# Checking out JSON.Search function in MBS FileMaker Plugin

JMESPath (pronounced "jay-mess-path") is a query language and a specification for searching and extracting data from JSON documents. It provides a way to perform complex queries and transformations on JSON data, making it easier to work with structured data within JSON objects. JMESPath is similar in concept to other query languages like SQL for databases or XPath for XML documents, but it's specifically designed for JSON.

We add JMESPath to our MBS FileMaker Plugin for the upcoming version 13.5 with the JSON.Search function. You may use this function to find things and then also check JSON.Replace to replace values you selected with an expression.

Here are some key features and concepts of JMESPath:

1. **Path Expressions**:
   JMESPath uses path expressions to navigate and query JSON data. These expressions are a sequence of identifiers and operators that specify a path within the JSON structure to locate data.
2. **Wildcards**:
   JMESPath allows you to use wildcards like `*` to match all elements within an array or object, and `**` to search recursively through nested structures.
3. **Filters**:
   You can use filter expressions within JMESPath to select elements that meet specific conditions. For example, you can filter a list of objects to only include those with a certain property or value.
4. **Functions**:
   JMESPath includes a variety of functions that can be used to manipulate and transform data during queries. Functions like `map()`, `join()`, and `slice()` can be used to process data.
5. **Projection**:
   JMESPath supports projection, which means you can specify the shape of the output data. This allows you to return only the specific attributes or elements you're interested in, creating a more focused result.
6. **Piping**:
   JMESPath supports the pipe operator (`|`) to chain multiple operations together, allowing you to create complex queries by combining filters, projections, and functions.

Here's an example JMESPath query to illustrate some of these concepts. Suppose you have a JSON document representing a list of people, each with a "name", "city" and "age" property:

```
{
  "people": [
    {
      "name": "Alice",
      "age": 30,
      "city": "New York"
    },
    {
      "name": "Bob",
      "age": 25,
      "city": "San Francisco"
    },
    {
      "name": "Charlie",
      "age": 35,
      "city": "Los Angeles"
    }
  ]
}
```

You can use JMESPath to extract the names of people who are older than 30:

```
people[?age > `30`].name
```

The result of this query would be:

```
["Charlie"]
```

Let's try a bigger example with filter, projection and piping.

You want to filter the people who are older than 26, then project their names and cities, and finally sort the result by name. You can do this with the following JMESPath query that uses piping:

```
people[?age > `26`].{Name: name, City: city} | sort_by(@, &Name)
```

This query breaks down as follows:

- **people[?age > `26`]**:
  This part filters the "people" array to include only objects where the "age" is greater than 26.

- **[{Name: name, City: city}]**:
  Next, the result is piped into a projection operation that constructs new objects with "Name" and "City" properties, renaming the "name" and "city" properties.

- **sort_by(@, &Name)**:
  Finally the new records are sorted by the "Name" attribute in ascending order (`&name`).

The result of this query would be:

```
[
    {
        "Name":  "Alice",
        "City":  "New York"
    },
    {
        "Name":  "Charlie",
        "City":  "Los Angeles"
    }
]
```

In this result, two people are included, and their name and city are projected, sorted by name. This showcases the use of piping in JMESPath for complex data transformation operations, especially when you need to change the key names for JSON objects.

JMESPath is widely used in various contexts, especially when working with APIs that return data in JSON format. It provides a standardized way to access and manipulate JSON data, making it valuable for data analysis, transformation, and integration tasks. JMESPath libraries and tools are available for many programming languages to make it easier to use JMESPath in your applications.

Please let us know if you have questions.