TM Forum Specification

REST API Design Guidelines Part 1

Practical guidelines for RESTful APIs naming, CRUD, filtering, notifications

TMF630

Team Approved Date: 13-Jan-2021

Release Status: Production	**Approval Status:** TM Forum Approved
Version: 4.0.2	**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

The “REST API Design Guidelines” document provides guidelines and design patterns used in developing TM Forum REST APIs. The document is organized in seven parts as follow:

- **Part One:** Practical guidelines for RESTful APIs naming, CRUD, filtering, notifications
- **Part Two:** Advanced guidelines for RESTful APIs polymorphism, extension patterns, depth and expand directive, entity RefOrValue
- **Part Three:** Guidelines for extending TMF Open APIs with hypermedia support
- **Part Four:** Advanced guidelines for RESTful APIs lifecycle management, common tasks
- **Part Five:** JSON Patch extension to manage arrays
- **Part Six:** JSON Path extension
- **Part Seven:** JSON Schema patterns

This part of the document provides recommendations and guidelines for the design and implementation of Entity CRUD operations and Task operations.

It also provides information on filtering and attribute selection. Finally, it does provide information on supporting notification management in REST based systems.

The uniform contract establishes a set of methods that are expected to be reused by services within a given collection or inventory.

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 RFC 2119 [RFC2119].

Chapter 1. General

1.1. API Resource Archetypes

The following section describes the resource archetypes supported by the TMF REST APIs

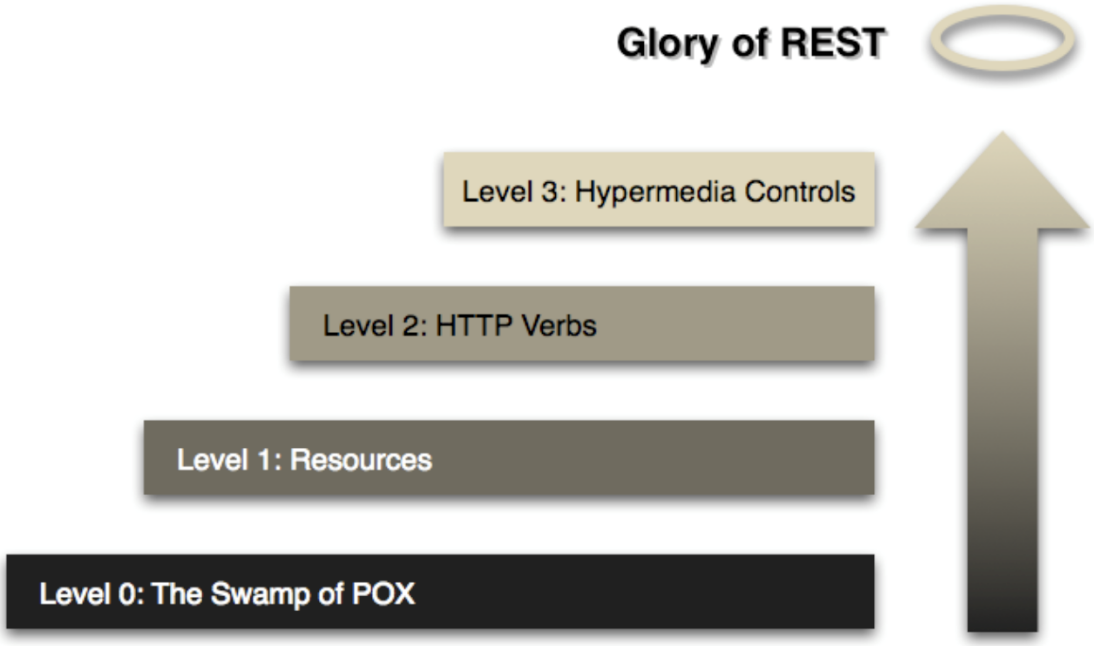
A REST API is composed of 3 distinct resource archetypes and should align each resource to just one of these:

- **Collection** Resource – server managed *collection* of resources. In the TMF REST API Design Patterns collections are anonymous. A resource with no identifier represents the resource collection (matching the resource type). Example: troubleTicket, productOrder
- **Managed** Resource – e.g. a database record or an object instance–. Its representation includes: fields with values and links to related resources. Can have child resources, of different resource types. Client can create, query, update and delete resources. Example: trouble Ticket/{id}, productOrder/{id}
- **Task** Resource – resources that are executable functions – with associated input and output parameters. Necessary where the required action cannot be mapped to standard CRUD methods. Tasks play the role of Controllers in Rails. E.g. ProductOfferingQualification, ServiceQualification

1.2. REST Levels

All APIs must implement Level 2 of the Richardson Maturity Model http://www.crummy.com/writing/speaking/2008-QCon/act3.html

The Level 3 is not mandatory and not specified in this part of the Design Pattern Guideline. Level 3 issues are addressed in the REST Guidelines Part 3 sections as they relate to workflow oriented patterns.



1.3. REST API SPECIFICATION INFORMATION

For each REST API specification, the following information MUST be included:

- Purpose of the API.
- URL of resources and API including version number.
- HTTP verbs supported.
- Representations supported: JSON (XML is optional)
- Response schema (*and where PUT, POST, PATCH are supported – request schema*).
- Links supported (Optional in L2 APIs)
- Response status codes supported.

The API MUST be described using Open API Specification (http://swagger.io/specification/)

1.4. API implementation technology

APIs are independent of the dependent technology implementation.

REST APIs embrace all aspects of the Hypertext Transfer Protocol, version 1.1 (HTTP/1.1) including its request methods, response codes, and message headers.

1.5. HTTP HEADERS

The following describe the header protocol elements that MUST be used by the TMF REST APIs.

1.6. GENERAL HTTP HEADERS

Cache-Control, Expires, and Date in response headers SHOULD be used when caching is required as specified in [RFC7234].

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-control: max-age=60, must-revalidate
```

- Cache-Control and Expires response headers MAY be used to discourage caching. If a response must not be cached, Cache-Control can be added with the value no-cache and no-store.
- Caching SHOULD be encouraged. The no-cache directive will prevent any cache from serving cached responses. This SHOULD not be used unless absolutely necessary.
- Expiration caching headers SHOULD be used with 200 (“OK”) responses to GET and HEAD requests.
- Expiration caching headers MAY be used with 3xx and 4xx responses - this helps reduce the amount of redirecting and error-triggering load on a REST API.

1.7. REQUEST AND RESPONSE HTTP HEADERS – CLIENT SIDE

The negotiation of the media type required by the client and provided by the server requires a number of headers in both the request and the response.

- The client MAY include ANY HTTP Header as specified in [RFC2730], [RFC2731], [RFC2732],[RFC2734],[RFC2735].
- The client MUST expect any HTTP header as specified in in [RFC2730], [RFC2731], [RFC2732],[RFC2734],[RFC2735].

The media type is the form of the response payload that the client would like to receive; this is specified using the standard HTTP header: Accept

- The client MUST use the Accept HTTP header to specify the media type.
- The Accept HTTP header MUST have a value matching [RFC2616].
- The client MUST support “application/json” by default.

If a media type is specified that is not supported by the server then the response must be returned using the default, application/json, i.e. as if no media type was specified.

- The client MAY specify the locale of the response in the request header.
- The client MUST use the Accept-Language HTTP header to specify the locale.
- The Accept-Language http header MUST have a value matching the template:

    - Locale: ISO-639, ("_", ISO3166-Alpha2)?
- The client MUST expect “en_GB” by default.

The Accept-Language header uses the [ISO639] language code and [ISO3166] country code standards separated by an underscore “_”, with the second half of the format being optional. If a language is specified that is not supported by the server then the response payload must be returned as “en_GB”.

1.8. REQUEST AND RESPONSE HTTP HEADERS – SERVER SIDE

As part of the content negotiation, just as the client can request particular behavior from the server, the server needs to respond back saying; “this is what you’re getting”.

- The server MAY include any HTTP header as specified in [RFC2730], [RFC2731], [RFC2732],[RFC2734],[RFC2735].
- The server MUST expect any HTTP header as specified in in [RFC2730], [RFC2731], [RFC2732],[RFC2734],[RFC2735].
- The server SHOULD specify the media type of the response in the response header
- The server MUST use the Content-Type HTTP header to specify media type
- The Content–Type HTTP header MUST have a value matching in [RFC2730], [RFC2731], [RFC2732],[RFC2734],[RFC2735].
- The server MUST support “application/json” by default.

  The above rules work in the same way as the Accept; “application/json” is supported as the default media type of the server. This is returned either when requested or when no Accept header is present. The server SHOULD provide this header back to the client for the sake of clarity and so that the client does not have to “detect” the actual media type of the response. However, it is not required and the client SHOULD assume that either the requested media type or the default is being returned.

- The server SHOULD specify the Content-Length HTTP header. Client can thus know whether it has read the correct number of bytes from the connection and can make a HEAD request to find out how large the entity-body is, without downloading it.
- e.g. HTTP/1.1 200 OK Content-Type: application/json Content-Language: en_GB Content-Length: 1024 ETag: “x234dff”
- The server SHOULD use Last-Modified HTTP header in the responses to specify the time at which the resources where created.
- The server SHOULD use ETag in responses. The entity tag may be any string value, so long as it changes along with the resource’s representation.
- The server MUST use Location HTTP header to specify the URI of a newly created resource and may be used to direct clients to the operational status of an asynchronous controller resource. This is the preferred method to implement asynchronous behavior.

The following specifies the locale within the response; this matches the Accept-Language HTTP header in the request message.

- The server MAY specify the locale of the response in the response header.
- The server MUST use the Content-Language HTTP header to specify the locale.
- The Content-Language HTTP header MUST have a value matching the template:
  - Locale: ISO-639, ("_", ISO3166-Alpha2)?
- The server MUST support “en_GB” by default.
- The supported locales MUST be documented.

1.9. CUSTOM HEADER

- The server SHOULD return the HTTP header “X-Total-Count” in a response to a Get List resource creation request.

  The HTTP header must specify the total number of matching resources to the Get, for example:

Request

```
GET /troubleTicketManagement/troubleTicket?count=0&limit=20
```

Response

```
Content-Type: application/json
X-Total-Count: 200
```

- The server SHOULD specify a rate limit. Rate Limiting of an API can defined on a per-user basis and per application basis. If an operation allows X requests per rate limit window, then it allows the client to make X requests per window.
- The server SHOULD return a 429 Too Many Requests http response when this limit is exceeded.
- The server SHOULD return the following HTTP headers:
  - X-Rate-Limit-Limit: The # of requests allowed in the current period
  - X-Rate-Limit-Remaining: The #of requests remaining in the current period
  - X-Rate-Limit-Reset: The # of seconds left in the current period.

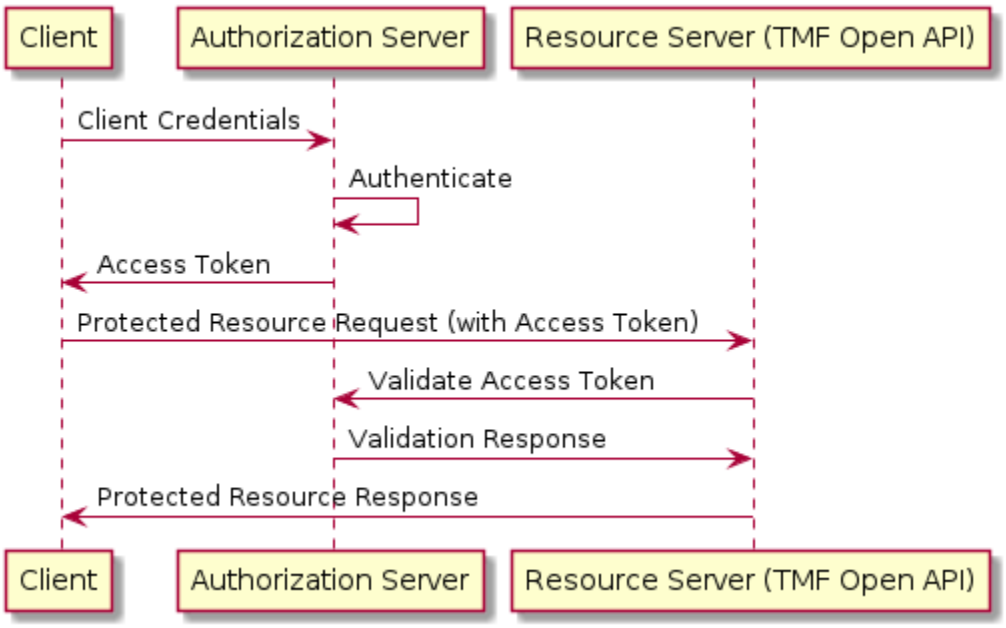
1.10. Authentication

Many TMF Open APIs contain sensitive information and provide access to both customer and business

operation details. Care should be taken to secure these OpenAPI interfaces to prevent unauthorized access. Each API should be secured following the principle of least privilege (PoLP).

OAS provides many different mechanisms to secure APIs and one implementation method will not satisfy every provider's implementation. Additionally, one API may have different authentication strategies depending on where the service is deployed or which client is consuming the service. The TM Forum recommends all Open API interfaces are secured and provides general guidelines and methods to implement Authentication and authorization of these interfaces.

The most common method of securing APIs is using OAuth 2.0. OAuth separates the authentication provider from the resource. The authorization server issues access tokens which are used to access the protected resource. This allows consumption of the API without sharing credentials with the API provider. OAuth 2.0 is implemented in various ways using *flows*. The most common flow for machine-to-machine interfaces is the *Client Credentials* flow. Below is a sequence diagram illustrating the *Client Credentials* flow.



This security method can be applied to TMF Open APIs by decorating the specifications using standard OAS security definitions. Below is an example of a Client Credential flow security definition (Swagger 2.X)

```
{
  "securityDefinitions": {
    "oauth2": {
      "flow": "application",
      "tokenUrl": "https://*****/oauth/token",
      "scopes": {
        "read": "Read access",
        "write": "Write Access",
        "admin": "Admin Access"
      },
      "type": "oauth2"
    }
  }
}
```

Once defined this security definition can be applied to all methods in the API or applied to individual methods.

Example of security applied to ALL methods:

```
{
  "swagger" : "2.0",
  "paths" : {
    "/resource" : {
      "get" : {},
      "post" : {}
    },
    "security" : [{
      "oauth2" : ["read"]
    },
      "securityDefinitions" : {
        "oauth2" : {
          "flow" : "application",
          "tokenUrl" : "https://*****/oauth/token",
          "scopes" : {
            "read" : "Read access",
            "write" : "Write Access",
            "admin" : "Admin Access"
          },
          "type" : "oauth2"
        }
      }]
  }
}
```

Example of security applied to individual methods:

```
{
  "swagger": "2.0",
  "paths": {
    "/resource": {
      "get": {
        "security": [
          {
            "oauth2": [
              "read"
            ]
          }
        ]
      },
      "post": {
        "security": [
          {
            "oauth2": [
              "write"
            ]
          }
        ]
      }
    },
    "securityDefinitions": {
      "oauth2": {
        "flow": "application",
        "tokenUrl": "https://*****/oauth/token",
        "scopes": {
          "read": "Read access",
          "write": "Write Access",
          "admin": "Admin Access"
        },
        "type": "oauth2"
      }
    }
  }
}
```

Chapter 2. Domain and URI Naming Standards

Resources represent managed entities acted upon by the REST API. Resource identifiers represent the actual resources that a service exposes. A resource can be a Trouble Ticket, a Logical Port, an Order, a Task etc…

A resource identifier is like a unique ID assigned to one or more service resources (also known as a key in OSS/J and entity identifier in TIP). The Resource Identifiers recommendation standardizes the syntax used to represent them. The most common syntax used to express resource identifiers is the Web's Uniform Resource Identifier (URI) syntax.

2.1. Managed Entity Model

Resources represent managed entities. Resources are acted upon by the REST API using the Uniform Contract Verbs (POST, GET, PUT, DELETE, PATCH, etc.…). Operations on Resources affect the state of the corresponding managed entities.

There is a direct mapping between the managed entities and the corresponding Resources in the REST model.

The mapping between the managed entity types and the corresponding Resource model MUST be included in the API specification.

2.2. Resource Model and Naming

A resource identifier is like a unique ID assigned to one or more service resources (also known as a key in OSS/J and entity identifier in TIP).

The URI path may convey the REST API's resource model, with each forward slash separated path segment corresponding to a unique resource within the model's hierarchy.

The URI of a resource can also be represented by an ID which contains the sequence of Relative Distingushed Name (RDN) without having the RDNs to be necessarily encoded as individual resources with an URI.

Characters that require encoding MUST NOT be used in the URI.

Individual resources MUST have a unique identifier field called id.

If the Id is a composite key then it should be split into the following elements: key = {part}-{part}*.

For example, if the key is a combination of managementSystem and name then the id would be represented as:

```
id = {managementSystem}-{name}
```

The structure of a Resource Path URI is given by the following expression:

```
{resourcePath} = {resourceName} [ ( / {resourceID*} [ / {resourcePath} or
{taskResource} ] ) Or ( /{taskResource} ) ]
```

For example, in our sample Trouble Ticket Management application: {resourcePath} = ticket/{ticketID}

Resource URIs SHOULD be formed according to the following base pattern:

```
{apiRoot} /{resourcePath}
```

apiRoot points to the root path of collection of resources.

2.3. Resource ID and href

Every Resource have two mandatory attributes:

- id: A unique identifier in the context of the application collection `id=42`
- href: The URI of the resource as per Location header

  `href= troubleTicket/42` (maybe a full URL but in general relative URIs are used)

All APIs SHOULD use HTTP/SSL, “HTTPS” as the scheme.

The ID MUST be canonicalized and MUST not be a business identifier.

The ID must be unique and immutable within the collection.

2.4. Resource Naming Convention

Names in URI (tasks, individual resources, etc.) MUST be camel case or lower case.

e.g. `/account/billSummary/billDetail` or `/account/billSummary/billdetail`

Each resource name must be a full name, abbreviations and acronyms MUST NOT be used.

e.g. `resourceSpecification` not `ResourseSpec`

The resource URI MUST NOT use meaningless words.

e.g. `Management`, `Maintenance`, `Provision`

The API Resource name MUST be derived from the TMF Information Framework.

Managed resource names and collection resource names MUST be a noun; except tasks resource names that SHOULD be a verb.

e.g. Service Problem, Trouble Ticket, Service Order - as a resource

e.g. close, cancel, group, send, check - as a task

The resource name MUST NOT expose technical or implementation details.

2.5. Identifier syntax and uniform contract verbs

URI names MUST NOT contain the names of HTTP verbs (task resources should be used if needed for clarity).

2.6. Resource collection naming

Collection names MUST not use a Collection postfix. Collections are implicit when the resource name is used.

For example, API/ticket represents the Ticket Collection and GET API/ticket retrieves all the items from the Ticket Collection (i.e. all the resources of type Ticket contained within the application).

Chapter 3. Uniform Contract Methods and Media Types

HTTP provides us with a set of generic methods, such as GET, PUT, PATCH, POST, DELETE, HEAD and OPTIONS, that are pre-defined in the HTTP specification. The complete protocol interactions include a set of response codes, plus syntax for expressing various parameters that can be encoded in HTTP messages.

The REST uniform contract is based on three fundamental elements:

- resource identifier syntax – How can we express where the data is being transferred to or from?
- methods – What are the protocol mechanisms used to transfer the data?
- media types – What type of data is being transferred?

The following section describes the guidelines for modeling operations and for specifying what media types to use.

3.1. Uniform Contract Operations

All API operations are based on the REST Uniform Contract operations.

- GET and POST must **not** be used to tunnel other request methods in the TMF Open API specifications although it is recognized as a practice in an implementation.
- HEAD must be used to retrieve response headers. HEAD support is always optional.

The following describe the relationships between the elements of the API and the Uniform Contract.

Only legitimate resources can be acted upon using Uniform operation and not any dependent entities.

Operations on Entities are mapped to operations on the corresponding resources:

Operation on Entities	Uniform API Operation	Description
Query Entities	GET Resource	GET must be used to retrieve a representation of a resource.
Create Entity	POST Resource	POST must be used to create a new resource
Partial Update of an Entity	PATCH Resource	PATCH must be used to partially update a resource
Complete Update of an Entity	PUT Resource	PUT CAN be used to completely update a resource identified by its resource URI

Operation on Entities	Uniform API Operation	Description
Remove an Entity	DELETE Resource	DELETE must be used to remove a resource
Execute an Action on an Entity	POST on TASK Resource	POST must be used to execute Task Resources
Other Request Methods	POST on TASK Resource	GET and POST must **not** be used to tunnel other request methods.

3.2. API Media Types

When defining methods for REST services, we can further specify the types of data a given method can process. For example, a GET method may be able to transfer a Trouble Ticket representation in XML or JSON. Each is represented by its own media type.

- REST APIs MUST support the “application/json” media type by default.
- In case of PATCH if application/json is provided then the default rule will be to apply the same rules as for JSON merge.
- In case of JSON Patch RFC6902 for partial updates “application/json-patch+json” media type MUST be used.
- In case of JSON PATCH Query, “application/json-patch-query+json” media type MUST be used.
- In case of PATCH as per RFC7396 “application/merge-patch+json” MUST be used
- The default for resource representation MUST be JSON.
- An API MUST only use the ACCEPT HEADER and CONTENT-TYPE (POST) to control the representation media types. Other mechanisms SHOULD not be supported.

3.3. API RESPONSE STATUS and EXCEPTION CODES

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

In particular, the following response codes must be used:

Status Code	Rule
2xx	**Success Indicates that the client’s request was accepted successfully.**
200	OK - SHOULD be used to indicate nonspecific success. Must **not** be used to communicate errors in the response

Status Code	Rule
201	Created - MUST be used to indicate successful resource creation. Return message SHOULD contain a resource representation and a Location header with the created resource's URI
202	Accepted - MUST be used to indicate successful start of an asynchronous action
204	No Content - SHOULD be used when the response body is intentionally empty
206	Partial Content – MUST be used for **Partial resource returned in response (with pagination)**
3xx	**Redirection Indicates that the client must take some additional action in order to complete their request.**
301	Moved Permanently - SHOULD be used to relocate resources
302	Found - SHOULD not be used
303	See Other - SHOULD be used to refer the client to a different URI – can be used with a Location header containing the URI of a resource that shows the outcome of an asynchronous task.
304	Not Modified - SHOULD be used to preserve bandwidth
307	Temporary Redirect - SHOULD be used to tell clients to resubmit the request to another URI
4xx	**Client Error This category of error status codes points the finger at clients.**
400	Bad Request - MAY be used to indicate nonspecific failure. The request could not be understood by the server. The client SHOULD NOT repeat the request without modifications
401	Unauthorized - MUST be used when there is a problem with the client's credentials

Status Code	Rule
403	Forbidden - SHOULD be used to forbid access regardless of authorization state. *For example, a client may be authorized to interact with some, but not all of a REST API's resources. If the client attempts a resource interaction that is outside of its permitted scope, the REST API should respond with 403.*
404	Not Found - MUST be used when a client's provided URI cannot be mapped to a resource URI
405	Method Not Allowed - MUST be used when the HTTP method is not supported
406	No Acceptable - The requested resource is capable of generating only content not acceptable according to the Accept headers sent in the request
409	Conflict -The request could not be completed due to a conflict with the current state of the target resource.
410	Gone - The requested resource is no longer available at the server and no forwarding address is known.
411	Length required-The server refuses to accept the request without a defined Content- Length.
412	Precondition Failed - The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server
413	Request Entity Too Large- The server is refusing to process a request because the request entity is larger than the server is willing or able to process
414	Request-URI Too Long- The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret

Status Code	Rule
415	Unsupported Media Type - The request entity has a media type which the server or resource does not support. For example, the client uploads an image as image/svg+xml, but the server requires that images use a different format.
422	Unprocessable Entity - The request was well-formed but was unable to be followed due to semantic errors.
429	Too Many Requests – SHOULD be used to indicate that the client has sent to many requests in a given amount of time (“rate limiting”).
5xx	**Server Error This category of error status codes points the finger at servers.**
500	Internal Server Error – a generic error message, given when an unexpected condition was encountered and no more specific message is suitable.
501	Not implemented- the server either does not recognize the request method, or it lacks the ability to fulfil the request. Usually this implies future availability (e.g., a new feature of a web-service API).
503	Service unavailable - The server is currently unavailable (because it is overloaded or down for maintenance). Generally, this is a temporary state.

3.4. User, Application and Extended Error Codes

The HTTP 4xx or 5xx standard error codes should always be used in the response header.

An API MAY support user and application specific error codes.

User and Application specific Error Codes MUST be represented in the Error Representation (Body) of the response.

Sub codes are possible (400-2) however they MUST be in the Error Representation (Body)

Error Representation in body of the response MUST have the following structure:

Property	Description	Mandatory
code	Application related code (as defined in the API or from a common list)	Mandatory
reason	Text that explains the reason for error. This can be shown to a client user.	Mandatory
message	Text that provide more details and corrective actions related to the error. This can be shown to a client user.	Optional
status	http error code extension like 400-2	Optional
referenceError	url pointing to documentation describing the error	Optional
@type	The class type of a REST resource.	Optional
@schemaLocation	it provides a link to the schema describing a REST resource.	Optional

Example:

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

```
{
  "code": "ERR001",
  "reason": "Missing mandatory field",
  "message": "Please provide an Authorisation header"
}
```

The error codes if supported SHOULD be defined in the API specification.

`@type` and `@schemaLocation` SHOULD be used for run time extension of the error.

3.5. Representations

- (Well formed) JSON MUST be the default for resource representation

- XML and other formats may optionally be supported via content negotiation with the client
- Media type selection MUST be signaled with the Accept header.
- Documents (individual resources) MUST have a unique identifier field called ID.
- All documentation SHOULD link to a schema (JSON schema or W3C XML Schema)
- Additional envelopes must not be created. A REST API must leverage the message “envelope” provided by HTTP.

3.6. Common Information Model

When applicable, the data types used to define the information model in API parameters SHOULD follow the Information Framework reference model from TM Forum. In particular:

- Where there are fields defined in the TMF Information Framework for an API resource then the same field names SHOULD be used in the API.
- The mandatory fields from the TMF Information Framework SHOULD be represented in the API resource model.
- The optional fields from the TMF Information Framework MAY be represented in the API resource model.
- If mandatory fields for the API resource are not found in the TMF Information Framework then the fields MAY be added to the TMF Information Framework.
- If optional fields for the API resource are not found in the TMF Information Framework then the fields MAY be added to the TMF Information Framework.

Chapter 4. Query Resources Patterns

The following section describes the structure and constraints of query operations.

All examples are relative to the management of TroubleTicket entities having the following JSON representation.

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
  "status": "In Progress",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": " /CustomerManagement /customer/1234",
      "role": "Originator"
    },
    {
      "href": " /partyManagement/party/1234",
      "role": "Owner"
    },
    {
      "href": " /partyManagement/party /23445",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": "/customerBill/1234"
    },
    {
      "involvement": "Adjusted",
      "reference": "/CustomerBill/5678"
    }
  ],
  "note": [
```

```
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
    }
  ]
}
```

4.1. Query single Resource all attributes

GET {apiRoot} /{resourceName}/{resourceID} MUST be used to retrieve the representation of a resource named resourceID.

If the resource exists, the complete resource representation (with all the attributes) must be returned.

The returned representation must contain a field called « id» and that field be populated with the resourceID.

If the request is successful then the returned code MUST be 200 OK.

If there are no matching resource then a 404 Not Found must be returned.

HTTP Response Codes - Success	200 OK
HTTP Response Codes - Failure	400 Bad Request 404 Not Found 405 Method Not Allowed 500 Internal Server Error

The exceptions code must use the exception codes from http://www.iana.org/assignments/http-status-codes/http-status-codes.xml as explained in section 4.3.

Retrieving a Single TroubleTicket with an ID of 42:

|**REQUEST**

```
GET /api/troubleTicket/42
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
  "status": "In Progress",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": " /customerManagement/customer/1111",
      "role": "Originator"
    },
    {
      "href": " /partyManagement/individual/2222",
      "role": "Owner"
    },
    {
      "href": " /partyManagement/individual/12345",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": "/billingManagement/customerBill/1234"
    },
    {
      "involvement": "Adjusted",
      "reference": " /billManagement/customerBill/5678"
    }
```

```
  ],
  "note": [
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
    }
  ]
}
```

4.2. Querying multiple Resources

The following section describes how to use the GET operation to retrieve multiple resources without specifying id's.

`GET {apiRoot}/{resourceName}` must be used to retrieve the representation of all the resource for a resource type corresponding to /{resourceName}

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

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

If the request is successful then the returned code MUST be 200 OK otherwise, the appropriate error code SHOULD be returned.

HTTP Response Codes - Success	200 OK
HTTP Response Codes - Failure	400 Bad Request 404 Not Found 405 Method Not Allowed 500 Internal Server Error

The exceptions code must use the exception codes from http://www.iana.org/assignments/http-status-codes/http-status-codes.xml as explained in section 4.3.

Example:

Retrieving all troubleTickets.

REQUEST

```
GET /api/troubleTicket
```

RESPONSE

```
200 OK
Content-Type: application/json
```


```
[
  {
    "id": "42",
    "href": " /troubleTicketManagement/troubleTicket/42",
    "correlationId": "TT53482",
    "description": "Customer complaint over last invoice.",
    "severity": "Urgent",
    "@type": "TroubleTicket",
    "creationDate": "2013-07-23T08:16:39.0Z",
    "targetResolutionDate": "2013-07-30T10:20:01.0Z",
    "status": "In Progress",
    "statusChangeReason": "Waiting for invoicing expert.",
    "statusChangeDate": "2013-07-24T08:55:12.0Z",
    "relatedParty": [
      {
        "href": " /billManagement/customer/1234",
        "role": "Originator"
      },
      {
        "href": " /partyMangement/individual/1234",
        "role": "Owner"
      },
      {
        "href": " /partyManagement/individual/Roger Collins",
        "role": "Reviser"
      }
    ],
    "relatedObject": [
      {
```

```
        "involvement": "Disputed",
        "reference": " /billManagement/customerBill/1234"
      },
      {
        "involvement": "Adjusted",
        "reference": " /billManagement/customerBill/5678"
      }
    ],
    "note": [
      {
        "date": "2013-07-24T09:55:30.0Z",
        "author": "Arthur Evans",
        "text": "Already called the expert"
      },
      {
        "date": "2013-07-25T08:55:12.0Z",
        "author": "Arthur Evans",
        "text": "Informed the originator"
      }
    ]
  },
  {
    "id": "43",
    "href": " /troubleTicketManagement/troubleTicket/43",
    "correlationId": "TT53482",
    "description": "Customer complaint over last invoice.",
    "severity": "Urgent",
    "@type": "TroubleTicket",
    "creationDate": "2014-07-23T08:16:39.0Z",
    "targetResolutionDate": "2014-07-30T10:20:01.0Z",
    "status": "In Progress",
    "statusChangeReason": "Waiting for invoicing expert.",
    "statusChangeDate": "2014-07-24T08:55:12.0Z",
    "relatedParty": [
      {
        "href": " /customerManagement/customer/1234",
        "role": "Originator"
      },
      {
        "href": " /PartyManagement/individual/1234",
        "role": "Owner"
      }
    ],
    "relatedObject": [
      {
```

```
        "involvement": "Disputed",
        "reference": " /billManagement/customerBill/1236"
      },
      {
        "involvement": "Adjusted",
        "reference": " /billManagement/customerBill/5679"
      }
    ],
    "note": [
      {
        "date": "2014-07-24T09:55:30.0Z",
        "author": "Arthur Evans",
        "text": "Already called the expert"
      }
    ]
  }
]
```

4.3. Query partial Resource representation or attribute selection

The following section describes how to select a subset of the attributes of an entity to be present in a returned representation.

Attribute selection MUST be enabled on all first level attributes but not on inner classes.

Attribute selection MAY optionally be enabled on all attributes and inner classes.

An attribute selector directive called “fields” MUST be used to specify the attributes to be returned as part of a partial representation of a resource.

`GET {apiRoot}/{resourceName}/{resourceID}?fields=\{attributeName*}` MUST be used to retrieve the partial representation of a resource with the attributes named resourceID*.

If the no attribute selector directive is provided, then the complete resource representation (with all the attributes) must be returned.

In order to indicate that no resource properties should be returned the following directive can be used: “fields=none”.

ID and HREF MUST be returned in the resource body representation when fields=none is used.

Fields MAY be used with other Uniform Operations. See chapter 6 for an example where fields can be used with POST operation.

Example:

REQUEST

```
GET /api/troubleTicket?fields=none
Content-Type: application/json
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42"
}
```

If there is a name clash between the attribute name and the name of a resource directly under the parent then /?fields=\{attributeName*} should be used.

The returned representation must contain a field called « id» and that field be populated with the resourceID.

If the request is successful, then the returned code MUST be 200 OK.

The exceptions code must use the exception codes from http://www.iana.org/assignments/http-status-codes/http-status-codes.xml as explained in section 4.3.

Example:

Retrieve the report with an « id » of 42 and populate the representations with the description and status attributes. Note that the id and href attributes are always present.

REQUEST

```
GET /api/troubleTicket/42/?fields=description,status
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
{
    "id": "42",
    "href": " /troubleTicketManagement/troubleTicket/42",
    "description": "Customer complaint over last invoice.",
    "status": "In Progress"
}
```

4.4. Query Resources with attribute filtering

The following section describes how to retrieve resources using an attribute filtering mechanism. The filtering is based on using name value query parameters on entity attributes.

The basic expression is a sequence of attribute assertions being ANDed to formulate a filtering expression:

```
GET \{apiRoot} /\{resourceName}?[\{attributeName}=\{attributeValue}&*]
```

For examples:

```
GET /api/troubleTicket/?status=acknowledged & creationDate=2017-04-20
```

Note that the above expressions match only for attribute value equality.

Attribute values ORING is supported and is achieved by providing a filtering expression where the same attribute name is duplicated a number of times `[{attributeName}={attributeValue}&*]` different values.

Alternatively the following expression `[{attributeName}={attributeValue},\{ attributeValue }*]` is also supported. ORING can also be explicit by using “ ; “ many time `[{attributeName}={attributeValue};{attributeValue}*]`

The “OR” behavior can also be achieved by providing a filtering expression using one of the following expressions.

For example:

- `GET /api/troubleTicket?status=acknowledged;status=rejected`
- `GET /api/troubleTicket status=acknowledged,rejected`

The following operators may be used:

Operator literal	Description	URL Encoded form
.gt >	greater than (>) Return results where the search criteria field is strictly greater than	%3E
.gte >=	greater than or equal to (>=) Return results where the search criteria field is equal of greater than	%3E%3D
.lt <	Less than (<) Return results where the search criteria field is strictly less than	%3C
.lte □	less than or equal to (□) Return results where the search criteria field is equal or less than	%3C%3D
regex *=	Regexp expression Note: all regexp special characters must be encoded	%3D~
.eq	Equal to (=) Returns results where the search criteria fields is equal to	%3D%3D

Examples:

Using ANDED to describe “017-04-20 >dateTime>2013-04-20” :

- `GET /api/troubleTicket?dateTime%3E2013-04-20&dateTime%3C2017-04-20`

Using ORING to describe dateTime<2013-04-20 or dateTime<2017-04-20 :

- `GET /api/troubleTicket?dateTime%3C2013-04-20;dateTime%3C2017-04-20`

Complex attribute value type may be filtered using a “.” notation.

```
{empty}[\{attributeName.attributeName}=\{attributeValue}&*]
```

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

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

If the request is successful then the returned code MUST be 200 OK.

The exceptions code must use the exception codes from http://www.iana.org/assignments/http-status-codes/http-status-codes.xml as explained in section 4.3.

Example:

Retrieve all Trouble Tickets with dateTime greater than 2013-04-20 and status acknowledged.

REQUEST

```
GET /api/troubleTicket?dateTime.gt=2013-04-20&status=acknowledged
```

or

```
GET /api/troubleTicket?dateTime%3E2013-04-20&status=acknowledged
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
[
  {
    "id": "42",
    "href": " /troubleTicketManagement/troubleTicket/42",
    "correlationId": "TT53482",
    "description": "Customer complaint over last invoice.",
    "severity": "Urgent",
    "@type": "TroubleTicket",
    "creationDate": "2013-07-23T08:16:39.0Z",
    "targetResolutionDate": "2013-07-30T10:20:01.0Z",
    "status": "Acknowledged",
    "statusChangeReason": "Waiting for invoicing expert.",
    "statusChangeDate": "2013-07-24T08:55:12.0Z",
    "relatedParty": [
      {
        "href": " /customerManagement/customer/1234",
        "role": "Originator"
      },
      {
```

```
        "href": " /partyManagement/party/1234",
        "role": "Owner"
      },
      {
        "href": " /partyManagement/individual/3455",
        "role": "Reviser"
      }
    ],
    "relatedObject": [
      {
        "involvement": "Disputed",
        "reference": " /billManagement/customerBill/1234"
      },
      {
        "involvement": "Adjusted",
        "reference": " /billManagement/customerBill/5678"
      }
    ],
    "note": [
      {
        "date": "2013-07-24T09:55:30.0Z",
        "author": "Arthur Evans",
        "text": "Already called the expert"
      },
      {
        "date": "2013-07-25T08:55:12.0Z",
        "author": "Arthur Evans",
        "text": "Informed the originator"
      }
    ]
  },
  {
    "id": "43",
    "href": " /troubleTicketManagement/troubleTicket/43",
    "correlationId": "TT53482",
    "description": "Customer complaint over last invoice.",
    "severity": "Urgent",
    "@type": "TroubleTicket",
    "creationDate": "2013-07-23T08:16:39.0Z",
    "targetResolutionDate": "2013-07-30T10:20:01.0Z",
    "status": "Acknowledged",
    "statusChangeReason": "Waiting for invoicing expert.",
    "statusChangeDate": "2013-07-24T08:55:12.0Z",
    "relatedParty": [
      {
```

```
        "href": " /customerManagement/customer/1234",
        "role": "Originator"
      },
      {
        "href": " /partyManagement/individual/1234",
        "role": "Owner"
      },
      {
        "href": " /partyManagement/individual/2222",
        "role": "Reviser"
      }
    ],
    "relatedObject": [
      {
        "involvement": "Disputed",
        "reference": " /billManagement/customerBill/1234"
      },
      {
        "involvement": "Adjusted",
        "reference": " /billManagement/CustomerBill/5678"
      }
    ],
    "note": [
      {
        "date": "2013-07-24T09:55:30.0Z",
        "author": "Arthur Evans",
        "text": "Already called the expert"
      },
      {
        "date": "2013-07-25T08:55:12.0Z",
        "author": "Arthur Evans",
        "text": "Informed the originator"
      }
    ]
  }
]
```

4.5. Query Resources with attribute filtering and Iterators

To support the return of large collections of resources the GET HTTP method must support pagination. Attribute filtering and iterators may be combined in a single request as per the following example:

REQUEST GET /api/troubleTicket?offset=20&limit=10&status =acknowledged

4.5.1. Paging

If no paging query parameters are specified in the URI then all matching resources SHOULD be returned to the consumer. The server MAY return a partial response with default values for offset and limit.

For query based pagination, the following query parameters MUST be supported:

Parameter	Type	Description
offset	integer	Requested index for start of resources to be provided in response requested by client
limit	integer	Requested number of resources to be provided in response requested by client

The above pagination query parameters support both getting a page in the middle of a resultset and the capability to get the next page.

If the offset query parameter is missing then it must default to zero.

?limit=20	Get the first twenty matching resources.
?offset=0&limit=20	Get the first twenty matching resources.
?offset=10&limit=20	Get the twenty resources starting at the tenth

For query-based pagination, the server MUST return the HTTP header “X-Total-Count” in a response, with the total number of matching resources so that the client can calculate the next page.

The server SHOULD return navigation links as Web Linking HTTP header, to ease the navigation and avoid that client has to construct the related links (first, next, previous, last).

The following HTTP response code MUST be returned by the server:

Code	Message	Description
200	OK	Full resource returned in response (no pagination)*
206	Partial Content	Partial resource returned in response (with pagination)

Example:

REQUEST

```
GET api/troubleTicket?offset=20&limit=10
Content-type: application/json
```

RESPONSE

```
206 Partial Content
Content-Type: application/json
X-Total-Count:50
```

```
Link:
<https:server:port/troubleTicketManagement/troubleTicket?offset=20&limit=10>;
rel="self",
<https:server:port/troubleTicketManagement/troubleTicket?offset=0&count=10>;
rel="first",
<https:server:port/troubleTicketManagement/troubleTicket?offset=30&limit=10>;
rel="next",
<https:server:port/troubleTicketManagement/
troubleTicket?offset=10&limit=10>; rel="prev",
<https:server:port/troubleTicketManagement/
troubleTicket?offset=40&limit=10>; rel="last"
```

```
[{
  "id": "20",
  "href": " /troubleTicketManagement/troubleTicket/20",
  "status": " acknowledged"
},
  {
    "id": "21",
    "href": " /troubleTicketManagement/troubleTicket/21",
    "status": "acknowledged"
  },
  {
    "id": "29",
    "href": " /troubleTicketManagement/troubleTicket/29",
    "status": "acknowledged"
  }]
```

The content Range header MAY also be used to control the amount of data returned. This header is present in the request and control the minimum and maximum values returned.

Filtered queries returns collections and the content range header is relative to the entities in the returned collection.

The following example shows how to use the Range header to iterate a collection of trouble tickets. The example assumes that 50 trouble tickets match a filtering criteria and that the maximum amount of trouble tickets being retrieved by calls is set by default to 10.

Example:

REQUEST

```
GET /api/troubleTicket?dateTime.gt=2013-04-20&status=acknowledged
```

RESPONSE

```
200 OK
Content-Type: application/json
Content-Range: items 1-10/50
```

```
[
  {
    "id": "41",
    "href": " /troubleTicketManagement/troubleTicket/41",
    "status": "In Progress"
  },
  {
    "id": "42",
    "href": " /troubleTicketManagement/troubleTicket/42",
    "status": "acknowledged"
  },
  {
    "id": "43",
    "href": " /troubleTicketManagement/troubleTicket/43",
    "status": "acknowledged"
  }
]
```

Retrieving the next elements:

REQUEST

```
GET /api/troubleTicket?dateTime.gt=2013-04-20&status=acknowledged
Accept-Range:items
Range:items=11-20
```

RESPONSE

```
206 Partial Content
Content-Type: application/json
Content-Range: items 11-20/50
```

```
[
  {
    "id": "47",
    "href": " /troubleTicketManagement/troubleTicket/47",
    "status": " acknowledged"
  },
  {
    "id": "48",
    "href": " /troubleTicketManagement/troubleTicket/48",
    "status": "acknowledged"
  },
  {
    "id": "49",
    "href": " /troubleTicketManagement/troubleTicket/49",
    "status": "acknowledged"
  }
]
```

4.6. Query Resources with attribute filtering and attribute selection

Attribute filtering and attribute selection may be combined in a single request as per the following example:

REQUEST GET /api/troubleTicket?fields=description,status&creationDate.gt=2013-04-20&status=acknowledged

4.7. Sorting

To support sorting the sort directive is used in HTTP query parameters is used.

If sorting is used then following parameters MUST be supported:

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

Sort-Direction : “-“ | “+”

Sort-Field: The field to sort on.

The default direction is Ascending order, the use of the modifier in front of the sort field name, “-“, changes the sort order direction.

`?sort=name`	Sort the resultset on the name.
`?sort=-name`	Sort the resultset on the name in descending order.

Coma separated list of resource attributes can be requested by client:

```
?sort= [attributeName],⋯,[atributeName]
```

Nested resource attributes can be specified,as pattern:

```
{empty}[parentResource].[childResource].[attributeName]
```

Sorting order is specified by the left to right order of the fields listed.

Note that the “.sort” format can be used (as for any directive) if there is an ambiguity regarding the semantic of sort within the query parameters.

REQUEST

```
GET /api/troubleTicket?sort=creationDate,statusChangeDate,note.date
```

Chapter 5. Modify resources patterns

The following section describes the patterns used to modify resources.

5.1. Uniform contract operations for modifying resources

TMF REST APIs MUST only use PUT and PATCH to modify the attributes of an entity.

The TMF REST Design Guideline is strict about the usage of PUT versus PATCH.

- PUT should be used when the semantic of the operation is replace all.
- PUT MUST NOT be used for the partial updates of attributes.
- PATCH MUST be used if a partial update is required.

5.2. Replace all attributes of a resource

A PUT replaces the current object with the provided object.

If the intent is to update a limited subset of properties of the resource then PATCH MUST be used update an object.

One approach to using PUT to facilitate a partial “replace” functionality is to first GET the object, modify it in memory, then to PUT the modified object back.

If the request is successful then the returned code MUST be 200 OK.

If the request is asynchronous then the returned code MUST be 202 Accepted.

The exceptions code must use the exception codes from http://www.iana.org/assignments/http-status-codes/http-status-codes.xml as explained in section 4.3.

HTTP Response Codes - Success	200 OK 202 Accepted
HTTP Response Codes - Failure	400 Bad Request 404 Not Found 405 Method Not Allowed 409 Conflict 500 Internal Server Error

Full resource representation MUST be included in the response if the request is successful, and error code otherwise with no resource representation.

Example:

Change the status of the trouble ticket with ID= 42 to resolved.

REQUEST

```
PUT /api/troubleTicket/42
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
  "status": "Resolved",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": " /customerManagement/customer/1234",
      "role": "Originator"
    },
    {
      "href": " /partyManagement/individual/1234",
      "role": "Owner"
    },
    {
      "href": " /partyManagement/individual/2222",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": " /bilManagement/customerBill/1234"
    },
    {
      "involvement": "Adjusted",
      "reference": " /bilManagement/CustomerBill/5678"
    }
```

```
  ],
  "note": [
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
    }
  ]
}
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
  "status": "Resolved",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": "/customerManagement/customer/1234",
      "role": "Originator"
    },
    {
      "href": " /partyManagement/individual/1234",
      "role": "Owner"
    },
    {
```

```
      "href": " /partyManagement/individual/12222",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": " /bilManagement/customerBill/1234"
    },
    {
      "involvement": "Adjusted",
      "reference": " /billManagement/CustomerBill/5678"
    }
  ],
  "note": [
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
    }
  ]
}
```

Note that using PUT with only the "status": " Resolved ", if and href attributes subset will result in all other attributes being set to null or to empty.

Note that id, href and read-only and non-nullable attributes MUST also be specified explicitly.

REQUEST

```
PUT /api/troubleTicket/42
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "status": " Resolved "
}
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
{
    "id": "42",
    "href": " /troubleTicketManagement/troubleTicket/42",
    "status": "Resolved",
    "relatedParty": [],
    "relatedObject": [],
    "note": []
}
```

5.3. Modify Attribute subset of a resource

PATCH MUST be used to update a limited subset of the attributes of an entity as described in RFC5789, using the various PATCH syntax alternatives: * PATCH as per RFC7396 and application/merge-patch+json * JSON PATCH as per RFC6902 and application/json-patch+json * JSON PATCH query as per TMF REST Guidelines Part 5 and application/json-patch-query+json

If the request is successful then the returned code MUST be 200 OK.

If the request is asynchronous then the returned code MUST be 202 Accepted.

The exceptions code must use the exception codes from http://www.iana.org/assignments/http-status-codes/http-status-codes.xml as explained in section 4.3.

HTTP Response Codes - Success	200 OK 202 Accepted
HTTP Response Codes - Failure	400 Bad Request 404 Not Found 405 Method Not Allowed 409 Conflict 500 Internal Server Error

Full representation MUST be included in the response if the request is successful, error code otherwise

with no content.

In case of PATCH if application/json is provided then the default rule will be to apply the same rules as for JSON merge.

Example:

Change the status of the Trouble Ticket with ID= 42 to resolved

REQUEST

```
PATCH /api/troubleTicket/42
Content-Type: application/merge-patch+json
```

```
{
  "status": "Resolved"
}
```

RESPONSE

```
200 OK
Content-Type: Application/json
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
  "status": "Resolved",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": " /customerManagement/customer/1234",
      "role": "Originator"
    },
    {
```

```
      "href": " /partyManagement/individual/1234",
      "role": "Owner"
    },
    {
      "href": " /partyManagement/individual/23333",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": " /billManagement/customerBill/1234"
    },
    {
      "involvement": "Adjusted",
      "reference": " /billManagement/customerBill/5678"
    }
  ],
  "note": [
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
    }
  ]
}
```

5.4. Modify Multi-Valued Attribute

The modification of list based attributes can be performed using:

- PATCH as per RFC7396 and application/merge-patch+json
- JSON PATCH as per RFC6902 and application/json-patch+json
- JSON PATCH query as per TMF REST Guidelines Part 5 and application/json-patch-query+json..

When PATCH is used with Content-Type: application/json the semantic of the operation is the replacement of the whole list with the supplied list.

When JSON PATCH is used the Content-Type MUST be set to Content-Type: application/json-patch+json

and the directives for changing, adding, removing array elements follow the directives stated in the JSON PATCH specification.

The HTTP PATCH method is atomic, as per RFC5789.

If successful, MUST returns a 200 Ok HTTP response code if there is a representation of the modified resource in the payload or a 202 Accepted response code if the operation is performed asynchronous.

Example: Add a related party entry to the relatedParty array of an individual resource.

REQUEST

```
PATCH partyManagement/individual/11
Content-type: application/json-patch+json
```

```
[
  {
    "op": "add",
    "path": "/relatedParty",
    "value": [
      {
        "role": "Employee",
        "href": "/partyManagement/organization/1",
        "validFor": {
          "startDateTime": "2013-04-19T16:42:23-04:00",
          "endDateTime": ""
        },
        "status": "Active"
      }
    ]
  }
]
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
{ ... // JSON Resource Representation with ALL Attributes }
```

Where:

- “op” is the action to take. The following operations are supported: Add, Remove, Replace, Copy, Move, Test
- “path” is a pointer/reference to the field to change.
- “value” is the value to use, this is optional depending on the value of “op”

Note, the path notation used in the Patch standard is JSON Pointer.

5.5. JSON-PATCH extension to manage arrays

JSON Patch Query defines a JSON document structure for expressing a sequence of operations to apply to a JavaScript Object Notation(JSON) document, specifically to manage arrays; it is suitable for use with the HTTP PATCH method.

The "application/json-patch-query+json" media type is used to identify such patch documents.

JSON Patch Query is a format that extends JSON Patch in order to handle partial updates of elements within an array without previous knowledge of the order in which the impacted elements are located within the array.

A JSON Patch Query document follows the same structure as a JSON Patch RFC6902 document, which represents an array of objects, each object representing a single operation to be applied to the target JSON document, but the "path" member includes a query parameter to allow identifying uniquely the element within an array that is impacted by the operation.

The following is an example JSON Patch Query document, transferred in an HTTP PATCH request:

REQUEST

```
PATCH partyManagement/individual/11
Content-type: application/json-patch-query+json
```

```
[
  {
    "op": "remove",
    "path": "/relatedParty?role=Employee"
  },
  {
    "op": "replace",
    "path": "/relatedParty?role=Employee,value=Owner"
  }
]
```

RESPONSE

```
200 OK
Content-Type: application/json-patch+query
```

```
{  ... //_JSON Resource Representation with ALL Attributes_ }
```

Evaluation of a JSON Patch Query document is performed in the same way as for a JSON Patch document.

Refer to TMF REST API Guidelines Part 5 for a complete list of examples of the use of JSON Patch Query document for partial update of a resource.

Chapter 6. Create Resource Patterns

The following section describes the patterns used to create resources.

6.1. Creating a single Resource

POST must be used to create single resources.

Return message MUST contain a resource representation and a Location header with the created resource's URI except when fields= none is provided in the request.

Not all the attributes of the entity must be specified but all the attributes of the created entity MUST be populated by default values (could be null).

A successful creation MUST return a 201 Created HTTP code on success.

A creation operation MUST return a 202 Accepted HTTP response code when the resource creation is asynchronous.

Response Code List - Success	201 Created 202 Accepted
Response Code List - Failure	400 Bad Request 404 Not Found 405 Method Not Allowed 409 Conflict 500 Internal Server Error

If operation is async, a Monitor resources MUST be returned. Please check section on Monitor pattern for more details.

Example:

Create a trouble ticket

REQUEST

```
POST /api/troubleTicket
Content-Type: application/json
```

```
{
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
  "status": "In Progress",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": " /customerManagement/customer/1234",
      "role": "Originator"
    },
    {
      "href": " /partyMangement/individual/1234",
      "role": "Owner"
    },
    {
      "href": " /partyManagement/individual/21333",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": " /billManagement/customerBill/1234"
    },
    {
      "involvement": "Adjusted",
      "reference": " /billManagement/customerBill/5678"
    }
  ],
  "note": [
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
```

```
    }
  ]
}
```

RESPONSE

```
201 Created
Location: https:server:port/troubleTicletManagement/troubleTicket/42
Content-Type: application/json
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
  "status": "In Progress",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": " /customerManagement/customer/1234",
      "role": "Originator"
    },
    {
      "href": " /partyManagement/individual/1234",
      "role": "Owner"
    },
    {
      "href": " /partyManagement/individual/2333",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": " /billManagement/customerBill/1234"
    },
    {
```

```
      "involvement": "Adjusted",
      "reference": " /billManagement/customerBill/5678"
    }
  ],
  "note": [
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
    }
  ]
}
```

The resource created is identified by a service generated URI. Alternatively, an ID can be provided. The ID in the resource representation and the URI are related.

6.2. Creating Multiple Resources

JSON PATCH with application/json-patch+json MUST be used to create multiple resources. POST can't be used for that purpose.

The JSON PATCH operation is relative to the container collection for the entity types to be created. Alternatively, "/path" in the PATCH directive can be used to scope a specific collection. Entities are added to the targeted collection.

Return message MUST contain the resource representation of all the resources created except if fields=none is provided in the request.

When fields=none is used, id and href MUST be specified in the return message.

Not all the attributes of the entity must be specified in the request but all the attributes of the created entity MUST be populated by default values (could be null).

A successful PATCH must return either 200 OK HTTP code or a 204 No Content

An update operation MUST return a 202 Accepted response code if the partial update operation is performed asynchronous.

Response Code List - Success	200 OK 202 Accepted 204 No Content
Response Code List - Failure	400 Bad Request 404 Not Found 405 Method Not Allowed 409 Conflict 500 Internal Server Error

Note that the Internet media type for a JSON Patch document is application/json-patch+json.

The HTTP PATCH method is atomic, as per RFC5789 i.e., all operations MUST be successful otherwise the application of the full PATCH is unsuccessful.

Example:

Create multiple trouble tickets

REQUEST

```
PATCH /api/troubleTicket
Content-type: application/json-patch+json
```

```
[
  {
    "op": "add",
    "path": "/",
    "value": {
      "id": "42",
      "href": " /troubleTicketManagement/troubleTicket/42",
      "correlationId": "TT53482",
      "description": "Customer complaint over last invoice.",
      "severity": "Urgent",
      "@type": "TroubleTicket",
      "creationDate": "2013-07-23T08:16:39.0Z",
      "targetResolutionDate": "2013-07-30T10:20:01.0Z",
```

```
      "status": "In Progress",
      "statusChangeReason": "Waiting for invoicing expert.",
      "statusChangeDate": "2013-07-24T08:55:12.0Z",
      "relatedParty": [
        {
          "href": " /customerManagement/customer/1234",
          "role": "Originator"
        },
        {
          "href": " /partyManagement/individual/1234",
          "role": "Owner"
        },
        {
          "href": " /partyManagement/individual/2333",
          "role": "Reviser"
        }
      ],
      "relatedObject": [
        {
          "involvement": "Disputed",
          "reference": "/customerBill/1234"
        },
        {
          "involvement": "Adjusted",
          "reference": " https:server:port/billManagement/customerBill/5678"
        }
      ],
      "note": [
        {
          "date": "2013-07-24T09:55:30.0Z",
          "author": "Arthur Evans",
          "text": "Already called the expert"
        },
        {
          "date": "2013-07-25T08:55:12.0Z",
          "author": "Arthur Evans",
          "text": "Informed the originator"
        }
      ]
    }
  },
  {
    "op": "add",
    "path": "/",
    "value": {
```

```
    "id": "43",
    "href": " /troubleTicketManagement/troubleTicket/43",
    "correlationId": "TT53483",
    "description": "Customer complaint over last invoice.",
    "severity": "Urgent",
    "type": "TroubleTicket",
    "creationDate": "2013-07-23T08:16:39.0Z",
    "targetResolutionDate": "2013-07-30T10:20:01.0Z",
    "status": "In Progress",
    "statusChangeReason": "Waiting for invoicing expert.",
    "statusChangeDate": "2013-07-24T08:55:12.0Z",
    "relatedParty": [
      {
        "href": " /customerManagement/customer/1234",
        "role": "Originator"
      },
      {
        "href": " /partyManagement/individual/1234",
        "role": "Owner"
      },
      {
        "href": " /partyManagement/individual/2344",
        "role": "Reviser"
      }
    ],
    "relatedObject": [
      {
        "involvement": "Disputed",
        "reference": " /bilManagement/customerBill/1234"
      },
      {
        "involvement": "Adjusted",
        "reference": " /billManagement/customerBill/5678"
      }
    ],
    "note": [
      {
        "date": "2013-07-24T09:55:30.0Z",
        "author": "Arthur Evans",
        "text": "Already called the expert"
      },
      {
        "date": "2013-07-25T08:55:12.0Z",
        "author": "Arthur Evans",
        "text": "Informed the originator"
```

```
        }
      ]
    }
  }
]
```

RESPONSE

```
[
  {
    "id": "42",
    "href": " /troubleTicketManagement/troubleTicket/42",
    "correlationId": "TT53482",
    "description": "Customer complaint over last invoice.",
    "severity": "Urgent",
    "@type": "TroubleTicket",
    "creationDate": "2013-07-23T08:16:39.0Z",
    "targetResolutionDate": "2013-07-30T10:20:01.0Z",
    "status": "In Progress",
    "statusChangeReason": "Waiting for invoicing expert.",
    "statusChangeDate": "2013-07-24T08:55:12.0Z",
    "relatedParty": [
      {
        "href": " /customerManagement/customer/1234",
        "role": "Originator"
      },
      {
        "href": " /partyManagement/individual/1234",
        "role": "Owner"
      },
      {
        "href": " /partyManagement/individual/2333",
        "role": "Reviser"
      }
    ],
    "relatedObject": [
      {
        "involvement": "Disputed",
        "reference": "/customerBill/1234"
      },
      {
        "involvement": "Adjusted",
        "reference": " https:server:port/billManagement/customerBill/5678"
      }
```

```
    ],
    "note": [
      {
        "date": "2013-07-24T09:55:30.0Z",
        "author": "Arthur Evans",
        "text": "Already called the expert"
      },
      {
        "date": "2013-07-25T08:55:12.0Z",
        "author": "Arthur Evans",
        "text": "Informed the originator"
      }
    ]
  },
  {
    "id": "43",
    "href": " /troubleTicketManagement/troubleTicket/43",
    "correlationId": "TT53483",
    "description": "Customer complaint over last invoice.",
    "severity": "Urgent",
    "type": "TroubleTicket",
    "creationDate": "2013-07-23T08:16:39.0Z",
    "targetResolutionDate": "2013-07-30T10:20:01.0Z",
    "status": "In Progress",
    "statusChangeReason": "Waiting for invoicing expert.",
    "statusChangeDate": "2013-07-24T08:55:12.0Z",
    "relatedParty": [
      {
        "href": " /customerManagement/customer/1234",
        "role": "Originator"
      },
      {
        "href": " /partyManagement/individual/1234",
        "role": "Owner"
      },
      {
        "href": " /partyManagement/individual/2344",
        "role": "Reviser"
      }
    ],
    "relatedObject": [
      {
        "involvement": "Disputed",
        "reference": " /bilManagement/customerBill/1234"
      },
```

```
      {
        "involvement": "Adjusted",
        "reference": " /billManagement/customerBill/5678"
      }
    ],
    "note": [
      {
        "date": "2013-07-24T09:55:30.0Z",
        "author": "Arthur Evans",
        "text": "Already called the expert"
      },
      {
        "date": "2013-07-25T08:55:12.0Z",
        "author": "Arthur Evans",
        "text": "Informed the originator"
      }
    ]
  }
]
```

The resources created are identified by a service generated URI. Alternatively, an ID can be provided. The ID in the resource representation and the URI are related.

6.3. Create a single resource with attribute selection

Attribute selection MAY be used with POST operation.

Example:

REQUEST

```
POST /api/troubleTicket?fields=none
Content-Type: application/json
```

```
{
  "id": "42",
  "href": " /troubleTicketManagement/troubleTicket/42",
  "correlationId": "TT53482",
  "description": "Customer complaint over last invoice.",
  "severity": "Urgent",
  "@type": "TroubleTicket",
  "creationDate": "2013-07-23T08:16:39.0Z",
  "targetResolutionDate": "2013-07-30T10:20:01.0Z",
```

```
  "status": "In Progress",
  "statusChangeReason": "Waiting for invoicing expert.",
  "statusChangeDate": "2013-07-24T08:55:12.0Z",
  "relatedParty": [
    {
      "href": " /customerManagement/customer/1234",
      "role": "Originator"
    },
    {
      "href": " /partyManagement/individual/1234",
      "role": "Owner"
    },
    {
      "href": " /partyMangement/individual/1222",
      "role": "Reviser"
    }
  ],
  "relatedObject": [
    {
      "involvement": "Disputed",
      "reference": " http:server:port/billManagement/customerBill/1234"
    },
    {
      "involvement": "Adjusted",
      "reference": " http:server:port/billManagement/CustomerBill/5678"
    }
  ],
  "note": [
    {
      "date": "2013-07-24T09:55:30.0Z",
      "author": "Arthur Evans",
      "text": "Already called the expert"
    },
    {
      "date": "2013-07-25T08:55:12.0Z",
      "author": "Arthur Evans",
      "text": "Informed the originator"
    }
  ]
}
```


RESPONSE

```
201 Created
Location: http:server:port/troubleTicletManagement/troubleTicket/42
Content-Type: application/json
```

```
{
  "id": "42",
  "href":" /troubleTicketManagement/troubleTicket/42"
  ... // Full representation of the resource
}
```

Chapter 7. Delete Resource Pattern

A Delete operation on a resource is used to remove a new instance of a resource.

A Delete uses the HTTP method DELETE on a Resource URI, for example:

REQUEST

```
DELETE /troubleTicketManagement/troubleTicket/42
```

RESPONSE

```
204 No Content
```

A Delete operation MUST use an HTTP DELETE on a resource.

A Delete operation MUST NOT accept a request payload.

A Delete operation MUST return exactly one resource.

If a DELETE method is successfully applied, the origin server SHOULD send

- a 202 (Accepted) status code if the action will likely succeed but has not yet been enacted,
- a 204 (No Content) status code if the action has been enacted and no further information is to be supplied,

or

a 200 (OK) status code if the action has been enacted and the response message includes a representation describing the status.

Chapter 8. Task Resource Pattern

This section describes the use of Task resources to expose complex operations that are not easily or not decomposable to CRUD Entity based operations. Examples include:

- Address Validation
- Prepay Balance top-up
- Order Cancelation
- Alarm Clearing

8.1. Modeling Complex operations with task resources

If proper REST design is limited during translation from action-based operational interfaces to REST interfaces **it is allowable** to define a resource for the operation and POST to that operation to execute.

A TASK resource name should be a **verb** representing the task to be executed. TASK creation may result in the creation of a number of related resources or other tasks. E.g. validate, check, heal, migrate, clear, close, transfer, etc.

A task operation MUST return:

- a 200 OK if successful and the full representation,
- a 202 Accepted if the operation is performed asynchronous or
- a 204 No content otherwise.

Response Code List - Success	200 OK 202 Accepted 204 No content
Response Code List - Failure	400 Bad Request 404 Not Found 405 Method Not Allowed 409 Conflict 500 Internal Server Error

There are use cases:

- when a task request is simple and there is no need to persist the request but the server may choose to persist the request in any case, e.g. for accounting purposes
- long running task when there is a need to persist the request in order to monitor it.

Simple task request MAY return an ID.

If an ID is returned it MUST represent a resource that was created for the task.

Iterators MAY be supported on simple task.

Example:

REQUEST

```
POST /alarmManagement/alarm/123/clear
Content-Type: application/json
```

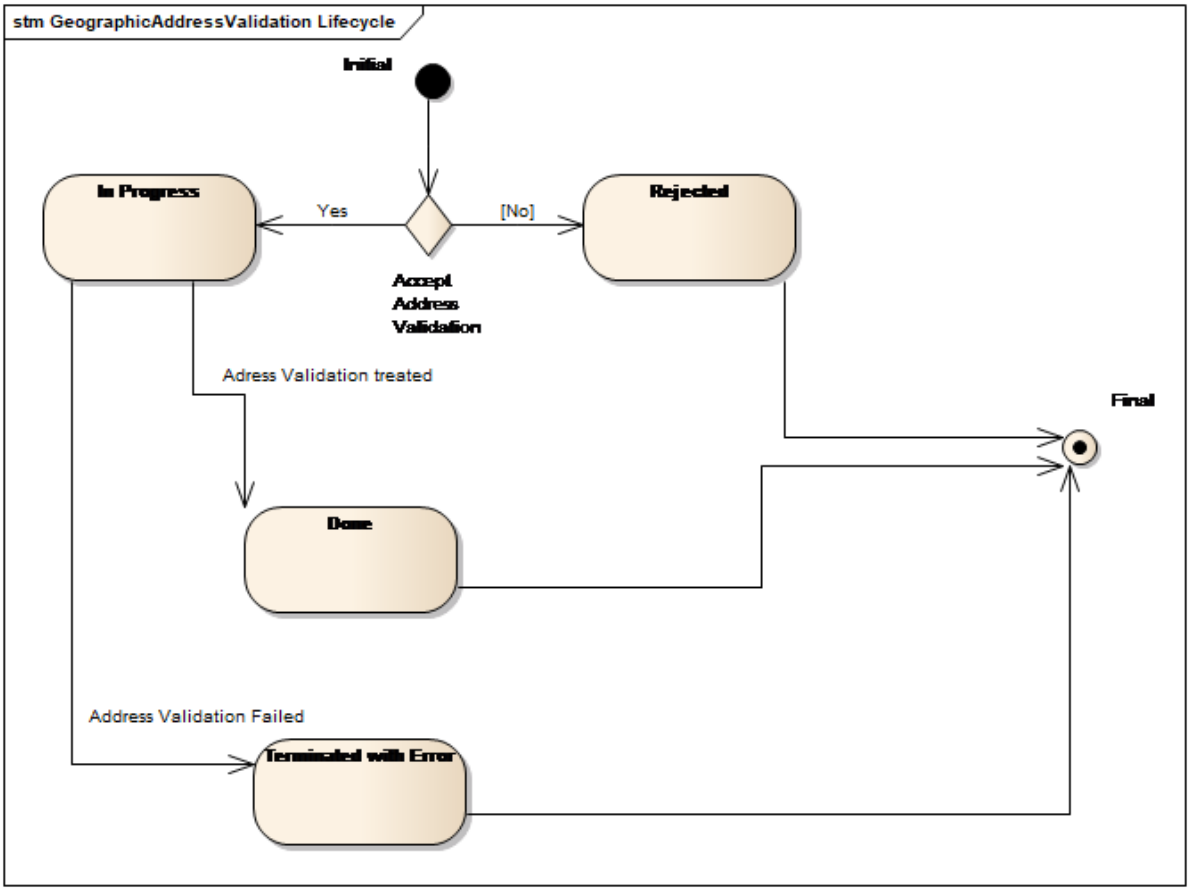
```
{
......
}
```

RESPONSE

```
204 No Content
```

Long Running TASKS SHOULD have a state.

Example:



Long running TASKS MUST return an ID and href.

It is not recommended to delete a task but rather to cancel the task with PATCH.

Example:

REQUEST

```
POST /geographicAddressManagement/geographicAddressValidation
Content-Type: application/json
```

```
{
  "provideAlternative": true,
  "submittedGeographicAddress": {
    "streetName": "Denfert-Rocheraut",
    "streetType": "rue",
    "postcode": "69004",
    "city": "Lyon",
    "stateOrProvince": "Rhone",
    "country": "France"
  }
}
```

RESPONSE

```
200 OK
```

```
{
  "id": "8318",
  "href": "/geographicAddressManagement/geographicAddressValidation/8318"
}
```

8.2. Using Task with iterator based response

There are scenarios when there MAY be a need to iterate on the response part of the TASK (e.g. iterate on array of the response of the TASK).

The iterator resource SHOULD include as minimum the sub-resource within the response that need to iterate on.

Iterator resource example:

```
[
  {
    "id": "1960",
    "href": " /location/geographicAddress/1960",
    "streetNr": "2",
    "streetName": "Denfert-Rocheraut",
    "streetType": "rue",
    "postcode": "69004",
    "locality": "Lyon",
    "city": "Lyon",
```

```
    "stateOrProvince": "Rhone",
    "country": "France",
    "geographicLocationReforValue": {
      "id": "7200",
      "href": " /location/geographicLocation/7200"
    }
  },
  {
    "id": "8965",
    "href": " /location/geographicAddress/8965",
    "streetNr": "4",
    "streetName": "Denfert-Rocheraut",
    "streetType": "rue",
    "postcode": "69004",
    "locality": "Lyon",
    "city": "Lyon",
    "stateOrProvince": "Rhone",
    "country": "France",
    "geographicLocationRefOrValue": {
      "id": "7204",
      "href": " /location/geographicLocation/7204"
    }
  },
  {
    "id": "7854",
    "href": " /location/geographicAddress/7854",
    "streetNr": "6",
    "streetName": "Denfert-Rocheraut",
    "streetType": "rue",
    "postcode": "69004",
    "locality": "Lyon",
    "city": "Lyon",
    "stateOrProvince": "Rhone",
    "country": "France",
    "geographicLocationRefOrValue": {
      "id": "7263",
      "href": " /location/geographicLocation/7263"
    }
  }
]
```


Example POST …/validate:

REQUEST

```
POST /geographicAddressManagement/geographicAddressValidation
Accept: application/json
```

RESPONSE

```
201 Created
Content-Type: application/json
Link:
<https://host:port:location/geographicAddressManagement/geographicAddressValidati
on/8318/iterator>
```

```
{
  "id": "8318",
  "href": "/geographicAddressManagement/geographicAddressValidation/8318",
  "alternateGeographicAddress ": [
    {
      "id": "1960",
      "href": " /location/geographicAddress/1960",
      "streetNr": "2",
      "streetName": "Denfert-Rocheraut",
      "streetType": "rue",
      "postcode": "69004",
      "locality": "Lyon",
      "city": "Lyon",
      "stateOrProvince": "Rhone",
      "country": "France",
      "geographicLocationReforValue": {
        "id": "7200",
        "href": " /location/geographicLocation/7200"
      }
    },
    {
      "id": "8965",
      "href": " /location/geographicAddress/8965",
      "streetNr": "4",
      "streetName": "Denfert-Rocheraut",
      "streetType": "rue",
      "postcode": "69004",
      "locality": "Lyon",
      "city": "Lyon",
      "stateOrProvince": "Rhone",
      "country": "France",
```

```
      "geographicLocationRefOrValue": {
        "id": "7204",
        "href": "/location/geographicLocation/7204"
      }
    },
    {
      "id": "7854",
      "href": " /location/geographicAddress/7854",
      "streetNr": "6",
      "streetName": "Denfert-Rocheraut",
      "streetType": "rue",
      "postcode": "69004",
      "locality": "Lyon",
      "city": "Lyon",
      "stateOrProvince": "Rhone",
      "country": "France",
      "geographicLocationRefOrValue": {
        "id": "7263",
        "href": " /location/geographicLocation/7263"
      }
    }]
}
```

```
GET /iterator
```

REQUEST

```
GET /geographicAddressManagement/ geographicAddressValidation/8318/iterator
Accept: application/json
```

RESPONSE

```
206 Partial Content
Content-Type: application/json
X-Total-Count:10
```

Link: </geographicAddressManagement/validate/8313/iterator?offset=0&limit=3>; rel="first", </geographicAddressManagement/validate/8313/iterator?offset=3&limit=6>; rel="next", </geographicAddressManagement/validate/8313/iterator?offset=0&limit=3>; rel="prev", </geographicAddressManagement/ validate/8313/iterator?offset=6&limit=9>; rel="last"

```
[
  {
    "id": "1960",
    "href": " /location/geographicAddress/1960",
    "streetNr": "2",
    "streetName": "Denfert-Rocheraut",
    "streetType": "rue",
    "postcode": "69004",
    "locality": "Lyon",
    "city": "Lyon",
    "stateOrProvince": "Rhone",
    "country": "France",
    "geographicLocationReforValue": {
      "id": "7200",
      "href": " /location/geographicLocation/7200"
    }
  },
  {
    "id": "8965",
    "href": " /location/geographicAddress/8965",
    "streetNr": "4",
    "streetName": "Denfert-Rocheraut",
    "streetType": "rue",
    "postcode": "69004",
    "locality": "Lyon",
    "city": "Lyon",
    "stateOrProvince": "Rhone",
    "country": "France",
    "geographicLocationRefOrValue": {
      "id": "7204",
      "href": " /location/geographicLocation/7204"
    }
  },
  {
    "id": "7854",
    "href": "https://host:port/location/geographicAddress/7854",
    "streetNr": "6",
    "streetName": "Denfert-Rocheraut",
    "streetType": "rue",
    "postcode": "69004",
    "locality": "Lyon",
    "city": "Lyon",
    "stateOrProvince": "Rhone",
    "country": "France",
    "geographicLocationRefOrValue": {
```

```
      "id": "7263",
      "href": " /location/geographicLocation/7263"
    }
  }
]
```

Chapter 9. Monitor pattern

The monitor resource is used to monitor the execution of async requests on specific resource.

The monitor resource MUST support HTTP GET method.

Supporting Async requests are optional. If supported, then, monitor or notification pattern MUST be used.

A monitor resource SHOULD be defined to support a monitor pattern.

Monitor resource MUST have at least the following attributes:

Field	Description
id	Identifier of an instance of the monitor. Required to be unique within the resource
type	Used in URIs as the identifier for specific instances of a type
name	Name of resource type i.e. monitor
request	Represents the request
response	Represents the response
state	The monitor state of the resource. InProgress, InError, Completed
href	The reference to this monitor
sourceHref	The monitored resource href

PUT/POST/PATCH/DELETE operations SHOULD not be supported.

REQUEST

```
PUT|POST|PATCH|DELETE /api/…/monitor
Accept: application/json
```

RESPONSE

```
405 Method Not Allowed
```

The monitors can be accessed via the Monitor collection or a singleton under their source for example:

api/activation/service/47/monitor.

Returns HTTP/1.1 status code 200 OK if the request was successful.

On failure, appropriate error code SHOULD be returned.

The monitor resources SHOULD support at least the following states: in progress, success, failed.

Structure of the monitor can be extended using the extension patterns described in Part 2 of the guidelines.

Example:

The response to POST /activation/service operation cannot be sent synchronously, a monitor resource hyperlink is given in the response

REQUEST

```
POST /api/activation/service
Accept: application/json
```

```
{
  "state": "Active",
  "serviceSpecification": {
    "id": "conferenceBridgeEquipment",
    "href": " /catalogManagement/serviceSpecification/conferenceBridgeEquipment",
    "serviceCharacteristic": [
      {
        "name": "numberOfVc500Units",
        "value": "1"
      },
      {
        "name\"": "numberOfVc100Units",
        "value": "2"
      },
      {
        "name": "routerType",
        "value": "CiscoASR1000"
      },
      {
        "name": "powerSupply",
        "value": "UK"
      }
    ]
  }
}
```

RESPONSE

```
202 Accepted
Content-Type: Application/JSON
Location: http://server/api/activation/service/14

{
      ... // same as in request
}

Link: < **api/activation/monitor/1514**>;rel=related;title=monitor,
< /api/activation/service/14>;rel=self,
< /api/activation/service/14>;rel=canonical
```

Example: GET a monitor.

REQUEST

```
GET /api/activation/monitor/1514
Content-type: application/json
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
{
    "id": "1514",
    "state": "Completed",
    "type": "monitor",
    "request": {
        "method": "POST",
        "to": "",
        "body": {
            "state": "Active",
            "serviceSpecification": {
            "id": "conferenceBridgeEquipment",
            "href": "
/catalogManagement/serviceSpecification/conferenceBridgeEquipment"
            },
            "serviceCharacteristic": [
            {
                "name": "numberOfVc500Units",
                "value": "1"
            },
            {
                "name": "numberOfVc100Units",
                "value": "2"
            },
            {
                "name": "routerType",
                "value": "CiscoASR1000"
            },
            {
                "name": "powerSupply",
                "value": "UK"
            }
        ]
```

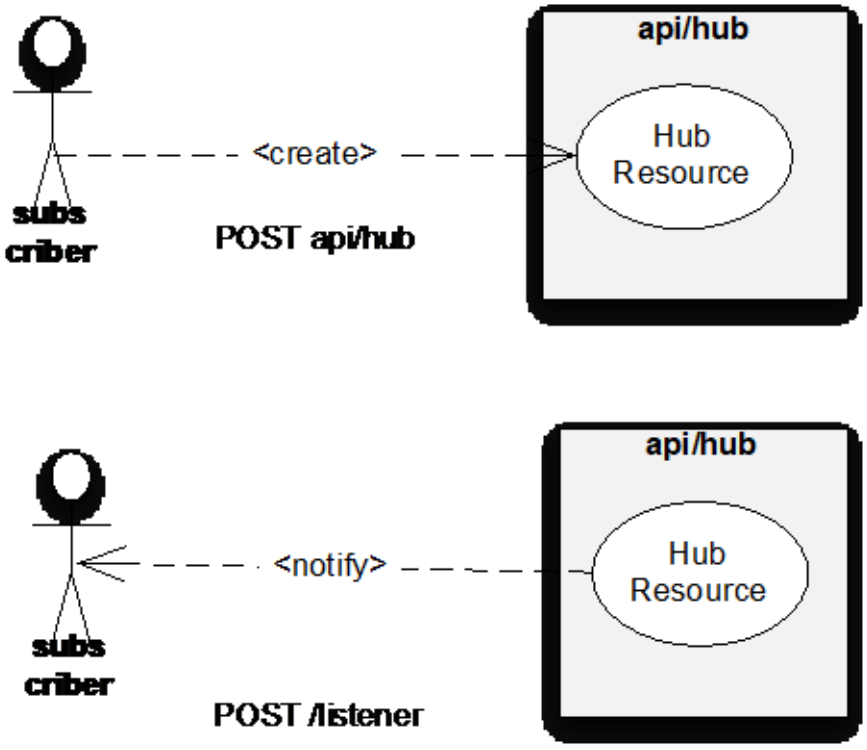
```
        },
        "header": [
            {
            "name": "Accept",
            "value": "application/json"
            }
            ]
        },
    "response": {
        "statusCode": "202",
        "body": {
            "state": "Active",
            "serviceSpecification": {
                "id": "conferenceBridgeEquipment",
                "href": "
/catalogManagement/serviceSpecification/conferenceBridgeEquipment"
            },
            "serviceCharacteristic": [
                {
                        "name": "numberOfVc500Units",
                        "value": "1"
                    },
                    {
                        "name": "numberOfVc100Units",
                        "value": "2"
                    },
                    {
                        "name": "routerType",
                        "value": "CiscoASR1000"
                    },
                    {
                        "name": "powerSupply",
                        "value": "UK"
                    }
                ]
            },
        "header": [
            {
                "name": "Content-Type",
                "value": "application/json"
            },
            {
                "name": "Link",
                "value":
"<http://server/api/activation/monitor/1514>;rel=related;title=monitor,<http://se
```

```
rver/api/activation/service/14>;rel=self,<http:
//server/api/activation/service/14>;rel=canonical"
            }
        ]
    },
    "href": " /activation/monitor/1514",
    "sourceHref": " /activation/service/14"
}
```

Chapter 10. Notification Patterns

The following section describes the publish subscribe pattern supported by REST based APIs supporting eventing.

A hub can be created at the API level or at each API resource level within an API.



10.1. Register Listener

The registration of a listener is done by creating a HUB resource unique to the listener (equivalent of a subscription). The HUB resource is attached or bound to the API and its attribute specify the POST event callback address of the listener.

The hub is created via a POST api/hub call.

The POST call sets the communication endpoint address the service instance must use to deliver notifications (by default on all supported events). Note that a query expression may be supplied. The query expression may be used to filter specific event types and/or any content of the event. The query expression structure used for notification filtering is the same than the one used for queries i.e. GET.

Subsequent POST calls may be rejected by the service if it does not support multiple listeners. In this case DELETE /api/hub/{id} must be called before the endpoint can be created again.

Returns HTTP/1.1 status code 204 No Content if the request was successful.

Returns HTTP/1.1 status code 409 Conflict if request is not successful.

Example: Create a Hub to receive events from the Trouble Ticket API on the “http://in.listener.com"

The resource model for the listeners described in each API under “notification listeners (client side)” are just examples.

REQUEST

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

```
{
"callback": "http://in.listener.com"
}
```

RESPONSE

```
201 Created
Content-Type: application/json
Location: /troubleTicket/hub/42
```

```
{
  "id": "42",
  "callback": "http://in.listener.com"
}
```

Example: Get a Hub

REQUEST

```
GET /troubleTicket/hub/42
Accept: application/json
```

RESPONSE

```
200 OK
Content-Type: application/json
```

```
[
  {
    "id": "42",
    "callback": "http://in.listener.com "
  }
]
```

Example: Subscribe to specific eventType.

REQUEST

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

```
{
  "callback": "http://in.listener.com",
  "query": "eventType=TroubleTicketStateChangeNotification"
}
```

RESPONSE

```
201 Created
Content-Type: application/json
Location: /api/hub/42
```

```
{
  "id": "42",
  "callback": "http://in.listener.com",
  "query": "eventType=TroubleTicketStateChangeNotification"
}
```

10.2. Unregister Listener

To unregister a listener the HUB resource corresponding to the listener must be destroyed.

```
DELETE hub/\{id}
```

This clears the communication endpoint address that was set by creating the Hub.

On successful deletion, returns HTTP status 200 (OK) along with a response body or returns HTTP status 204 (NO CONTENT) with no response body.

For the HTTP status 204 the HTTP payload returns no data.

On unsuccessful deletion, the HTTP status 404 (NOT FOUND) can be returned. The 404 (NOT FOUND) status means that the server has not found anything matching the request URI.

REQUEST

```
DELETE /api/hub/\{id}
Accept: application/json
```

RESPONSE

```
204 No Content
```

10.3. Publishing Events

Publishing an event is done by posting the event to the listener address.

The structure of the event is:

```
{
  "eventId":"eventId",
  "eventTime":"event Time",
  "eventType":"event Type",
  "event": {
    "resource" :
    { ... // RESOURCE SAMPLE }
    }
  }
```

Returns HTTP/1.1 status code 201 Created

For example, posting a TroubleTicket event:

REQUEST

```
POST /client/listener
Accept: application/json
```

```
{
  "eventId": "00001",
  "eventTime": "2015-11-16T16:42:25-04:00",
  "eventType": "TroubleTicketStateChangeNotification",
  "event": {
    "troubleTicket": {
      "id": "43",
      "href": " /troubleTicketManagement/troubleTicket/43",
      "correlationId": "TT53483",
      "description": "Customer complaint over last invoice.",
      "severity": "Urgent",
      "@type": "TroubleTicket",
      "creationDate": "2013-07-23T08:16:39.0Z",
      "targetResolutionDate": "2013-07-30T10:20:01.0Z",
      "status": "In Progress",
      "statusChangeReason": "Waiting for invoicing expert.",
      "statusChangeDate": "2013-07-24T08:55:12.0Z",
      "relatedParty": [
        {
          "href": " /customerManagement/customer/1234",
          "role": "Originator"
        },
        {
          "href": " /partyManagement/individual/1234",
          "role": "Owner"
        },
        {
          "href": " /partyManagement/individual/2334",
          "role": "Reviser"
        }
      ],
      "relatedObject": [
        {
          "involvement": "Disputed",
          "reference": " https:server:port/billManagement/customerBill/1234"
        },
        {
          "involvement": "Adjusted",
          "reference": " https:server:port/billManagement/customerBill/5678"
        }
```

```
      ],
      "note": [
        {
          "date": "2013-07-24T09:55:30.0Z",
          "author": "Arthur Evans",
          "text": "Already called the expert"
        },
        {
          "date": "2013-07-25T08:55:12.0Z",
          "author": "Arthur Evans",
          "text": "Informed the originator"
        }
      ]
    }
  }
}
```

10.4. Content type filtering

Provide example on how we can filter events based on their content.

REQUEST

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

```
{
  "callback": "http://in.listener.com",
  "query":"eventType =
TroubleTicketStateChangeNotification&event.troubletTicket.severity=Urgent&fields=
event.troubleTicket.id,event.TroubleTicket.name,event.troubleTicket.severity"
}
```

RESPONSE

```
201 Created
Content-Type: application/json
```

```
{
  "id": "string",
  "callback": "http://in.listener.com",
  "query": "eventType \u003d
TroubleTicketStateChangeNotification\u0026event.troubletTicket.severity\u003dUrge
nt\u0026fields\u003devent.troubleTicket.id,
event.TroubleTicket.name,\nevent.troubleTicket.severity"
}
```

Chapter 11. Versioning

11.1. API Versioning

REST APIs MUST state version with “v” following the API Name, e.g.: APIName/v1/resource.

The schema associated with a REST API must have its version number aligned with that of the REST API.

The version number has major, minor and revision numbers. E.g. v1.0.0

The version number (without the revision number) is held in the URI. E.g troubleTicketManagement/v1/ticket

The major version number is incremented for an incompatible change.

The minor version number is incremented for a compatible change.

For minor modifications of the API, version numbering must not be updated, provided the following backward compatibility rules are respected:

- New elements in a data type must be optional (minOccurs=0)
- Changes in the cardinality of an attribute in a data type must be from mandatory to optional or from lower to greater
- New attributes defined in an element must be optional (absence of use=”required”).
- If new enumerated values are included, the former ones and its meaning must be kept.
- If new operations are added, the existing operations must be kept
- New parameters added to existing operations must be optional and existing parameters must be kept

For major modifications of the API, not backward compatible and forcing client implementations to be changed, the version number must be updated.

The format for the API version number is defined as:

```
\{serverRoot}/\{apiName}/\{apiVersion}
```

where

{apiName} is the name of the API{apiVersion} is the version of the API (e.g. v1) {serverRoot} is implementation specific (e.g.: https://api.service.company.com)

The versioning is applied uniformly to all the entities under a versioned API. That is, it is assumed that entities under the management scope of the API are all aligned with the same version of the SID for

example.

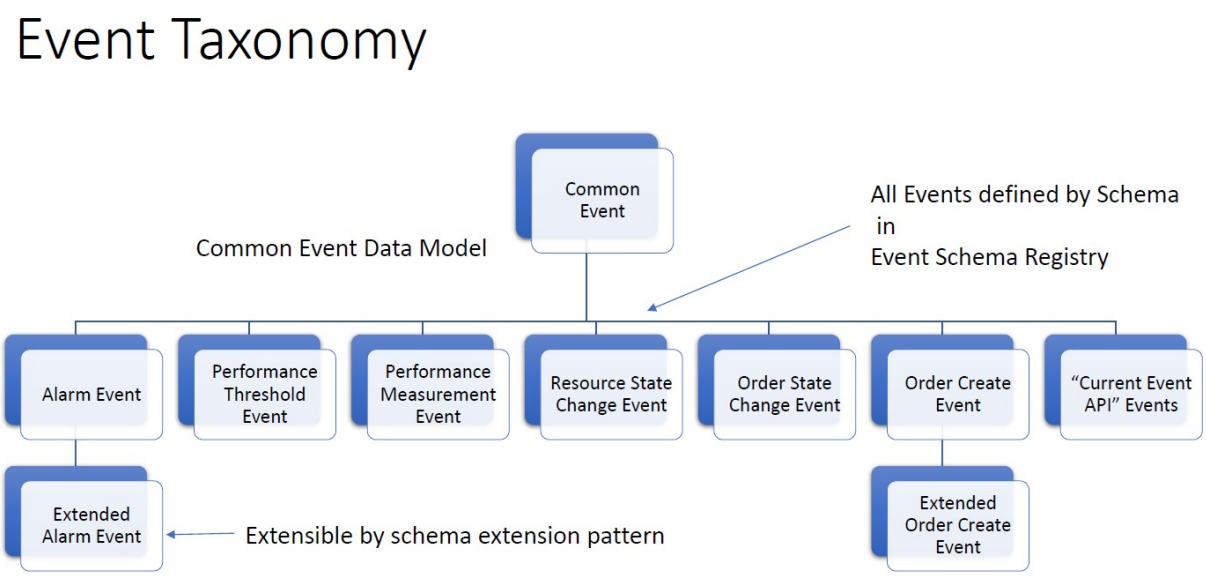
Chapter 12. Event management

12.1. Decoupling

Event Management supports decoupling directly. It decouples TMF APIs from each other and powers decoupling on domain and API level.

12.2. Event API relationship to Notification events by a generalized event model

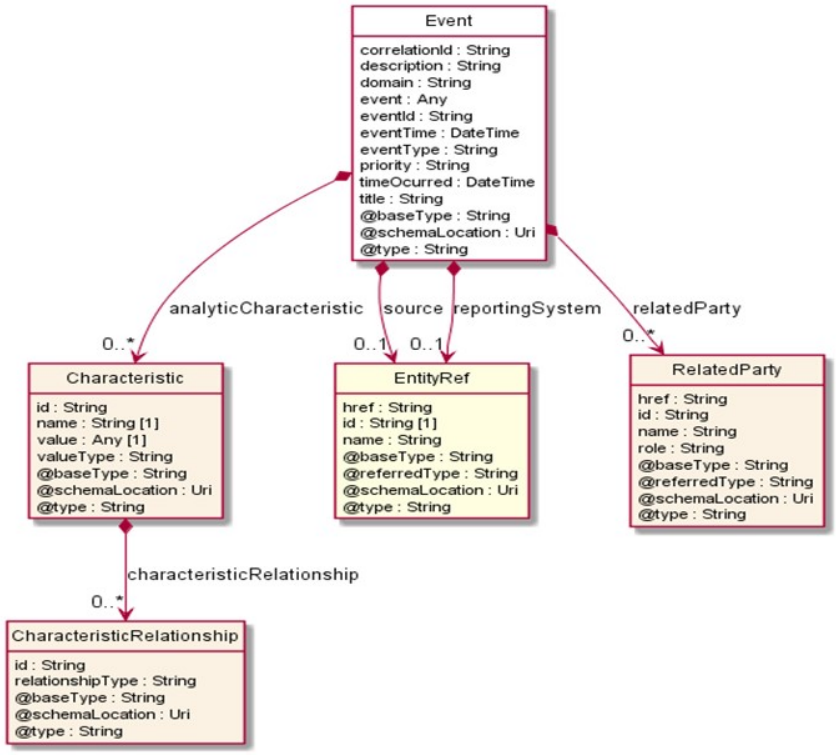
Support of polymorphic collections and types and schema based extension is provided by means of a list of generic meta-attributes that are described below. Polymorphism in collections occurs when entities inherit from base entities, for Instance the “Alarm Event” or the “Order Create Event” or any kind of event payload can be described the event entity. This means the common event data model can be extended by any kind of event payload because of the polymorphic data structure definition. By that polymorphic pattern the common event data model describes the event taxonomy of any event type.



Generic support of polymorphism and pattern extensions is described in the TMF API Guidelines Part 2 document.

The @type attribute provides a way to represent the actual class type of an entity. For example, within a list of Event instances some may be instances of Alarm Events where other could be instances of OrderCreation Events. The @type gives this information. All resources and sub-resources of this API have a @type attributes that can be provided when this is useful.

12.3. Event Resource Model



The “event” attribute element contains the payload of the sourced entity resource (e.g. Alarm, Resource, Order..).

The “eventType” attribute contains the triggered event type (e.g. alarmCreateEvent, resourceAttributeValueChangeEvent,..) thrown by the notification interface of the TMF Open API.

Additional attributes (e.g. “domain”, “reportingSystem”, “source”, “relatedParty”,..) describe the complete Sourcing of the Event.

Especially by the @type and @schemaLocation of the “event” (eventPayload) the Event taxonomy is described by TMF Data Model (schemas) and can be used for further processing (e.g. Analytics).

As example the Event “alarmCreateNotification” is given below:

|**AlarmNotification**

```
{
  "@type": "event",
  "@schemaLocation": "http://xx/Event.schema.json",
  "@baseType": "event",
  "eventId": "256c42f0-7cae-4cfe-8b96-f5773796f8ff",
  "eventTime": "2019-11-02T14:20:54",
  "eventType": "AlarmCreateNotification",
```

```
  "correlationId": "238764827364827t367",
  "domain": "domain-x",
  "title": "Mail service not responding",
  "description": "The mail service is no longer responding and sending mails",
  "timeOccurred": "2019-11-02T14:20:54",
  "timeReceived": "2019-11-02T14:21:08",
  "priority": "Normal",
  "source": {
    "id": "12345",
    "href": ".../relatedEntity/12345"
  },
  "reportingSystem": {
    "id": "34534",
    "href": "/reportingEntity/12345",
    "name": "name"
  },
  "relatedParty": [
    {
      "id": "12312",
      "href": "/party/12345",
      "role": "Owner"
    }
  ],
  "event": {
    "alarm": {
      "@type": "Alarm",
      "@schemaLocation": "http://../registry/Alarm.schema.json",
      "@baseType": "Alarm",
      "id": "ROUTER_IF@Cisco-7609-6-4-4-4-14-14-4--Gi9/20@42",
      "href": "http://api/alarm/ROUTER_IF@Cisco-7609-6-4-4-4-14-14-4--Gi9/20@42",
      "externalAlarmId": "cisco-7609-1937465789",
      "alarmType": "QualityOfServiceAlarm",
      "perceivedSeverity": "CRITICAL",
      "probableCause": "Threshold crossed",
      "specificProblem": "Inbound Traffic threshold crossed"
    }
  }
}
```


12.4. Topic/Event Resource Graph

Event Management supports the Event streaming concept by the Topic resource. The Topic resource is the target container for the event resource. The topic/event resource can be used for storing different events into different topics separated into domains. (e.g. Party, Resource, Product,..) A topic/event

resource graph can be set-up for further processing and analytic functions.

The different topics can also be used for different access control on the topic/event resource because of the different endpoints of the Event API by topics.

Example: Creation of a “Resource” topic for Alarm events:

REQUEST

```
POST //eventhub/eventapi/topic
```

```
{
  "contentQuery": "AlarmValue",
  "headerQuery": "AlarmType",
  "name": "Resource",
  "@baseType": "topic",
  "@schemaLocation": "http://schemaregistry/Topic.schema.json",
  "@type": "topic"
}
```

RESPONSE

```
201 Created
Content-Type: application/json
```

```
{
  "id": "13",
  "href": "http://eventhub/eventapi/topic/13",
  "contentQuery": "AlarmValue",
  "headerQuery": "AlarmType",
  "name": "Resource",
  "@baseType": "topic",
  "@schemaLocation": "http://schemaregistry/Topic.schema.json",
  "@type": "topic"
}
```

12.5. Event Management as API for Consumers

Event Management has, like all other APIs, a consumer API. That consumer API allows Consumers to have access to the different topic/event resources into the graph. Different topics might provide content

for different Consumers.

e.g. the “Party/TroubleTicket” Topic contains TroubleTicket events for the Consumers service support.

12.6. Event Management in a multi cloud scenario

Event Management supports the sourcing and storing of any API notification. It allows storing and consuming events (data) as streams.

For example storing the complete PhysicalResource data replication (copy) asstream of events into topic “Resource” (channel/domain):

POST //eventhub/topic/{topicId}/event (e.g. topicId = 111, topic.name = Resource)

The corresponding Resource Inventory API should support the data replication by a resource data export operation according to the event notification interface. (s. TMF639_Inventory_Resource specification)

The Event stream can be consumed by list of events selected by the offset keyword.

List of events as a stream (example):

```
GET
/eventhub/topic/111/event?evenType=PhysicalResourceDataReplication&offset=10000&l
imit=1000
```

Or by eventTime

```
GET
/eventhub/topic/111/event?evenType=PhysicalResourceDataReplication&eventTime>=20-
02-2020:12:00:00&limit=1000
```

It allows the usage of the Event API into a multi cloud scenario, where every cloud might contain an Event Management component.

Because of the unified and polymorph designed Event API a distribution of events in any cloud container is possible and can support decoupling multi cloud APIs by event (data) distribution and replication.

Event management supports data replication into the multi cloud environment by event streaming.

Chapter 13. 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.

13.1. Document History

13.1.1. 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
0.1	23-Jun-2013	Pierre Gauthier, TM Forum	Description e.g. first issue of document
0.2	July 2013	Tina O'Sullivan	Updated branding
0.3	8-Oct-2013	Pierre Gauthier	Further updates
1.0	9-Oct-2013	Tina O'Sullivan	Minor corrections
1.1.0	November 2014	Pierre Gauthier	Updates for Fx14.5
1.1.1	March 2015	Alicja Kawecki	Updated cover, footer and Notice to reflect TM Forum Approved status
3.0.0	November 2017	Nicoleta Stoica	Updates for Fx 17.5
3.0.1	12 Dec 2017	Adrienne Walcott	Formatting/style edits prior to publishing
3.0.2	20 Mar 2018	Adrienne Walcott	Updated to reflect TM Forum Approved Status
4.0.0	23 April 2020	Several contributors / Pierre Gauthier	Updated typos and some , add security, event
4.0.1	20 Juli 2020	Adrienne Walcott	Updated to reflect TM Forum Approved Status
4.0.2	13 January 2021	Several contributors / Pierre Gauthier	Fix authentication diagram and some RFC references

13.1.2. 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 Status	Date Modified	Modified by:	Description of changes
17.5.0	08-Nov-2017	Nicoleta Stoica	first release of document
17.5.1	20-Mar-2018	Adrienne Walcott	Updated to reflect TM Forum Approved Status
Pre-production	14-May-2020	Alan Pope	Minor edits prior to publication
Production	20-Jul-2020	Adrienne Walcott	Updated to reflect TM Forum Approved Status
Pre-production	13-Jan-2021	Pierre Gauthier	Team Approved
Production	24-May-2021	Adrienne Walcott	Updated to reflect TM Forum Approved Status

13.2. Acknowledgments

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

- Pierre Gauthier, TM Forum, **Editor** and Team Leader
- Nicoleta Stoica, Vodafone, Author
- Luis Velarde, Telefonica, Contributor
- Kamal Maghsoudlou, Ericsson, Contributor
- Christophe Michel, Amdocs, Contributor
- Hongxia Hao, Huawei, Contributor,
- Jonathan Goldberg, Amdocs, Contributor
- Fengjing Tan, Huawei, Contributor
- Anoop Gupta, Amdocs, Contributor
- Sophie Bouleau, Orange, Contributor
- Ludovic Robert, Orange, Contributor
- Alexis de Peufeilhoux, Deutsche Telekom, Contributor
- Thomas Braun, Deutsche Telekom, Contributor
- Johanne Mayer, MayerConsult Inc, Contributor