Currently only single quotes allowed inside of JSONPath expressions.
- Script expressions inside of JSONPath locations are currently not recursively evaluated by `jsonPath`. Only the global `$` and local `@` symbols are expanded by a simple regular expression.

- An alternative for `jsonPath` to return `false` in case of *no match* may be to return an empty array in future.