TM Forum Specification

TMF630 REST API Design Guidelines Part 6

JSON path extensions

TMF630

Team Approved Date: Nov. 25th, 2020

Release Status: Pre-Production	**Approval Status:** Team Approved
Version: 4.0.0	**IPR Mode:** RAND

NOTICE

Direct inquiries to the TM Forum office:

181 New Road, Suite 304
Parsippany, NJ 07054, USA
Tel No. +1 973 944 5100
Fax No. +1 973 998 7916
TM Forum Web Page: www.tmforum.org

Table of Contents

Executive Summary . . . 1
- Conventions . . . 1

JSON Path extension . . . 2
- Introduction . . . 2
- Overview collection filtering using JSON Path (“filter”) . . . 6
- Overview resource partial representation using JSON Path (“*fields*”) . . . 7
- JSON Path . . . 7
- URL / URI Encoding of the JSONPath expressions . . . 29
- Collection filtering using JSONPath . . . 30
- Partial resource representation using JSONPath . . . 36
- Sorting selector . . . 39
- Notification Pattern - Registering Listener . . . 40
- Error handling . . . 41
- JSON Patch . . . 42

Administrative Appendix . . . 45
- Source Artefacts . . . 45
- Referenced Artefacts . . . 45
- Document History . . . 46
- Acknowledgments . . . 46

Executive Summary

This document provides information for the development of TMForum OpenAPIs using REST.

It provides recommendations and guidelines, to extend the existing design patterns through the usage of JSON Path across the TMForum OpenAPI REST ecosystem.

Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

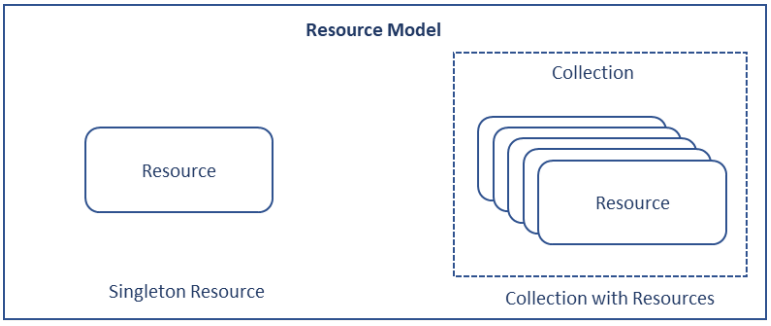
JSON Path extension

Introduction

The fundamental concept in any RESTful API is the *resource*. A resource is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

Resources can be grouped into *collections*. Each collection is homogeneous so that it contains only one type of resource, and unordered. Collections are themselves resources as well.

The diagram below illustrates the key concepts in a RESTful API.



[_Toc34901090 .anchor]**#Figure 1: Resource Model**

Resources have data associated with them which are represented using JSON.

JavaScript Object Notation (JSON) [RFC4627] is one of the most common REST API specification formats for the exchange and storage of structured data. TMForum OpenAPI Design Guidelines (DG) enforce the use and support of the JSON media type across the REST APIs as:

- _The server MUST support “application/json” by default. – ([DG4-1] page 12) _
- *REST APIs MUST support the “application/json” media type by default. – ([DG4-1] page 22)*

Considering the resource model, the main use cases are:

- Retrieving a sub-set of resources from a collection based on a set of JSONPath conditions, via the **Filter** directive.
- Retrieving a partial representation from a singleton resource, via the **Fields** directive.

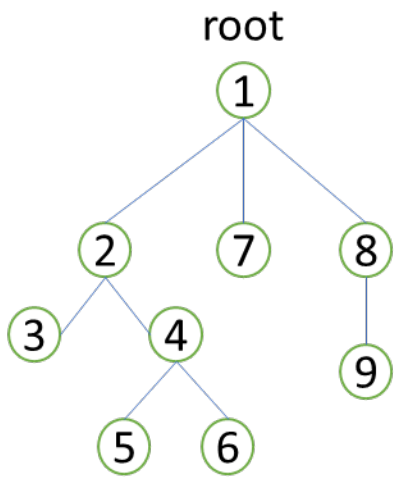
The TMForum OpenAPI DG provides support for querying resources with (or without) attribute filtering via the design patterns. This document aims to extend by augmenting the existing design patterns with the JSON Path query language features.

At the time of writing, JSON Path can be considered a de-facto standard for querying JSON documents due to the wide usage across the community. Numerous libraries are already available for a wide range of programming languages allowing the developers easy access to the features of the query language.

The adoption of JSON Path in the TMForum OpenAPI Design Guidelines is motivated by the desire to fulfill the query gaps in hierarchical multilevel array structures and to provide new query & filtering capabilities, standardized by a common syntax and behavior model.

One scenario where the collection filtering using JSON Path is required is in the scenario of the hierarchical query. The hierarchical query is heavily used across different TMForum openAPIs.

In the case of the hierarchical query, each node needs to be evaluated and the conditions to be satisfied.



[_Toc34901091 .anchor]**#Figure 2: Simple graph**

* Select each the root node of the hierarchy * Select each root node child. For each child check if the path and condition are satisfied * Select successive child nodes and again for all check paths and conditions. * If all conditions are satisfied the resource is selected.

As JSON is a hierarchical structure, let's consider a simple example to demonstrate the limitations of the standard key-value query and the solution proposed through the usage of JSONPath for collection filtering.

Considering the following simple example, for a "Building" collection, where each building has a set of attributes (lift condition) and an array for apartments where the number of rooms are listed.

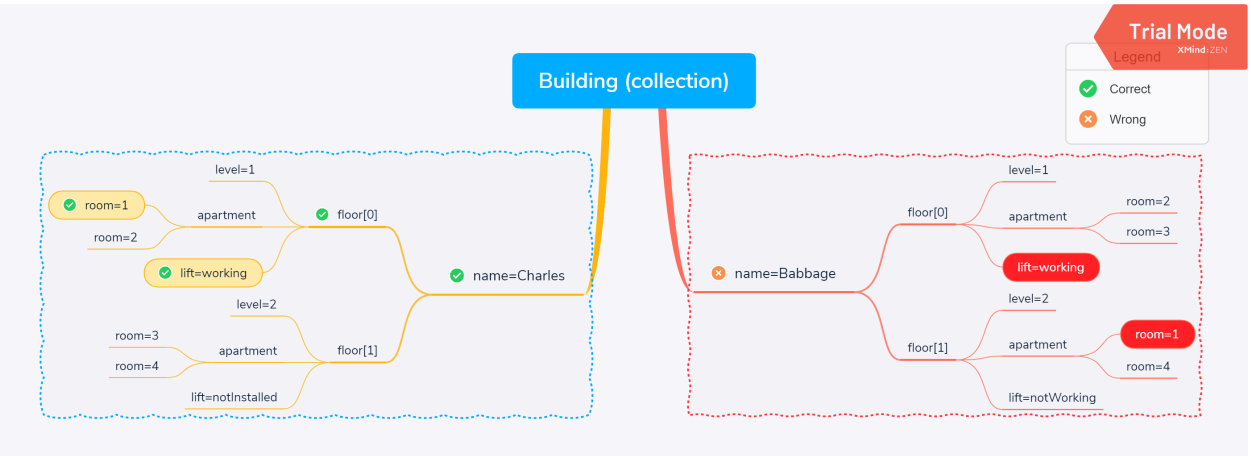
```
{
    "building":[{
            "name":"Babbage",
            "floor": [{
                    "level":1,
                    "apartment":[ {"rooms":2}, {"rooms":3} ],
"lift":"working"
                },
                {
                    "level":2,
                    "apartment":[ {"rooms":1}, {"rooms":4}],
                    "lift":"notworking"
                }]
        },
        {
            "name":"Charles",
            "floor":[{
                    "level":1,
                    "lift":"notinstalled",
                    "apartment":[ {"rooms":1}, {"rooms":2} ]
                },
                {
                    "level":2,
                    "lift":"working",
                    "apartment":[ {"rooms":1}, {"rooms":4}]
                }
            ]}
        ]}
}
```

If the request is to select all buildings that have a lift in working condition and there is an apartment with a single room, the query string will look like this:

```
GET /api/building?building.floor.lift=working&building.floor.apartment.rooms=1
```

However, the above query will produce the wrong result, returning both “Babbage” and “Charles” resources, because there is no mechanism in the query string to highlight that there should be an intersection between the two input conditions.

The graphical representation of the above query selection:



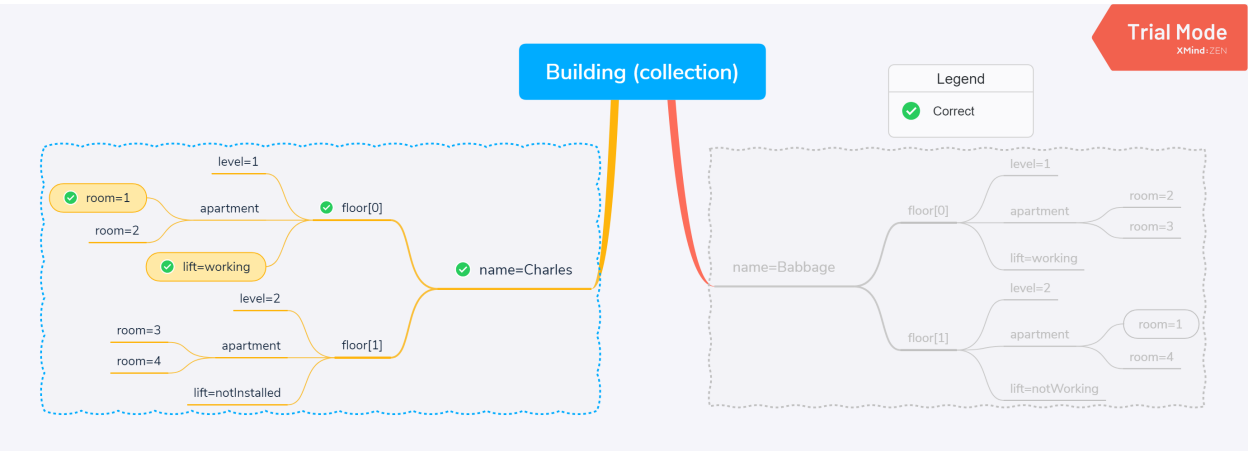
[_Toc34901092 .anchor]#**Figure 3: Collection filtering through simple query selection result**

As seen above, the resource “Babbage” will be selected when the query should only return the “Charles” resource.

The same JSON resource as above, this time using the collection filtering with JSONPath:

```
GET
/api/building?filter=$.building[*].floor[?(@.lift=="working")].apartment[?(@.rooms==1)]
```

will result only in the “Charles” resource to be selected and returned to the client.



[*Toc34901093 .anchor]* # **Figure 4: Collection filtering through _filter** selector result

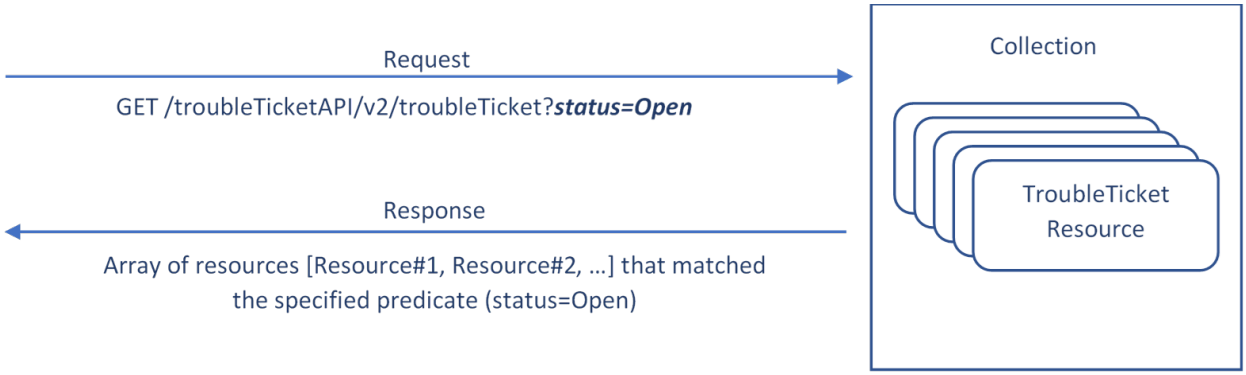
Note: for clarity in the above example the characters [and] are not encoded as they should

Overview collection filtering using JSON Path ("filter")

As per RESTful APIs guideline ([DG4-1]– page 21), GET HTTP operation is being used to retrieve either all the resources in a collection (subject to server limitations, pagination, etc.) or to retrieve only a subset of the resources from the collection based on condition(s) filtering.

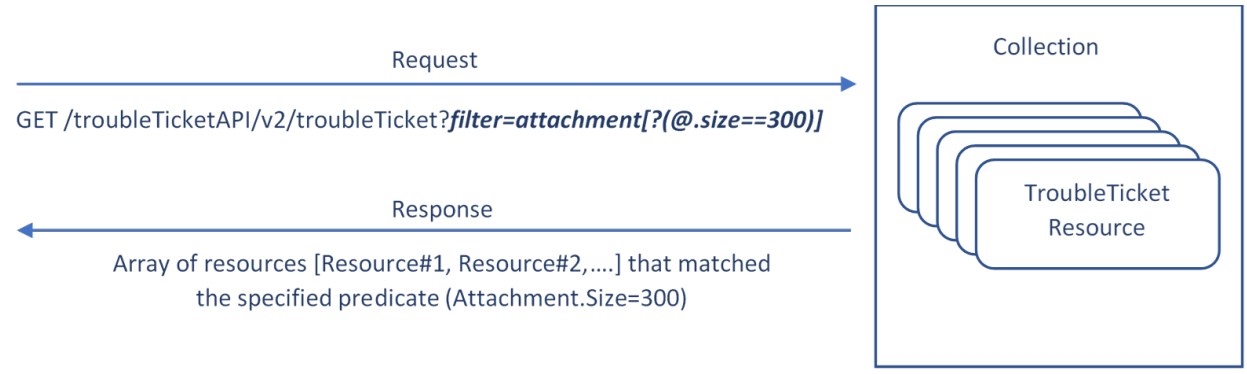
Collection filtering is done by taking the input condition (Boolean) and applying it against the collection of resources. By doing so, only the resources where the condition is satisfied (true) are selected and then returned to the API client.

Other parts of the TMForum Design Guidelines support only the mechanism of basic filtering which is based on using name-value query parameters on entity attributes (as described in [DG4-1] – Page 36)



[_Toc34901094 .anchor]**#Figure 5: Example of a basic query mechanism**

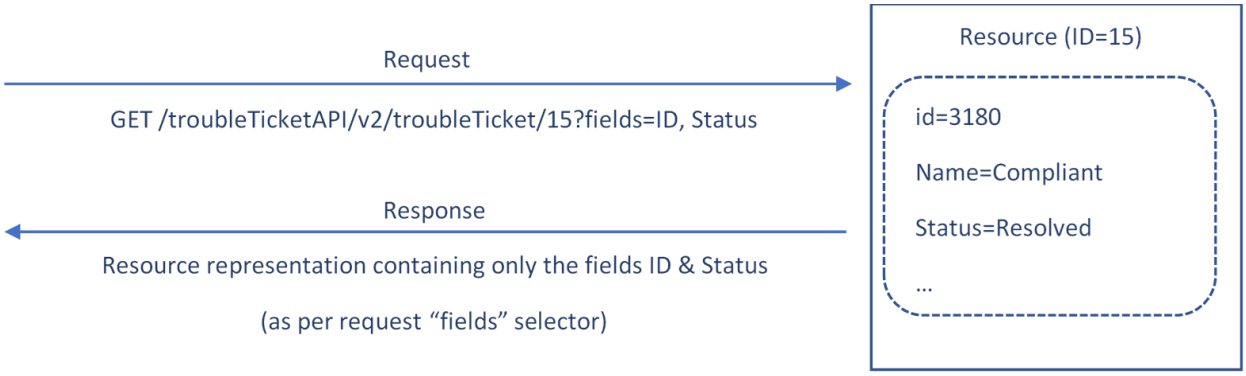
This part of the design guide enhances the collection filtering mechanism through the use of a JSONPath selector called "filter" to provide a flexible way to select a subset of resources from a collection by using the JSONPath predicate selectors and functions. The JSONPath collection filtering provides a standardized way of navigating through complex multilevel resources data models.



[_Toc34901095 .anchor]**#Figure 6: Example of a filter query mechanism**

Overview resource partial representation using JSON Path ("*fields*")

This pattern describes how to retrieve a subset of the attributes from an entity in the response using an attribute filtering mechanism. In the current [DG4-1](page 34) the filtering is done using the selector directive called "fields" - [DG4-1](page 34).



[_Toc34901096 .anchor]#**Figure 7: Example of a partial response mechanism**

The existing "fields" selector is enhanced to support a JSON Path expression so that the API consumers have more flexibility in selecting parts of the resources that are in a multi-level array structure.

The "."(dot) notation, specified in the [DG4-1], is the same notation used by JSON Path so the transition to the JSONPath expression will be transparent.

JSON Path

What is it

In the XML world, the XPath (XML Path Language), standardized by the W3C provides the ability to select and extract data out of XML documents. The equivalent of XPath for JSON documents is called JSON Path. It is a query-oriented language that allows querying and extraction of subsection(s) of a JSON document.

Every JSON document is based on a tree hierarchy of nodes (leaves), where every node is a JSON element which can be a simple leaf or complex one. The JSON Path language navigates this tree representation, selecting nodes through filtering criteria.

Considering the following simplified version of a TMForum TroubleTicket JSON as an example, that will be used as a reference throughout this document:

```
{
    "id": "3180",

"href":"https://host:port/troubleTicket/v2/troubleTicket/3180[[.underline]#https:
```

```
//host:port/troubleTicket/v2/troubleTicket/3180#]",
    "name": "Complaint over last bill",
    "status": "Resolved",
    "relatedEntity": [{
            "id": "3472",
            "href":
"https://host:port/customerBillManagement/v2/customerBill/3472[[.underline]#https
://host:port/customerBillManagement/v2/customerBill/3472#]",
            "name": "November Bill",
            "@referredType": "CustomerBill"
        },
        {
            "id": "3473",
            "href":
"https://host:port/customerBillManagement/v2/customerBill/3473[[.underline]#https
://host:port/customerBillManagement/v2/customerBill/3473#]",
            "name": "December Bill",
            "@referredType": "CustomerBill"
        }],
    "statusChange": [{
            "status": "Pending",
            "changeReason": "Need more information from the customer",
            "changeDate": "2018-05-01T00:00"
        },
        {
            "status": "InProgress",
            "changeReason": "Working on the issue resolution",
            "changeDate": "2018-05-02T00:00"
        },
        {
            "status": "Resolved",
            "changeReason": "Issue has been resolved",
            "changeDate": "2018-05-02T00:00"
        }],
    "note": [{
            "id": "1",
            "date": "2018-05-01T00:00",
            "author": "Mr John Wils",
            "text": "Missing necessary information from the customer"
            },
            {
            "id": "2",
            "date": "2018-05-01T00:00",
            "author": "Mr Erika Xavy",
            "text": "Information has been received, we're working on the
```

```
resolution"
            },
            {
            "id": "3",
            "date": "2018-05-02T00:00",
            "author": "Mr Redfin Tekram",
            "text": "Issue has been resolved, the service has been restored"
            }],
    "attachment": [{
            "description": "Scanned disputed December bill",
            "href":
"http://hostname:port/documentManagement/v2/attachment/44[[.underline]#http://hos
tname:port/documentManagement/v2/attachment/44#]",
            "id": "44",
            "url": "http://xxxxx[[.underline]#http://xxxxx#]",
            "name": "December Bill",
            "size": 300,
            "sizeUnit": "KB",
            "@referredType": "Attachment"
        },
        {
            "description": "Scanned disputed November bill",
            "href":
"http://hostname:port/documentManagement/v2/attachment/45[[.underline]#http://hos
tname:port/documentManagement/v2/attachment/45#]",
            "id": "45",
            "url": "http://xxxxx[[.underline]#http://xxxxx#]",
            "name": "November Bill ",
            "size": 500,
            "sizeUnit": "KB",
            "@referredType": "Attachment"
        }],
    "channel": {
        "id": "8774",
        "name": "Self Service",
        "@type": "Channel"
    }
}
```

Note: All further examples are relative to the TroubleTicket entity presented in the JSON representation above.

A JSON document doesn't necessarily have a dedicated element to represent the "root" of the structure but in the JSONPath case, a special notation has been introduced to represent the top-level

element/root node.

The root node contains all the other JSON elements and based on our example some of them are simple nodes (id, href, name) and non-leaf ones - containing other nodes like (relatedEntity, statusChange, note).

JSONPath provides a uniform syntax (further details in the "syntax" chapter to define expressions that can traverse a JSON document to extract the relevant subsections. For example, to retrieve the value for the "name" node, the following simple JSONPath query expression is used: "$.name" ("$" represents the root element). Much more complex paths and expressions can be used, that will allow traversal of the entire document looking for an indicated field, or to filter based on the field's values.

Syntax

JSONPath makes use of special notation and syntax to represent the nodes and links between them.

The top-level element of the JSON document represents the "root" element and is notated with the dollar sign "$".

The following notation styles are currently supported:

- dot-notation - e.g. $.name
- bracket-notation - e.g. $['name']

Note: The leading $ represents the root object or array and can be omitted

Both of the above paths refer to the same node from the above JSON document example, the "name" field which is the child of the root element.

Operators

The following operators and expressions are described below:

Expression & Operators	Description
$	Root object/element to query. This is how all the path expressions are started
@	The current node being processed - it is used in the input expressions for filter predicates.
.<attribute>	Child selector - selects the specified property in a parent object using the "dot" (.) notation

Expression & Operators	Description
['<attribute>' (, '<attribute>')]	Child or children selector - selects the specified property or properties in a parent object using the "bracket" ([) notation Note: This expression SHOULD be used if the attribute contains special characters such as spaces, or begins with a character other than A..Za..z_.
..	Recursive descent. JSONPath borrows this syntax from E4X. It will search for the specified attribute name recursively and it will return an array of all values with the attribute name. It will always return a list, even if only one property is found.
*	Wildcard selector. It will select all elements (object or array) regardless of their names or indexes. For example, $.relatedEntity[0][**] means all attributes of the first object from the relatedEntity array, and $.relatedEntity[**] means all items of the relatedEntity array.
[n]	Selects the n^{th} element from an array. Indexes are 0-based.
[indexNr, indexNr, …]	Selects array elements with the specified indexes. Returns a list.
[start:end] [start:]	Array slice operator. JSONPath borrows this from E4X. It will select array elements from the start index and up to, but not including, end index. If the end is omitted, selects all elements from start until the end of the array. Returns a list.

Expression & Operators	Description
[:n]	Selects the first n elements of the array. Returns a list
[-n:]	Selects the last n elements of the array. Returns a list.
[,]	Union operator. JSONPath allows alternate names or array indices as a set.
[?(expression)]	Filter expression. Selects all elements in an object or array that match the specified filter.
[(expression)]	Script expressions can be used instead of explicit property names or indexes. An example is [(@.length-1)] which selects the last item in an array. Here, length refers to the length of the current array rather than a JSON field named length.

Examples:

JSON Path	Result
$.channel	All the properties of the channel Output: `{ "id" : "8774", "name" : "Self Service", "@type" : "Channel" }`

JSON Path	Result
$['id','name','href']	Select multiple children from the root element. This operator is very useful in the partial response scenario. Output: `{ "id" : "3180", "name" : "Complaint over last bill", "href" : "https://host:port/troubleTicket/v2/troubleTicket/3180[[.underline]#https://host:port/troubleTicket/v2/troubleTicket/3180#]" }`
$..name	Select all the 'name' attributes across all the resource structure. Output: `{empty}[ "Complaint over last bill", "November Bill", "December Bill", "December Bill", "November Bill ", "Self Service" ]`

JSON Path	Result
$.note[*].author	All the authors attribute values from all the elements of the 'note' array Output: {empty}["Mr John Wils", "Mr Erika Xavy", "Mr Redfin Tekram"]

JSON Path	Result

$.note[*]	Wildcard select for all the elements within the note array
	
	Output:
	
	`----`
	`{empty}[{`
	`"id" : "1",`
	`"date" : "2018-05-01T00:00",`
	`"author" : "Mr John Wils",`
	`"text" : "Missing necessary information from the customer"`
	`},`
	`{`
	`"id" : "2",`
	`"date" : "2018-05-01T00:00",`
	`"author" : "Mr Erika Xavy",`
	`"text" : "Information has been received, we're working on the resolution"`
	`},`
	`{`
	`"id" : "3",`
	`"date" : "2018-05-02T00:00",`
	`"author" : "Mr Redfin Tekram",`
	`"text" : "Issue has been resolved, the service has been restored"`
	`}`
	`]`
	`----`

JSON Path	Result
$.note[1]	Retrieve the first element of the note array Output: `{ "id" : "2", "date" : "2018-05-01T00:00", "author" : "Mr Erika Xavy", "text" : "Information has been received, we're working on the resolution" }`
$.note[0,1]	Retrieve the elements from the array positions 0 and 1 (remember indexes start from 0) Output: `{empty}[ { "id" : "1", "date" : "2018-05-01T00:00", "author" : "Mr John Wils", "text" : "Missing necessary information from the customer" }, { "id" : "2", "date" : "2018-05-01T00:00", "author" : "Mr Erika Xavy", "text" : "Information has been received, we're working on the resolution" } ]`

JSON Path	Result
$.note[1:]	Retrieve the elements from the array starting from position 1 till the end Output: {empty}[{ "id" : "2", "date" : "2018-05-01T00:00", "author" : "Mr Erika Xavy", "text" : "Information has been received, we're working on the resolution" }, { "id" : "3", "date" : "2018-05-02T00:00", "author" : "Mr Redfin Tekram", "text" : "Issue has been resolved, the service has been restored" }]

JSON Path	Result
$.note[:2]	Selects the first 2 elements of the array. Output: {empty}[{ "id" : "1", "date" : "2018-05-01T00:00", "author" : "Mr John Wils", "text" : "Missing necessary information from the customer" }, { "id" : "2", "date" : "2018-05-01T00:00", "author" : "Mr Erika Xavy", "text" : "Information has been received, we're working on the resolution" }]

JSON Path	Result
$.note[-2:]	Selects the last 2 elements of the array. Output: {empty}[{ "id" : "2", "date" : "2018-05-01T00:00", "author" : "Mr Erika Xavy", "text" : "Information has been received, we're working on the resolution" }, { "id" : "3", "date" : "2018-05-02T00:00", "author" : "Mr Redfin Tekram", "text" : "Issue has been resolved, the service has been restored" }]

Functions

Several implementations of JSONPath support Functions that can be invoked at the tail end of a path. The input to a function is the output of the path expression.

The function output is dictated by the function itself.

Function	Description	Output
min()	Provides the min value of an array of numbers	Double
max()	Provides the max value of an array of numbers	Double
avg()	Provides the average value of an array of numbers	Double
stddev()	Provides the standard deviation value of an array of numbers	Double

Function	Description	Output
length()	Provides the length of an array	Integer

Note: Some of the above function capabilities are not currently supported by all implementations.

Considering the following example:

```
{
    "price": [
        1,
        2,
        3,
        4,
        5,
        6
    ]
}
```

JSON Path	Result
$.price.min()	Provides the min value of an array of numbers Output: 1.0
$.price.max()	Provides the max value of an array of numbers Output: 6.0
$.price.avg()	Provides the average value of an array of numbers Output: 3.5
$.price.stddev()	Provides the standard deviation value of an array of numbers Output: 1.707825127659933
$.price.length()	Provides the length of an array Output: 6

Filter predicates

JSONPath predicate filters are Boolean expressions (true or false) that restrict returned lists of nodes - usually applied to arrays. The filter syntax is:

```
[?(expression)]
```

The filter syntax is generally used in conjunction with the "@" operator to create the predicate.

A quick filter example could be:

Input:

```
statusChange[?(@.status=='Pending')]
```

Output:

```
{empty}[
{
    "status" : "Pending",
    "changeReason" : "Need more information from the customer",
    "changeDate" : "2018-05-01T00:00"
}]
```

The "@" represents the current item being processed.

If the predicate filter does not validate, an empty response will be returned.

The condition inside the predicate can also be based on the values outside the current object.

For example:

Input:

```
statusChange[?(@.status==$.status)]
```

Output:

```
{empty}[
{
    "status" : "Resolved",
    "changeReason" : "Issue has been resolved",
    "changeDate" : "2018-05-02T00:00"
}]
```

As expected, a filter that specifies only the property name, for example, $.statusChange[?(@.status)] will match and return all items that have this property defined, regardless of the value.

Additionally, filters support the following operators:

Operator	Description
==	Equals to. 1 and '1' are considered equal. String values should be enclosed in single quotes or double-quotes.
!=	Not equal to.
>	Greater than.
>=	Greater than or equal to.
<	Less than.
□	Less than or equal to.
=~	Match a JavaScript Regular Expression. For example, $.statusChange[?(@.status=~/Resol.*?/i)] matches items whose status starts with *Resol*. ***Note:*** There is a difference in library implementations on the behavior of this operator.

!	Use to negate a filter: $.attachment[?(!@.size)] matches items that do not have the size property.
&&	Logical AND, used to combine multiple filter expressions: $.attachment[?(@.size==300 && @.sizeUnit=='KB')]
||	Logical OR, used to combine multiple filter expressions: $.attachment[?(@.size==300
	@.size==500)]

Examples (based on the TroubleTicket resource):

JSON Path	**Result**
$.attachment[?(@.size==300)]	[{ "description" : "Scanned disputed December bill", "href" : "http://hostname:port/documentManagement/v2/attachment/44[[.underline]]#http://hostname:port/documentManagement/v2/attachment/44#]", "id" : "44", "url" : "http://xxxxx[[.underline]]#http://xxxxx#]", "name" : "December Bill", "size" : 300, "sizeUnit" : "KB", "@referredType" : "Attachment" }]

JSON Path	Result

$.statusChange[?(@.status!='Pending')]	[{
	"status" : "InProgress",
	"changeReason" : "Working on the issue resolution",
	"changeDate" : "2018-05-02T00:00"
	},
	{
	"status" : "Resolved",
	"changeReason" : "Issue has been resolved",
	"changeDate" : "2018-05-02T00:00"
	}]

$.attachment[?(@.size==300)]	[{ "description" : "Scanned disputed November bill", "href": "http://hostname:port/documentManagement/v2/attachment/45[[.underline]]#http://hostname:port/documentManagement/v2/attachment/45#", "id" : "45", "url" : "http://xxxxx[[.underline]]#http://xxxxx#]", "name" : "November Bill ", "size" : 500, "sizeUnit" : "KB", "@referredType" : "Attachment" }]

JSON Path	Result

$.attachment[?(@.size>=300)]	[{
	"description" : "Scanned disputed December bill",
	"href": "http://hostname:port/documentManagement/v2/attachment/44[[.underline]]#http://hostname:port/documentManagement/v2/attachment/44#]",
	"id" : "44",
	"url" : "http://xxxxx[[.underline]]#http://xxxxx#]",
	"name" : "December Bill",
	"size" : 300,
	"sizeUnit" : "KB",
	"@referredType" : "Attachment"
	},
	{
	"description" : "Scanned disputed November bill",
	"href": "http://hostname:port/documentManagement/v2/attachment/45[[.underline]]#http://hostname:port/documentManagement/v2/attachment/45#",
	"id" : "45",
	"url" : "http://xxxxx[[.underline]]#http://xxxxx#]",
	"name" : "November Bill ",
	"size" : 500,
	"sizeUnit" : "KB",
	"@referredType" : "Attachment"
	}]

JSON Path	Result
$.attachment[?(@.size<301)]	[{ "description" : "Scanned disputed December bill", "href" :"http://hostname:port/documentManagement/v2/attachment/44[[.underline]]#http://hostname:port/documentManagement/v2/attachment/44#", "id" : "44", "url" : "http://xxxxx[[.underline]]#http://xxxxx#]", "name" : "December Bill", "size" : 300, "sizeUnit" : "KB", "@referredType" : "Attachment" }]
$.attachment[?(@.size□300)]	[{ "description" : "Scanned disputed December bill", "href" :"http://hostname:port/documentManagement/v2/attachment/44[[.underline]]#http://hostname:port/documentManagement/v2/attachment/44#", "id" : "44", "url" : "http://xxxxx[[.underline]]#http://xxxxx#]", "name" : "December Bill", "size" : 300, "sizeUnit" : "KB", "@referredType" : "Attachment" }]

JSON Path	Result

$.statusChange[?(@.status=~ /Resol.*?/i)]	<pre>{empty}[{
	"status" : "Resolved",
	"changeReason" : "Issue has been resolved",
	"changeDate" : "2018-05-02T00:00"
	}]</pre> *Note:* this works with Jayway library implementation of the JSON Path

$.attachment[?(!@.size)]	[] – Nothing is retrieved because all the elements in the attachment array have an attribute called "size".

$.attachment[?(@.size==300 && @.sizeUnit=='KB')]	<pre>[{
	"description" : "Scanned disputed December bill",
	"href": "http://hostname:port/documentManagement/v2/attachment/44[[.underline]]#http://hostname:port/documentManagement/v2/attachment/44#",
	"id" : "44",
	"url" : "http://xxxxx[[.underline]]#http://xxxxx#]",
	"name" : "December Bill",
	"size" : 300,
	"sizeUnit" : "KB",
	"@referredType" : "Attachment"
	}]</pre>

$.attachment[?(@.size==300	

JSON Path	Result

@.size==500)]	[{
	"description" : "Scanned disputed December bill",
	
	"href":"http://hostname:port/documentManagement/v2/attachment/44[[.underline]#http://hostname:port/documentManagement/v2/attachment/44#",
	"id" : "44",
	"url" : "http://xxxxx[[.underline]#http://xxxxx#]",
	"name" : "December Bill",
	"size" : 300,
	"sizeUnit" : "KB",
	"@referredType" : "Attachment"
	},
	{
	"description" : "Scanned disputed November bill",
	"href": "http://hostname:port/documentManagement/v2/attachment/45[[.underline]#http://hostname:port/documentManagement/v2/attachment/45#",
	"id" : "45",
	"url" : "http://xxxxx[[.underline]#http://xxxxx#]",
	"name" : "November Bill ",
	"size" : 500,
	"sizeUnit" : "KB",
	"@referredType" : "Attachment"
	}]

Notes:

- *JSONPath expressions, including property names and values, are case-sensitive.*
- Unlike XPath, JSONPath does not have operations for accessing parent or sibling nodes from the given node.

URL / URI Encoding of the JSONPath expressions

The JSONPath syntax makes use of a set of special characters like $, [,], (,), ?.

As the JSON Path will be passed as the value in one of the "*filter*" or "*fields*" selectors, special care must be taken in making sure the proper encoding is being used.

For this, the following arguments and standards have been taken into account:

- *RFC 3986*
  - *2.2. Reserved Characters*
  - *URIs include components and subcomponents that are delimited by characters in the "reserved" set. These characters are called "reserved" because they may (or may not) be defined as delimiters by the generic syntax, by each scheme-specific syntax, or by the implementation-specific syntax of a URI's dereferencing algorithm. If data for a URI component would conflict with a reserved character's purpose as a delimiter [emphasis added], then the conflicting data must be percent-encoded before the URI is formed.*
  - *reserved = gen-delims / sub-delims*
  - *gen-delims = ":" / "/" / "?" / "#" / "[" / "]" / "@"*
  - *sub-delims = "!" / "$" / "&" / "'" / "(" / ")"*
    / "" / "+" / "," / ";" / "="*
  - *3.3.Path Component*
    - *pchar = unreserved / pct-encoded / sub-delims / ":" / "@"*
  - *3.4 Query Component*
    - *query = *(pchar / "/" / "?")*
  - 3.2.2 Host
    - *A host identified by an Internet Protocol literal address, version 6 [RFC3513] or later, is distinguished by enclosing the IP literal within square brackets ("[" and "]"). This is the only place where square bracket characters are allowed in the URI syntax.*
  - *From appendix A of the same RFC:*
    - pchar = unreserved / pct-encoded / sub-delims / ":" / "@"

      […]

      pct-encoded = "%" HEXDIG HEXDIG

      unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"

      […]
      sub-delims = "!" / "$" / "&" / "'" / "(" / ")"

/ "*" / "+" / "," / ";" / "="

The TM Forum [DG4-1], already allows for the ";" and for the "," in the query segment of the URI to be used without being escaped.

As such only the “[“ and “]” characters will be encoded.

Decoded:

```
GET /troubleTicket/?filter=attachment[?(@.size==300)]
```

Encoded:

```
GET /troubleTicket/?filter=attachment%5B?(@.size==300)%5D
```

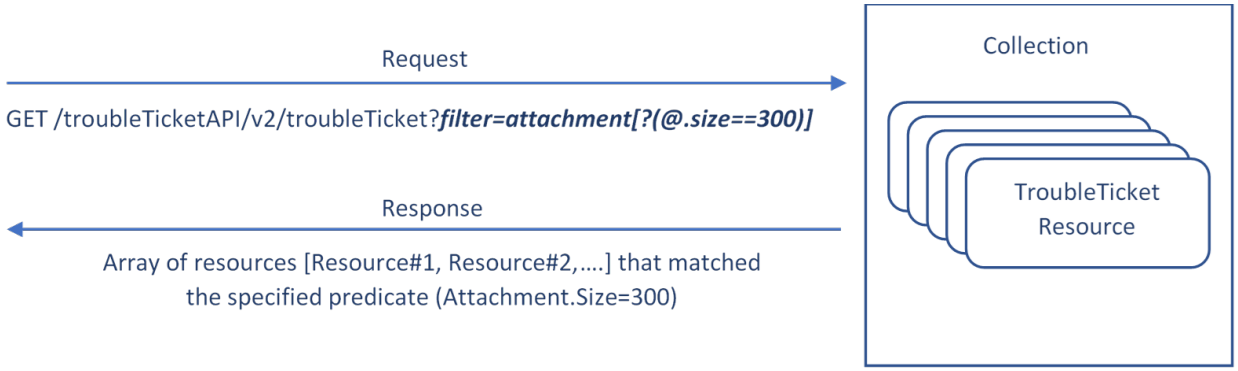
Note: For readability purpose the examples of JSON Path expression in the rest of the document don’t have the characters [and] in their encoded form.

Collection filtering using JSONPath

This pattern describes how to retrieve a subset of resources from a collection through an extended query filtering mechanism. In design guidelines Part 1 ([DG4-1]) the filtering is done using the name/value query parameters on the entity attributes (for further details, please see [DG4-1]- page 36).

This part of the design guide enhances the query design pattern by defining a JSONPath attribute(s) selector that will enhance the existing capabilities for the entity attributes that are arrays. The JSONPath predicate expression will allow the selection of all the resources that will match the predicate condition.

Note: All examples are relative to the management of the TroubleTicket entity having the same JSON representation as in the above chapter.



[_Toc34901097 .anchor]#**Figure 8: Example of the filtering mechanism**

The enhancement of the filtering mechanism is done by defining a *"filter"* query parameter that will act as a selector:

```
?filter=JSONPathExpression
```

Unlike the basic filtering, the *filter* selector takes as a parameter a JSON Path predicate expression. As long as the predicate condition validates for a particular resource it will be returned in the response to the client.

Rule: An attribute selector directive called "filter" MUST be used to specify the JSON Path expression.

Rule: Entity attribute selection via JSON Path expression MAY optionally be enabled.

Rule: In case the JSONPath "filter" selector is not supported a 501 Not Implemented MUST be returned

The filtering expression is a sequence of JSON Path predicates that will perform assertions at the resource's attribute level.

```
 GET \{apiRoot}/\{resourceName}/\{resourceID}/?filter=\{JSON Path
Expression}
```

For example:

```
GET \{apiRoot}/troubleTicket/?filter=attachment[?(@.size==300)]
```

Note: In the above example the leading "$." was omitted.

JSONPath syntax supports notations where the "$" (root element) is present or not. To maintain the compatibility with the simple "." dot notation, the recommendation is to omit the leading "$." characters.

Rule: In the JSON Path expression from the "filter" selector the leading "$" MAY be omitted.

The "*filter*" will select and return all the TroubleTicket resources that have in the "*attachment*" array, elements where the attribute "*size*" is equal to 300. The "*filter*" selector can also be combined with the standard query parameters forming a more complex query expression.

For example, the following query will return all the "TroubleTicket" resources that have *"status=resolved"* and that have in the "*attachment*" array, elements where the attribute "*size*" is equal to 300:

```
GET /troubleTicket/?status=resolved&filter=attachment[?(@.size==300)]
```

The complete resource representations (with all the attributes) of all the matching entities must be returned.

Rule: The complete resource representations (with all the attributes) of all the matching entities must be returned.

Rule: The returned representation of each entity must contain a field called « id» and that field must be populated with the resourceID.

Rule: If the request is successful then the returned code MUST be 200. The exceptions code must use the exception codes from http://www.iana.org/assignments/httpstatus-codes/http-status-codes.xml as explained in section 4.3.

Rule: If the JSONPath expression that is being passed in the “filter” selector is invalid a 400 Bad Request MUST be returned.

Rule: The subset obtained through the use of the “filter” selector is under the same pagination rules as described in the [DG4-1] (page 40)

For example:

GET /troubleTicket/?offset=10&limit=20&filter=attachment[?(@.size==300)]

will retrieve the twenty resources starting at the tenth where the *attachment* array has the element *size* equal with 300.

JSON Path filter ORING is also supported and is achieved following the equivalent pattern

```
[filter=\{JSON Path Expression},\{JSON Path Expression}*]
```

of the type described in the [DG4-1](page 36). ORING can be used many times as required.

For example:

```
GET /troubleTicket/?status=resolved&**filter=**attachment[?(@.sizeUnit=='KB' &&
@.size==500)],attachment[?(@.sizeUnit=='MB' && @.size==0.5)]
```

ORING can also be achieved by using ";" again as an equivalent pattern to the one described in the [DG4-1] (page36):

```
[filter=\{JSON Path Expression};filter=\{JSON Path Expression}*]
```

The default behavior is to return all resources where the JSONPath predicate is a match. As such simple filters like: [?(@)], which is return true for all the resource members will result in selecting all the resources in the collection.

The basic filtering can be combined with the “filter” selector. When both mechanisms are used, the result set must be a match for both conditions. For example:

Request:

```
GET /troubleTicket/?status=resolved&filter=attachment[?(@.sizeUnit=='KB' &&
@.size==500)]
```

Response:

```
200 Content-Type: application/json

[{
    "id": "3180",
    "href":
"https://host:port/troubleTicket/v2/troubleTicket/3180[[.underline]#https://host:
port/troubleTicket/v2/troubleTicket/3180#]",
    "name": "Complaint over last bill",
    "status": "resolved",
    "relatedEntity": [{
            "id": "3472",
            "href":
"https://host:port/customerBillManagement/v2/customerBill/3472[[.underline]#https
://host:port/customerBillManagement/v2/customerBill/3472#]",
            "name": "November Bill",
            "@referredType": "CustomerBill"
        },
        {
            "id": "3473",
            "href":
"https://host:port/customerBillManagement/v2/customerBill/3473[[.underline]#https
://host:port/customerBillManagement/v2/customerBill/3473#]",
            "name": "December Bill",
            "@referredType": "CustomerBill
        }],
    "statusChange": [{
```

```
            "status": "Pending",
            "changeReason": "Need more information from the customer",
            "changeDate": "2018-05-01T00:00"
        },
        {
            "status": "InProgress",
            "changeReason": "Working on the issue resolution",
            "changeDate": "2018-05-02T00:00"
        },
        {
            "status": "Resolved",
            "changeReason": "Issue has been resolved",
            "changeDate": "2018-05-02T00:00"
        }],
    "note": [{
            "id": "1",
            "date": "2018-05-01T00:00",
            "author": "Mr John Wils",
            "text": "Missing necessary information from the customer"
        },
        {
            "id": "2",
            "date": "2018-05-01T00:00",
            "author": "Mr Erika Xavy",
            "text": "Information has been received, we're working on the
resolution"
        },
        {
            "id": "3",
            "date": "2018-05-02T00:00",
            "author": "Mr Redfin Tekram",
            "text": "Issue has been resolved, the service has been restored"
        }],
    "attachment": [{
            "description": "Scanned disputed December bill",
            "href":
"http://hostname:port/documentManagement/v2/attachment/44[[.underline]#http://hos
tname:port/documentManagement/v2/attachment/44#]",
            "id": "44",
            "url": "http://xxxxx[[.underline]#http://xxxxx#]",
            "name": "December Bill",
            "size": 300,
            "sizeUnit": "KB",
            "@referredType": "Attachment"
        },
```

```
        {
            "description": "Scanned disputed November bill",
            "href":
"http://hostname:port/documentManagement/v2/attachment/45[[.underline]#http://hos
tname:port/documentManagement/v2/attachment/45#]",
            "id": "45",
            "url": "http://xxxxx[[.underline]#http://xxxxx#]",
            "name": "November Bill ",
            "size": 500,
            "sizeUnit": "KB",
            "@referredType": "Attachment"
        }],
    "channel": {
        "id": "8774",
        "name": "Self Service",
        "@type": "Channel"
    }
},
{
    "id": "3181",
    "href":
"https://host:port/troubleTicket/v2/troubleTicket/3181[[.underline]#https://host:
port/troubleTicket/v2/troubleTicket/3181#]",
    "name": "Complaint over last bill",
    "status": "Resolved",
    "relatedEntity": [{
            "id": "3473",
            "href":
"https://host:port/customerBillManagement/v2/customerBill/3472[[.underline]#https
://host:port/customerBillManagement/v2/customerBill/3472#]",
            "name": "November Bill",
            "@referredType": "CustomerBill"
            }],
    "statusChange": [{
            "status": "Pending",
            "changeReason": "Need more information from the customer",
            "changeDate": "2018-05-01T00:00"
            },
            {
            "status": "InProgress",
            "changeReason": "Working on the issue resolution",
            "changeDate": "2018-05-02T00:00"
            },
            {
            "status": "Resolved",
```

```
            "changeReason": "Issue has been resolved",
            "changeDate": "2018-05-02T00:00"
            }],
    "note": [{
            "id": "3",
            "date": "2018-05-02T00:00",
            "author": "Mr Redfin Tekram",
            "text": "Issue has been resolved, the service has been restored"
        }],
    "attachment": [{
            "description": "Scanned disputed December bill",
            "href":
"http://hostname:port/documentManagement/v2/attachment/44[[.underline]#http://hos
tname:port/documentManagement/v2/attachment/44#]",
            "id": "44",
            "url": "http://xxxxx[[.underline]#http://xxxxx#]",
            "name": "December Bill",
            "size": 300,
            "sizeUnit": "KB",
            "@referredType": "Attachment"
        }],
    "channel": {
        "id": "8774",
        "name": "Self Service",
        "@type": "Channel"
    }
}]
```


Partial resource representation using JSONPath

This pattern describes how to retrieve a subset of the attributes from an entity in the response using an attribute filtering mechanism. In the current [DG4-1](page 34) the filtering is done using the selector directive called “fields” - [DG4-1](page 34).

This part of the design guide enhances the "fields" selector to support a JSON Path (path or predicate).

***Note*:** All the rules described in [DG4-1](page 34) will continue to apply.

Rule: In order to retrieve a partial representation using a JSON Path, the “fields” selector MUST be used:

```
GET \{apiRoot} /\{resourceName}/\{resourceID}/?fields=\{JSONPath*}
```

For example:

The request is to return only the "name" attribute from the troubleTicket entity.

Note: as per [DG4-1] (page 35) the ID of the resource will always be returned.

Request:

```
GET /api/troubleTicket/42?fields=channel.name
```

Response:

```
200
Content-Type: application/json

{
    "id": "42",
    "channel": {
    "name": "Self Service"
}
```

Another example, to retrieve only the attributes from the "note" array that match a particular predicate condition (the author is Mr. John Wils)

Request:

```
GET /api/troubleTicket/42/?fields=note[?(@.author=='Mr John Wils')]
```

Response:

```
200

Content-Type: application/json
{
    "id" : "1",
    "date" : "2018-05-01T00:00",
    "author" : "Mr John Wils",
    "text" : "Missing necessary information from the customer"
}
```

Example to retrieve only specific JSON elements (simple or arrays):

Request:

```
GET /api/troubleTicket/42/?fields=['id','href','name','note']
```

Response:

```
200
Content-Type: application/json

{
    "id": "3180",
    "href":
"https://host:port/troubleTicket/v2/troubleTicket/3180[[.underline]#https://host:
port/troubleTicket/v2/troubleTicket/3180#]",
    "name" : "Complaint over last bill",
    "note" : [{
            "id" : "1",
            "date" : "2018-05-01T00:00",
            "author" : "Mr John Wils",
            "text" : "Missing necessary information from the customer"
        },
        {
            "id" : "2",
            "date" : "2018-05-01T00:00",
            "author" : "Mr Erika Xavy",
            "text" : "Information has been received, we're working on the
            resolution"
        },
        {
            "id" : "3",
            "date" : "2018-05-02T00:00",
            "author" : "Mr Redfin Tekram",
            "text" : "Issue has been resolved, the service has been restored"
        }]
}
```

JSON Path "fields" ORING is also supported and is achieved following the equivalent pattern:

```
[fields=\{JSON Path Expression},\{JSON Path Expression}*]
```

ORING can be used many times as required.

Example:

Request:

```
GET
/troubleTicket/?status=Resolved&fields=['id','href','name','note'],channel,note[?
(@.author=='Mr John Wils')]
```

Response:

```
200
Content-Type: application/json

{
    "id": "3180",
    "href": "https://host:port/troubleTicket/v2/troubleTicket/3180",
    "name": "Compliant over last bill",
    "channel": {
        "id": "8774",
        "name": "Self Service",
        "@type": "Channel"
    },
    "note": {
        "id": "1",
        "date": "2018-05-01T00:00",
        "author": "Mr John Wils",
        "text": "Missing necessary information from the customer"
    }
}
```

Sorting selector

This pattern describes how to request a particular sorting criterion. The sort directive is described in [DG4-1] (page 45). This part of the design guide enhances the "Sort-Field" selector to support a JSON Path.

The following specification is described in the [DG4-1]- Sorting directive:

Sort-Query-Parameters: *“sort”, “=”, (Sort-Direction), Sort-Field*

Sort-Direction: *“-“ | “+”*

Sort-Field: _The field to sort on. _

The above "Sort-Field" can take a JSON Path expression.

Example:

```
GET /api/troubleTicket?sort=channel.name
```

or

```
GET /api/troubleTicket?sort=attachment[*].name
```

The sorting selector can always be combined with “filter” or “fields” selectors. For example:

```
GET
/api/troubleTicket?filter=attachment[?(@.size==300)]&fields==['id','href','name']
&sort=attachment[*].name
```

Notification Pattern - Registering Listener

The Notification Pattern and the mechanism of registering a listener are described in [DG4-1](page 82). This part of the design guide enhances the "query" expression that can be passed when registering a listener to support a JSON Path expression.

The query expression may be used to filter specific event types and/or any content of the event.

Considering the following example of resource:

```
{
    "id": "3180",
    "href":
"https://host:port/troubleTicket/v2/troubleTicket/3180[[.underline]#https://host:
port/troubleTicket/v2/troubleTicket/3180#]",
    "name": "Complaint over last bill",
    "status": "Resolved"
}
```

The structure of the event is:

```
{
    "eventId": "eventId",
    "eventTime": "eventTime",
    "eventType": "event Type",
    "event": {
        "resource": {
            "id": "3180",
            "href": "https://host:port/troubleTicket/v2/troubleTicket/3180",
            "name": "Complaint over last bill",
            "status": "Resolved"
        }
    }
}
```

Registering a notification with a “filter” selector is:

```
POST /api/hub
Accept: application/json

{
    "callback": "http://in.listener.com",
    "query": "[?(@.event.resource.status=='Resolved')]"
}
```

Error handling

The REST APIs MUST use the exception and response codes documented at http://www.iana.org/assignments/http-status-codes/http-status-codes.xml.

The same rules as described in [DG4-1] (page 23) will also apply for the cases where JSON Path is being used as a filtering mechanism.

Rule: If the JSON Path expression used in the “filter” or “fields” selectors, fails at the syntax level a 400 Bad Request MUST be returned.

Rule: If the JSON Path is not supported by the server implementation, but the client is making use of it in either “filter” or “fields” selectors a 501 Not Implemented MUST be returned.

The syntax errors can be enhanced further through user and application-specific error codes via the mechanism described in the [DG4-1] – page 25.

In the example below, the JSONPath expression from the request is missing a “)” in the predicate:

Request:

```
GET /troubleTicket?filter=[?(@.status=='Resoslved']&fields=name
```

Response:

```
400 Bad Request
Content-type:application/json

{
    "code": "ERR001",
    "reason": "Invalid JSONPath expression present in the filter selector",
    "message": " Could not parse token starting at position 2. Expected ?,
', 0-9, *"
}
```

JSON Patch

This part of the design guide enhances the existing JSON Patch extension to manage arrays, described in [DG4-5] with the JSON Path capabilities.

The following solution is a deviation from the JSON Patch standard where the “path” element is a JSON Pointer, however, the deviation exists as described in the [DG4-5].

The existing proposal has the scope to extend the “selector” conditions.

It is always recommended to use the “test” operation before applying any patch.

Considering a troubleTicket resource such as /api/troubleTicket/1:

```
{
    "id": "1",
    "note": [{
            "date": "2013-07-25T06:55:12.0Z",
            "author": "John Doe",
            "status": "Edited"
        },
        {
            "date": "2013-07-24T09:55:30.0Z",
            "author": "Arthur Evans",
            "status": "Edited"
        },
        {
            "date": "2013-07-25T08:55:12.0Z",
            "author": "John Doe",
            "status": "Archived"
        }]
}
----
```

The existing specification enables a JSON Patch document such as following where the condition is based on a simple path:

```
[{
    "op": "add",
    "path": "/note?note.author=John Doe",
    "value": "Informed"
}]
```

The JSON Path extension enhances the query and filtering capabilities.

Example - add where the "note" array has an "author" called John Doe

```
[{
    "op": "add",
    "path": "note[?(@.author=='John Doe')] ",
    "value": \{text:Informed}
}]
```

Applying the above path to the sample JSON will result in the following path to be automatically computed: "$['note'][1]"

Example: add where the "author" is “John Doe” and the “status” is “Edited”

```
[{
    "op": "add",
    "path": "note[?(@.author=='John Doe' && @.status=='Edited')]",
    "value": {"text":"Informed"}
}]
```

All the other capabilities are also supported through the JSON Path mechanisms either by a simple path, by a predicate expression or by a combination of them:

Removing or replacing one of the components of an array

Example:

```
[{
    "op": "remove",
    "path": "note[?(@.author='John Doe')]"
}]
```

Note: JSON Path capabilities on arrays are described in the **"Operators"** section in the current document.

Removing or replacing an attribute from one of the components of an array

Example:

```
[{
    "op": "remove",
    "path": "note[?(@.author='John Doe')].date"
}]
```

Administrative Appendix

This Appendix provides additional background material about the TM Forum and this document. In general, sections may be included or omitted as desired; however, a Document History must always be included.

Source Artefacts

#	Title
[JaywayJPath]	GitHub
[CDAP]	CDAP Documentation
[JSONPath]	JSONPath - XPath for JSON
[JSDB]	MySQL JSONPath
[BAELDUNG]	JaywayJsonPath
[TOOLSQA]	TOOLSQA
[GNOME]	JSONPath
[SMARTBEAR]	JSONPath
[POSTGRESS]	JSONPath

Referenced Artefacts

#	Title
[RFC2119]	RFC 2119
[RFC4627]	RFC 4627
[DG4-1]	TMF630_REST_API_Design_Guidelines_4.0_Part_1
[DG4-2]	TMF630_REST_API_Design_Guidelines_4.0_Part_2
[DG4-3]	TMF630_REST_API_Design_Guidelines_4.0_Part_3
[DG4-4]	TMF630_REST_API_Design_Guidelines_4.0_Part_4
[DG4-5]	TMF630_REST_API_Design_Guidelines_4.0_Part_5
[RFC3986]	RFC 3986

Document History

Version History

This section records the changes between this and the previous document version as it is edited by the team concerned. Note: this is an incremental number which does not have to match the release number and used for change control purposes only.

Version Number	Date Modified	Modified by:	*Description of changes
1.0			

Release History

This section records the changes between this and the previous Official document release. The release number is the 'Marketing' number which this version of the document is first being assigned to.

Release Number	Date Modified	Modified by:	Description of changes

Acknowledgments

This document was prepared by the members of the TM Forum team:

- Pierre Gauthier, TM Forum, Editor and Team Leader
- Florin Tene, Vodafone, Author