JSONata

Overview

JSONata is a lightweight query and transformation language for JSON data, to extract the values you're interested in, returning them in JSON format. It's inspired by the location path semantics of XPath.
Here we're just going to do a brief overview of what you can do if you want to use this query language: since the enormous offer, we're unfortunately not able to put here everything, so for further information look at the official reference (at the end of this document).

Query composition

In JSONata everything is an expression and an expression is composed by

- values
- functions
- operators

Functions and operators are applied to values which themselves can be the results of evaluating sub-expressions. The language therefore is fully composable to any level of complexity.

Syntax

In order to efficiently support the data research in a JSON structure, JSONata is based on a location path syntax. The two structural constructs of JSON are objects and arrays.

JSON Objects

The location path syntax to navigate into JSON objects comprises the field names separated by dot ('.') delimiters. Hence, you navigate through objects in the most natural way (thinking about how you usually access these properties). Furthermore, you can also access them in another pretty common way: using square brackets [`property`].
The expression returns the JSON value referenced after navigating to the last step in the location path. If during the navigation of the location path, a field is not found, then the expression returns nothing (represented by `undefined`). No errors are thrown as a result of non-existing data in the input document.

For example:

- `Pokemon.nickname` we are accessing the `nickname` property of the `Pokemon` object
- `$.Team.moves.name` prints out ALL the names of ALL the moves of each pokemon in the Team object. Do not confuse the latter, with the following:
  `$.Team.moves[name]` here we are accessing the property moves (within the objects of the array `Team`) which have a property name.

- `$.Team.ciao` just return `undefined`, because there are no `ciao` properties within Team

Note: Putting $ in front of the query express the fact you're referring to the whole input document

JSON Arrays

JSON arrays are used when an ordered collection of values is required. Each value in the array is associated with an index (position) rather than a name, so in order to address individual values in an array, extra syntax is required to specify the index. This is done using square brackets after the field name of the array.
If the square brackets contains a number or an expression that evaluates to a number, then the number represents the index of the value to select. Indexes are zero offset, i.e. the first value in an array `arr` is `arr[0]`. If the number is not an integer, then it is rounded *down* to an integer. If the expression in square brackets is non-numeric, or is an expression that doesn't evaluate to a number, then it is treated as a predicate.

Negative indexes count from the end of the array, for example, `arr[-1]` will select the last value, `arr[-2]` the second to last, etc. If an index is specified that exceeds the size of the array, then nothing is selected.

If no index is specified for an array (i.e. no square brackets after the field reference), then the whole array is selected. If the array contains objects, and the location path selects fields within these objects, then each object within the array will be queried for selection.

For example:

- `$.Team[1]` return the Pokemon in the second position in the Team array of pokemon
- `$.Team[-2]` return the Pokemon in the second to last position in the Team array of pokemon
- `$.Team`, `$.Team[]` return the full Pokemon array in Team

Predicates

A set of items can be filtered using a predicate [expr] where expr evaluates to a boolean value.
The expression is related to the current item being tested, so if the predicate expression performs a navigation then it's relative to this context item.

For example:

- `Team[species='BULBASAUR']` selects all the BULBASAUR pokemons within the Team
- `$.Team[nickname='Sudoslim'].types` selects the types of the pokemons with nickname 'Sudoslim'
- `$.Team['GRASS' in types]` selects the pokemons of type GRASS

Syntax elements

Element	Description
`$`	At the start of an expression refers to the entire input document
`.`	Allows to navigate JSON objects
`moves[n]`	Access the n-th element of the moves array
`*`	Selects all fields in an object (e.g. `Team[0].*`)
`**`	Traverses all descendants (e.g. `$.**.moves` accesses directly to moves).

Functions and expressions

Numeric expressions

Numbers can be combined using the usual mathematical operators to produce a resulting number. The supported operators are the following:

Expression	Description
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Remainder (modulo)

Comparison expressions

Expression	Description
`=`	equals to
`!=`	not equals
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`in`	value is contained in array

Boolean expressions

Expression	**Description**
`and`	logical and
`or`	logical or

Note: not is supported as a function, not an operator, so you should just write `$not(arg)`

String expressions & functions

Expression	**Description**
`&`	Allows to concatenate strings (e.g. `Pi & ka & chu` = “Pikachu”). Note: it allows also to directly cast the operands to strings (e.g. `5&0&true`=50true)
Function	**Description**
`$substring(start, end[, length])`	Returns a string containing the characters in the first parameter `str` starting at position `start` (zero-offset). If `str` is not specified, then the context value is used as the value of `str`. If `length` is specified, then the substring will contain maximum `length` characters. If `start` is negative then it indicates the number of characters from the end of `str`
`$length(str)`	Returns the number of characters in the string `str`. If `str` is not specified (i.e. this function is invoked with no arguments), then the context value is used as the value of `str`. An error is thrown if `str` is not a string.
`$contains(str, pattern)`	Returns `true` if `str` is matched by `pattern`, otherwise it returns `false`. If `str` is not specified (i.e. this function is invoked with one argument), then the context value is used as the value of `str`. The `pattern` parameter can either be a string or a regular expression (regex). If it is a string, the function returns `true` if the characters within `pattern` are contained contiguously within `str`. If it is a regex, the function will return `true` if the regex matches the contents of `str`.

Array functions

Function	**Description**
`$count(array)`	Returns the number of items in the `array` parameter. If the `array` parameter is not an array, but rather a value of another JSON type, then the parameter is treated as a singleton array containing that value, and this function returns 1. If `array` is not specified, then the context value is used as the value of `array`.
`$append(array1, array2)`	Returns an array containing the values in `array1` followed by the values in `array2`. If either parameter is not an array, then it is treated as a singleton array containing that value.
`$sum(array)`	Returns the arithmetic sum of an array of numbers. It is an error if the input array contains an item which isn't a number.

`$max(array) /`
`$min(array) /`
`$average(array)`
	Returns the maximum / minimum / average number in an array of numbers. It is an error if the input array contains an item which isn't a number.

Date time functions

Function	**Description**
`$now()`	Returns the timestamp in an ISO 8601 formatted string (e.g. "2019-10-26T20:47:36.048Z")
`$millis()`	Returns the same timestamp as the number of milliseconds since midnight on 1st January 1970 UTC (e.g. 1572122951718)

References

JSONata offers you a heterogenous bunch of tools, so I strongly suggest you to go more into deep if you really need it. Just to make it clear: you can even define your own functions. JSONata is in fact a complete declarative functional language!
If I've made you a little bit more interested in the subject, take a look at the JSONata documentation here http://docs.jsonata.org/overview.