Frameworx Specification

REST API Design Guidelines Part 1

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

TMF630

Release 14.5.0

November 2014

Latest Update: Frameworx Release 14.5	**Member Evaluation**
Version 1.1.0	**IPR Mode: RAND**

Notice

This document and the information contained herein is provided on an "AS IS" basis and TM FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Direct inquiries to the TM Forum office:

240 Headquarters Plaza,
East Tower – 10th Floor,
Morristown, NJ 07960 USA
Tel No. +1 973 944 5100
Fax No. +1 973 944 5110
TM Forum Web Page: www.tmforum.org

Table of Contents

List of Figures

N/A

List of Tables

N/A

Executive Summary

This document, “REST API Design Guidelines” provides information for the development of TM Forum APIs using REST. It provides recommendations and guidelines for the 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.

1. GENERAL

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:-

1. Resource Collection – server managed *collection* of resources. In the TMF REST API DP collections are anonymous. A resource with no identifier represents the resource collection (matching the resource type).

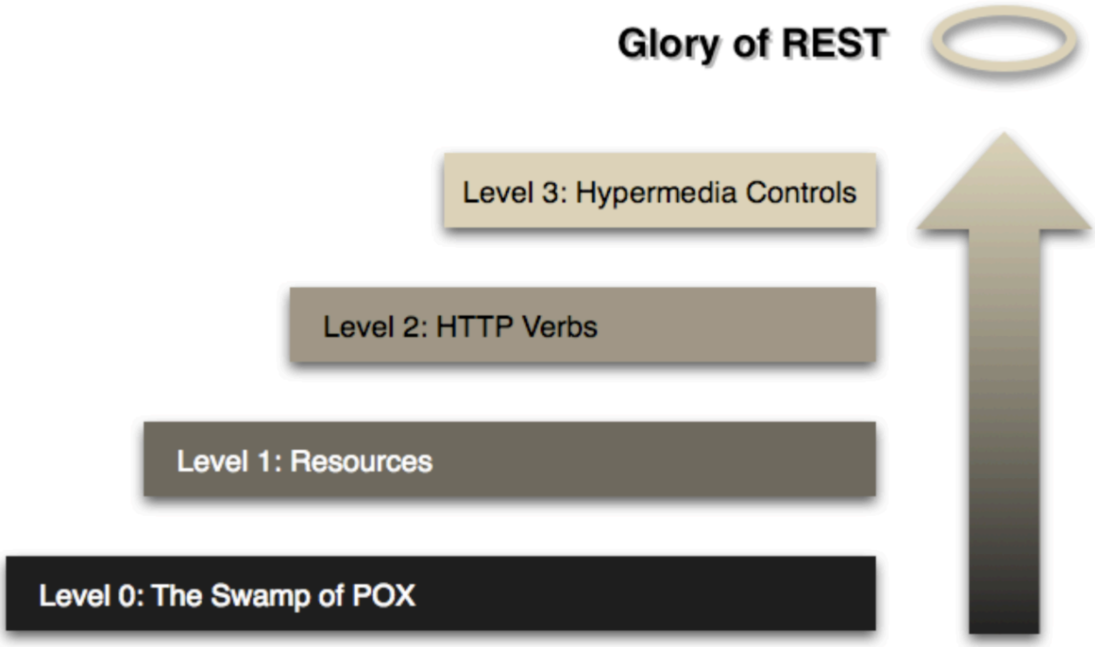
2. Managed Resource – e.g. a database record or a managed entity–. Its representation includes: fields with values and links to related resources. Can have child resources, of different resource types. Client can create, query, update and delete resources.

3. Tasks – 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.

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 Advanced Patterns sections as they relate to workflow oriented patterns.



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 and XML
- Response schema (*and where PUT, POST, PATCH are supported – request schema*).
- Links supported (Optional in L2 APIs)

- Response status codes supported.
- WADL

API implementation technology

APIs are not technology implementation dependent.

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.

HTTP HEADER

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

- Content-Type must be used when the message includes body
- Content-Length should be used. 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.
- Last-Modified should be used in responses
- ETag should be used in responses. The entity tag may be any string value, so long as it changes along with the resource's representation.
- If a resource is designed to be manipulated by more than one different client (i.e. application) then it must support conditional PUT requests.
- Location must be used 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.
- Cache-Control, Expires, and Date response headers should be used to encourage caching.
- Cache-Control and Expires response headers may be used to discourage caching.
- Caching should be encouraged.
- Expiration caching headers should be used with 200 ("OK") responses to GET and HEAD requests.

- Expiration caching headers may optionally be used with 3xx and 4xx responses - this helps reduce the amount of redirecting and error-triggering load on a REST API.

2. DOMAIN AND URI NAMING STANDARDS

Resources represents 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 know 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.

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.

Resource Model and Naming

A resource identifier is like a unique ID assigned to one or more service resources (also know 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 also be represented by an ID which contains the sequence of RDN without having the RDNs to be necessarily encoded as individual resources with an 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 Report Management application:

{reportPath} = report/reportID

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

{apiRoot} /{resourcePath}

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 full URI of the resource as per Location header “href=http://api/report/42”

Resource Naming Convention

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

e.g.*/account/billSummary/billDetail/account/billSummary/billdetail*

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)

Resource collection naming

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

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

3. Uniform Contract Methods and Media Types

HTTP provides us with a set of generic methods, such as GET, PUT, 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.

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.

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

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 must be used to completely update a resource identified by its resource URI
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.

- HEAD must be used to retrieve response headers. HEAD support is always optional.

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 JSON media type
- 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 are not supported.

.

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
200	OK - must **not** be used to communicate errors in the response body
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

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
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

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 signalled 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.

Common Information Model

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

4. Query Resources Patterns

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

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

```
{
   “id”: « 42 »,
  “href”: “/api/report/42”,

   "datetime": "2012-12-20 15:00:00",
   "health_state": "operational",
   "metrics": [
      {
         "code": "123",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
      },
      {
         "code": "321",
         "category_id": "48",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
      }
   ],
   "failures": [
      {
         "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
         "detail": "Network authentication failure",
         "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
      },
      {
         "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
         "detail": "Network authentication failure",
         "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
      }
   ]
     }
```

Query single Resource all attributes

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

If the resource exist 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.

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 Report with an ID of 42:

REQUEST

```
GET /api/report/42
```

RESPONSE

```
200

Content-Type: application/json

{
   "id": « 42 »,
  "href": "/api/report/42",
  "datetime": "2012-12-20 15:00:00",
  "health_state": "operational",
  "metrics": [
    {
      "code": "123",
      "category_id": "45",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
    },
    {
      "code": "321",
      "category_id": "48",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
    }
  ],
  "failures": [
    {
      "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
      "detail": "Network authentication failure",
      "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
    },
    {
      "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
      "detail": "Network authentication failure",
      "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
    }
  ]
}
```

Querying multiple Resources

The following section describe 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.

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.

The Content-Range header is used to indicate the presence of more elements in the collection and the current position of the elements in the overall collection.

Example :

Retrieving all reports.

REQUEST

```
GET /api/report
```

RESPONSE

```
200

Content-Type: application/json
Content-Range: items 1-2/2

[{
   “id”: « 42 »,
   “href”: “/api/report/42”,

   "datetime": "2012-12-20 15:00:00",
   "health_state": "operational",
   "metrics": [
      {
         "code": "123",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
      },
      {
         "code": "321",
         "category_id": "48",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
      }
   ],
   "failures": [
      {
         "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
         "detail": "Network authentication failure",
         "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
      },
      {
         "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
         "detail": "Network authentication failure",
         "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
      }
   ]
},
{
   “id”: « 52 « ,
   “href”: “/api/report/52”,

   "datetime": "2012-12-20 15:00:00",
   "health_state": "operational",
   "metrics": [
      {
         "code": "456",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
      "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
    },
    {
      "code": "321",
      "category_id": "48",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
    }
  ],
  "failures": [
    {
      "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
      "detail": "Network authentication failure",
      "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
    },
    {
      "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
      "detail": "Network authentication failure",
      "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
    }
  ]
}
]
```

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.

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.

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.

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 5 and populate the representations with the failures and metrics attributes. Note that the « id » attribute is always present.

REQUEST

```
GET /api/report/5/?fields=failures,metrics
```

RESPONSE

```
200

Content-Type: application/json

{
   “id”: 5,
   “href”: “/api/report/5”,
   "metrics": [
      {
         "code": "123",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
      },
      {
         "code": "321",
         "category_id": "48",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
      }
   ],
   "failures": [
      {
         "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
         "detail": "Network authentication failure",
         "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
      },
      {
         "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
         "detail": "Network authentication failure",
         "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
      }
   ]
}
```

Query Resources with attribute filtering

The following section describe 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}&*]

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};*]

For example :=

- GET /report?state=active&state=suspended
- GET /report?state=active;state=suspended
- GET /report?state=active,suspended

More complex filtering expressions involving non-equality operators are also supported and described in the "Advanced Patterns" document.

The following operators may be used :

.exact =	Description to be provided
.gt >	
.gte >=	
.lt <	
.lte <= .regex *=	

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

[{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.

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.

The Content-Range header is used to indicate the presence of more elements in the collection and the current position of the elements in the overall collection.

Example:

Retrieve all Reports with date_time greater than 2013-04-20 and health_state operational.

REQUEST

```
GET /api/report?date_time.gt=2013-04-20&health_state =operational        or
GET /api/report?date_time>2013-04-20&health_state = operational
```

RESPONSE

```
200

Content-Type: application/json
Content-Range: items 1-2/2

[ {
   “id”: « 42 »,
  “href”: “/api/report/42”,

   "datetime": "2012-12-20 15:00:00",
   "health_state": "operational",
      "metrics": [
        {
          "code": "123",
          "category_id": "45",
          "date_time": "2012-12-20 14:30:00",
          "reference": "4321",
          "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
          "value": 50,
          "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
        },
        {
          "code": "321",
          "category_id": "48",
          "date_time": "2012-12-20 14:30:00",
          "reference": "4321",
          "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
          "value": 50,
          "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
        }
      ],
      "failures": [
        {
          "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
          "detail": "Network authentication failure",
          "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
        },
        {
          "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
          "detail": "Network authentication failure",
          "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
        }
      ]
    },
    {
      “id”: « 52 « ,
     “href”: “/api/report/52”,

      "datetime": "2012-12-20 15:00:00",
      "health_state": "operational",
      "metrics": [
        {
          "code": "456",
          "category_id": "45",
          "date_time": "2012-12-20 14:30:00",
          "reference": "4321",
          "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
          "value": 50,
          "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
        },
        {
          "code": "321",
          "category_id": "48",
          "date_time": "2012-12-20 14:30:00",
          "reference": "4321",
          "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
          "value": 50,
          "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
        }
      ],
      "failures": [
        {
          "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
          "detail": "Network authentication failure",
          "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
        },
        {
          "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
          "detail": "Network authentication failure",
          "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
        }
      ]
    }
    ]
```

Query Resources with attribute filtering and Iterators

Some operations may return a very large amount of data. The content Range header MUST 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 reports. The example assume that 50 reports match a filtering criteria and that the maximum amount of reports being retrieved by calls is set by default to 10.

Example:

REQUEST

```
GET /api/report?date_time.gt=2013-04-20&state.execution=suspended
```

RESPONSE

```
200

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

[ {
   "id": « 42 »,
…
},
{
   "id": « 43 « ,
 …
}
…
{
   "id": « 62 « ,
 …
}

]
```

Retrieving the next elements:

REQUEST

```
GET /api/report?date_time.gt=2013-04-20&state.execution=suspended
    Range:: items=11-20
```

RESPONSE

```
200

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

[{
   “id”: « 72 »,
…
},
{
   “id”: « 73 « ,
 …
}
…
{
   “id”: « 82 « ,
 …
}

]
```

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/report/fields=failures,health_state&report.date_time.gt=2013-04-
20&report.state.execution=suspended
```

5. Modify resources patterns

The following section describe the patterns used to modify resources.

Uniform contract operations for modifying resources

TMF REST API’s MUST only uses 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.

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.

Example:

Change the health_state of the report with ID= 42 to non operational.

REQUEST

```
PUT /api/report/42
{
   “id”:  “42”,
   "datetime": "2012-12-20 15:00:00",
  "health_state": "nonop",
   "metrics": [
    {
      "code": "123",
      "category_id": "45",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
    },
    {
      "code": "321",
      "category_id": "48",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
    }
}
```

RESPONSE

```
200

Content-Type: application/json

{
  “id”:  “42”,
   “href”:=”/api/report/42”,

  "datetime": "2012-12-20 15:00:00",
 "health_state": "nonop",
  "metrics": [
    {
      "code": "123",
      "category_id": "45",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
    },
    {
      "code": "321",
      "category_id": "48",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
    }
}
```

Note that using PUT with only the "health_state": " nonop " attribute subset will result in all other attributes being set to null or to empty.

REQUEST

```
PUT /api/report/42
{

"health_state": " nonop "
}
```

RESPONSE

```
200

Content-Type: application/json

{
   “id”: 42
   "datetime": null,
   "health_state": "nonop",
   "metrics": [],
   "failures": []
}
```

Modify Attribute subset of a resource

PATCH MUST be used to update a limited subset of the attributes of an entity as described in http://tools.ietf.org/html/rfc5789

Example:

Change the health_state of the report with ID= 42 to non operational.

REQUEST

```
PATCH /api/report/5
Content-Type: Application/json

{
   “health_state”: “nonop”
}
```

RESPONSE

```
200

Content-Type: Application/json

{
   “id”: 5,
    “href”:=”/api/report/5”,
   "datetime": "2012-12-20 15:00:00",
   "health_state": "nonop",
   "metrics": [
      {
         "code": "123",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
      },
      {
         "code": "321",
         "category_id": "48",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
    }
  ],
  "failures": [
    {
      "failure_id": "0bec4420-4b01-11e2-bcfd-0800200c9a66",
      "detail": "Network authentication failure",
      "source_id": "24b2bc00-4b01-11e2-bcfd-0800200c9a66"
    },
    {
      "failure_id": "3b687c50-4b01-11e2-bcfd-0800200c9a66",
      "detail": "Network authentication failure",
      "source_id": "462f1f40-4b01-11e2-bcfd-0800200c9a66"
    }
  ]
}
```

Modify Multi-Valued Attribute

The modification of list based attributes can be performed using :

- PATCH per http://tools.ietf.org/html/rfc5789
- JSON PATCH as per http://tools.ietf.org/html/rfc5789

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]

The response message MUST contain the representation of the modified resource.

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": "http://serverlocation:port/partyManagement/organization/1",
           "validFor": {
              "startDateTime": "2013-04-19T16:42:23-04:00",
              "endDateTime": ""
           },
           "status": "Active"
         }
    }

}
```

RESPONSE

```
201
Content-Type: application/json

{ JSON Resource Representation with ALL Attributes
}
```

6. Create Resource Patterns

The following section describe the patterns used to create resources. Batch resource creation is addressed in the Advanced Patterns document.

Creating a single Resource

POST must be used to create single resources.

Return message SHOULD contain a resource representation and a Location header with the created resource’s URI.

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 code.

Example:

Create a Report

REQUEST

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



{
   “id”: 5,
   "datetime": "2012-12-20 15:00:00",
   "health_state": "nonop",
   "metrics": [
      {
         "code": "123",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
      },
      {
         "code": "321",
         "category_id": "48",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
      }
}
```

RESPONSE

```
200

Location: /api/report/5


Content-Type: application/json

{
    “id”: 5,
    "datetime": "2012-12-20 15:00:00",
    "health_state": "nonop",
    "metrics": [
        {
            "code": "123",
            "category_id": "45",
            "date_time": "2012-12-20 14:30:00",
            "reference": "4321",
            "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
            "value": 50,
            "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
        },
        {
            "code": "321",
            "category_id": "48",
            "date_time": "2012-12-20 14:30:00",
            "reference": "4321",
            "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
            "value": 50,
            "metric_id": "c50f3e90-4b00-11e2-bcfd-0800200c9a66"
        }
}
```

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.

Creating Multiple Resources

JSON PATCH 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 use to scope a specific collection. Entities are added to the targeted collection.

Return message SHOULD contain the resource representation of all the resource created.

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 PATCH must return a 200 code.

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 a multiple Reports

REQUEST

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

  {
    {
     "op": "add",
     "path": "/",
     "value":
       {
 “id”: 5,
  "datetime": "2012-12-20 15:00:00",
  "health_state": "nonop",
  "metrics": [
    {
      "code": "123",
      "category_id": "45",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
    }
      } ,

      {
     "op": "add",
     "path": "/",
     "value":
       {
  “id”: 6,
  "datetime": "2012-12-20 15:00:00",
  "health_state": "nonop",
  "metrics": [
    {
      "code": "123",
      "category_id": "45",
      "date_time": "2012-12-20 14:30:00",
      "reference": "4321",
      "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
      "value": 50,
      "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
    },
            }
      }
```

RESPONSE

```
201


Content-Type: application/json

{
   “id”: 5,
   "datetime": "2012-12-20 15:00:00",
   "health_state": "nonop",
   "metrics": [
      {
         "code": "123",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
      },
{
   “id”: 6,
   "datetime": "2012-12-20 15:00:00",
   "health_state": "nonop",
   "metrics": [
      {
         "code": "123",
         "category_id": "45",
         "date_time": "2012-12-20 14:30:00",
         "reference": "4321",
         "source_id": "8f27ce50-4b00-11e2-bcfd-0800200c9a66",
         "value": 50,
         "metric_id": "b0dbbc50-4b00-11e2-bcfd-0800200c9a66"
      }



}
```

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.

7. Task Resource Pattern

This section describe the use of Task resources to expose complex operations not easily or not decomposable to CRUD Entity based operations.

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 is a verb representing the task to be executed. TASK creation may result in the creation of a number of related resources or other tasks.

Example:

Given is the step-by-step process of establishing a call via MTNM, but this can be abbreviated as one POST to a call TASK with all of the related information for connections included.

Create a call

REQUEST

```
POST /call

{ // <callSNC::CallCreateData_T>
   "name": {}, // <globaldefs::NamingAttributes_T> callName
   "user_label": "...", // <string> userLabel
   "owner": "...", // <string> owner
   "network_access_domain": "...", // <string> networkAccessDomain
   "aEnd": { // <CallEndT> aEnd
      "tna": "...", // <mLSNPP::TNAName_T> tNANameOrGroupTNAName
      "snpp": "...", // <string> sNPPid
      "snp": "...", // <string> sNPid
      "tp": {} // <globaldefs::NamingAttributes_T> tpName
   },
   "zEnd": {}, // <CallEndT> zEnd
```

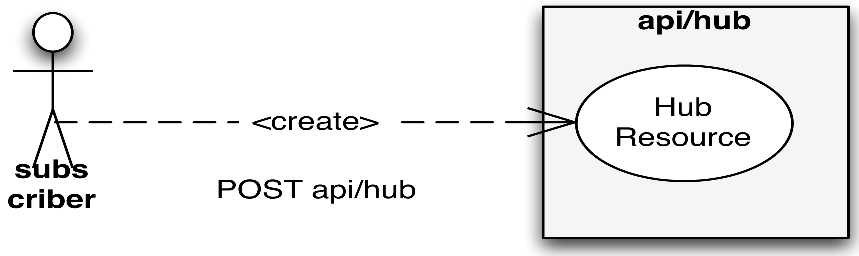
```
    "parameters": { // <CallParameterProfile_T> callParameters
        "severely_degraded_threshold": "...", // <string> severelyDegradedThreshold
        "degraded_threshold": "...", // <string> degradedTreshold
        "class_of_service": "...", // <string> classOfService
        "class_of_service_parameters": {}, // <globaldefs::NVSList_T>
classOfServiceParameters
    },
    "diversity": { // <Diversity_T> callDiversity
        "corouting_level_of_effort": "...", // <LevelOfEffort_T> coroutingLevelOfEffort
        "node_diversity_level_of_effort": "...", // <LevelOfEffort_T>
nodeDiversityLevelOfEffort
        "link_diversity_level_of_effort": "...", // <LevelOfEffort_T>
linkDiversityLevelOfEffort
        "node_srg_type": "...", // <string> nodeSRGType
       "link_srg_type": "..." // <string> linkSRGType
    },
    "additional": {} // <globaldefs::NVSList_T> additionalCreationInfo
}
```

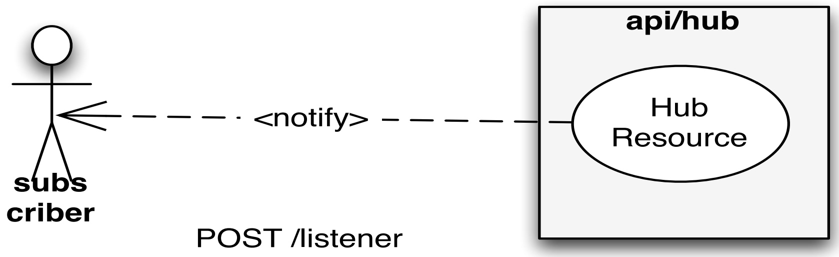
RESPONSE

```
201

Location: /call/e21cc459-0f4d-49c3-bc12-36c418c71c5f

{ // <CallT> with additional properties from <callSNC::CallCreateData_T> when
available
    "id": "e21cc459-0f4d-49c3-bc12-36c418c71c5f", // <string> callId
    "state": "...", // <CallState_T> callState
    "native_name": "...", // <string> nativeEMSName
    "diversity_synthesis": "...", // <string> diversitySynthesis
    "user_label": "...", // <string> userLabel
    "owner": "...", // <string> owner
    "network_access_domain": "...", // <string> networkAccessDomain
}
```

8. Notification Patterns

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





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 if the request was successful.

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

Example: Create a Hub to receive events on the "http://in.listener.com"

REQUEST
POST /api/hub Accept: application/json {"callback": "http://in.listener.com"}
RESPONSE
201 Content-Type: application/json Location: /api/hub/42 {"id":"42","callback":"http://in.listener.com","query":null}

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.

Returns HTTP/1.1 status code 204 if the request was successful.

Returns HTTP/1.1 status code 404 if the resource is not found.

REQUEST

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

RESPONSE

```
204
```

Publishing Events

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

The structure of the event is:

```
{
   "event": {
   EVENT BODY
         },
   "eventType": "eventType"
}
```

Returns HTTP/1.1 status code 201

For example posting a ManagementReport event:

REQUEST

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

{
   "event": {
      "dateTime": null,
      "id": "42",
      "state": {
         "healthState": "UNKNOWN",
         "executionState": "ACTIVE"
      },
      "metrics": {"item": [
         {
            "code": "76e27e2b-644e-11e2-8ea1-5c260a86d1e4",
            "categoryID": null,
            "dateTime": 1367266835571,
            "reference": null,
            "sourceID": null,
            "value": "20",
            "metricID": null
         },

         {
            "code": "7987c7bc-644e-11e2-8ea1-5c260a86d1e4",
            "categoryID": null,
            "dateTime": 1367266835573,
            "reference": null,
            "sourceID": null,
            "value": "60000",
            "metricID": null
         },
         {
            "code": "7c7d6e22-644e-11e2-8ea1-5c260a86d1e4",
            "categoryID": null,
            "dateTime": 1367266835573,
            "reference": null,
            "sourceID": null,
            "value": "12",
            "metricID": null
         },
         {
            "code": "7c98ea37-644e-11e2-8ea1-5c260a86d1e4",
            "categoryID": null,
            "dateTime": 1367266835573,
            "reference": null,
            "sourceID": null,
            "value": "70000",
            "metricID": null
         },
         {
            "code": "7dd589b2-644e-11e2-8ea1-5c260a86d1e4",
            "categoryID": null,
            "dateTime": 1367266835573,
            "reference": null,
            "sourceID": null,
            "value": "89",
            "metricID": null
         }
      ]},
      "failures": null
```

```
        },
        "eventType": "ManagementReport"
    }
```

9. Versioning

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.

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.

10. Appendix A: Terms and Abbreviations Used within this Document

Terminology

Term	Definition	TMF or Outside Source
<<BA Specific Term 1>>	<<Definition 1>>	<<Source>>
<<BA Specific Term n>>	<<Definition n>>	<<Source>>

Abbreviations and Acronyms

Abbreviation/ Acronym	Abbreviation/ Acronym Spelled Out	Definition	TMF or External Source
<<Abbreviation/ Acronym 1>>	<<Expansion of abbreviation/acronym 1>>	<<Definition 1>>	<<Source>>
<<Abbreviation/ Acronym n>>	<<Expansion of abbreviation/acronym n>>	<<Definition n>>	<<Source>>

11. References

References

Reference	Description	Source	Brief Use Summary
Project Charter	<<PROJECT name>> Project Charter		<<summary>>
Link To Models	<< Hyperlinks to or location of associated models>>		<<summary>>
<<Reference 1>>	<<Type, Title, Number, Revision, Date>>		<<summary>>
<<Reference n>>	<< Type, Title, Number, Revision, Date>>		<<summary>>

12. 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.

Document History

12.0.1. 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	23/06/2013	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 October 2013	Tina O'Sullivan	Minor corrections
1.1.0	November 2014	Pierre Gauthier	Updates for Fx14.5

12.0.2. 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 Number	Date Modified	Modified by:	Description of changes
1.0	DD/MMM/YY	<<name>>	first release of document

Company Contact Details

Company	Team Member Representative
Include all involved companies adding lines as necessary.	*Name* *Title* *Email* *Phone* *Fax*
	Name *Title* *Email* *Phone* *Fax*

Acknowledgments

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

- Pierre Gauthier, TM Forum, **Editor** and Team Leader
- John Morey Ciena
- Maxime Delon Orange

Additional input was provided by the following people:

- John Wilmes, TM Forum