**TM Forum Specification**

**Service Problem Management API REST Specification**

**TMF656**

**Release 18.5.0**

**January 2019**

|  |  |
| --- | --- |
| **Latest Update: TM Forum Release 18.5.0** | **Member Evaluation** |
| **Version 4.0.0** | **IPR Mode: RAND** |

# NOTICE

Copyright © TM Forum 2019. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to TM FORUM, except as needed for the purpose of developing any document or deliverable produced by a TM FORUM Collaboration Project Team (in which case the rules applicable to copyrights, as set forth in the [TM FORUM IPR Policy](http://www.tmforum.org/IPRPolicy/11525/home.html), must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by TM FORUM or its successors or assigns.

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:

4 Century Drive, Suite 100

Parsippany, NJ 07054, USA

Tel No. +1 973 944 5100

Fax No. +1 973 998 7196

TM Forum Web Page: [www.tmforum.org](http://www.tmforum.org/)

# Table of Contents

[NOTICE 2](#_Toc514836422)

[Table of Contents 2](#_Toc514836423)

[List of Tables 2](#_Toc514836424)

[Introduction 2](#_Toc514836425)

[SAMPLE USE CASES 2](#_Toc514836426)

[Support of polymorphism and extension patterns 2](#_Toc514836427)

[RESOURCE MODEL 2](#_Toc514836428)

[Managed Entity and Task Resource Models 2](#_Toc514836429)

[FIRST resource 2](#_Toc514836430)

[Notification Resource Models 2](#_Toc514836431)

[First Notification 2](#_Toc514836432)

[API OPERATIONS 2](#_Toc514836433)

[VERB url 2](#_Toc514836434)

[API NOTIFICATIONS 2](#_Toc514836435)

[Register listener 2](#_Toc514836436)

[Unregister listener 2](#_Toc514836437)

[Publish Event to listener 2](#_Toc514836438)

[Acknowledgements 2](#_Toc514836439)

[Release History 2](#_Toc514836440)

[Contributors to Document 2](#_Toc514836441)

# List of Tables

N/A

# Introduction

This Service Problem Management API is used for the service providers (Defined as the Middle B) to manage the service problems in their service area. Service problem is generated based on the information declared by Middle B or the event information notified from infrastructure providers (Defined as the First B) who provide the infrastructure of cloud or network. The event information includes alarm information, performance anomaly information, trouble ticket information, SLA violation, maintenance information and prediction information. Middle Bs can refer the service problems and the event information from First Bs and when the service problems occur or its status have been changed, Middle Bs can receive notifications. According to these functions, Middle Bs are able to grasp the service problems quickly and accurately.

# SAMPLE USE CASES

We assume following situation:  
There are Network Provider 1 (NP1) and 2 (NP2), which provide network infrastructure, and Cloud Provider 1, 2, which provides cloud infrastructure, as First Bs. Using these infrastructure, Service Provider 1 (SP1), 2 (SP2) and 3 (SP3) are providing their services to their end-users as Middle Bs.



## Use case 1

When trouble happened in any resources of NW/Cloud Providers, Service Providers can know their services are affected or not. The specific use case is following:

1. SPM collects configuration information of services provided by service providers in advance using Product Inventory Management API, etc.
2. Each of Middle Bs – Service Provider 1 (SP1), Service Provider 2 (SP2), Service Provider 3 (SP3) – registers the notification destination to SPM SPI.
3. When a fault occurs, SPM receives a trouble ticket from the Network Provider 1 (NP1).
4. SPM creates a Service Problem based on the Trouble Ticket.
5. SPM notifies the Service Problem creation notification to Middle B (SP1, SP3) to notify expected service impact, based on the configuration information collected in advance.
6. When SPM receives a notification that the trouble ticket has changed to "In Progress" state, update the status of the relevant Service Problem. Notify the Service Problem state change notification to Middle B (SP1, SP3).



## Use case 2

To analyze the past problems, Middle B collects the problem information in the past one year.

## Use case 3

Service providers can declare a new service problem based on trouble declarations from their end-users. In addition, the SPM administrator can associate the service problem, based on the Middle B declaration, with another problem based on a First B event such as a Trouble Ticket. The specific use case is following:

1. Based on the report from the user that there is a problem in the Internet access, Middle B (SP1) gets the current service problem.
2. After SPM collects the current Service Problems, returns that there are no problems related to the service of the Middle B (SP1).
3. In order to request the analysis of this event, Middle B declares a new service problem.
4. Since the SPM administrator found that necessary detailed information was insufficient, SPM administrator requests additional information about the behavior of the Middle B side.
5. SP1 collects the specified additional information and registers it.
6. SPM administrator checks the additional information, and accepts the Service Problem (Problem 1).
7. First B(NP1) registers a detected problem event to the trouble ticket, and notice a new trouble ticket generation to SPM. The SPM creates a Service Problem (Problem 2) based on the trouble ticket.
8. Since the two problems affects the same location, SPM administrator determines that the declared problem (Problem 1) and the new problem based on the new trouble ticket (Problem 2) is associated. SPM administrator associates Problem 1 with 2. In this case, Problem 1 will have Problem2 as the association “underlyingProblem”. Note that Problem 1 can have an “parentProblem” as another association if he would like to group those problems.
9. Since the Problem 1 was changed to add a “underlyingProblem”, Service Problem Change Notification is sent to SP1.



## Use case 4

The SPM administrator can associate and group multiple service problems so that service providers can easily recognize what the real problem is. The specific use case is following:

1. SPM receives an alarm from NP1, and creates a Service Problem based on it (Problem 1).
2. SPM recieves an SLA violation from NP2 and creates a Service Problem based on it (Problem 2).
3. By analyzing problems, SPM administrator determines that Problem 1 and 2 are the same problem. SPM administrator creates a new Service Problem (Problem 3) in order to group and associate Problem 1, 2 and 3. In this case, Problem 3 is a parent and Problem 1 and 2 are children.



## Service Problem Lifecycle

ServiceProblem states:

Following the available status values for a service problem are listed. The status value is in accordance with Trouble Ticket API. The state graphic gives an overview of the allowed status changes

* Submitted
* Rejected
* Acknowledged
* In Progress
  + Held
  + Pending
* Resolved
* Closed
* Cancelled

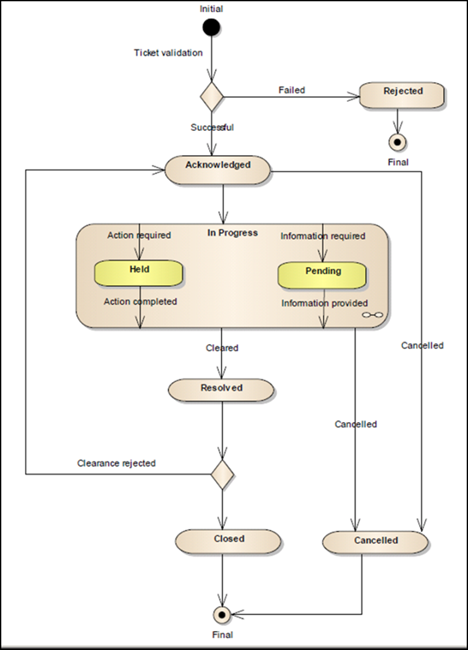


Figure 1 – Life Cycle

|  |  |
| --- | --- |
| State | Description |
| Submitted | The initial state of a service problem when created by a service problem originator |
| Acknowledged | The Service Problem was accepted and allocated a unique service problem id by Service problem handler. |
| In Progress | The service problem was validated by the service problem handler and is being processed. |
| Resolved | The fault indicated in the service problem was corrected by the service problem handler and acknowledgement is awaited from its originator. |
| Closed | The service Problem’s originator has acknowledged the ‘Resolved’ state of the service problem, or the timeframe for acknowledgement has passed without response from service problem originator. |
| Rejected | The service problem was rejected because it:   * is not submitted * provides invalid information * fails to meet the Business rules in respect of the product which originator is raising a service problem against * is otherwise defective |
| In Progress – Pending | Service problem handler is awaiting further confirmation on details of a Fault from originator before it can progress the Fault. An example is where appointment information is required. |
| In Progress - Held | Service problem handler is confirming further details internally before completing a service problem. An example is where service problem handler for network infrastructure spare parts to progress with the fault rectification. |
| Cancelled | The service problem was cancelled because it:   * Was cancelled by service problem originator |

# Support of polymorphism and extension patterns

Support of polymorphic collections and types and schema based extension is provided by means of a list of generic meta-attributes that we describe below. Polymorphism in collections occurs when entities inherit from base entities, for instance a RouterProblem and MainSwitchProblem inheriting properties from the ServiceProblem entity.

Generic support of polymorphism and pattern extensions is described in the TMF API Guidelines v3.0 Part 2 document.

The @type attribute provides a way to represent the actual class type of an entity. For example, within a list of ServiceProblem instances some may be instances of RouterProblem where other could be instances of MainSwitchProblem. 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.

The @referredType can be used within reference entities (like for instance an ServiceProblemRef object) to explicitly denote the actual entity type of the referred class. Notice that in reference entities the @type, when used, denotes the class type of the reference itself, such as ServiceProblemRef, and not the class type of the referred object. However, since reference classes are rarely sub-classed, @type is generally not useful in reference objects.

The @schemaLocation property can be used in resources to allow specifying user-defined properties of an Entity or to specify the expected *characteristics* of an entity.

The @baseType attribute gives a way to provide explicitly the base of class of a given resource that has been extended.

# RESOURCE MODEL

## Managed Entity and Task Resource Models

### FIRST resource

## Notification Resource Models

### First Notification

# API OPERATIONS

Remember the following Uniform Contract:

|  |  |  |
| --- | --- | --- |
| 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. |

Filtering and attribute selection rules are described in the TMF REST Design Guidelines Part 1 Document.

Notifications are also described in a subsequent section.

## VERB url

# API NOTIFICATIONS

For every single of operation on the entities use the following templates and provide sample REST notification POST calls.

It is assumed that the Pub/Sub uses the Register and UnRegister mechanisms described in the REST Guidelines part 1. Refer to the guidelines for more details.

## Register listener

**POST /hub**

**Description**

Sets the communication endpoint address the service instance must use to deliver information about its health state, execution state, failures and metrics. Subsequent POST calls will be rejected by the service if it does not support multiple listeners. In this case DELETE /api/hub/{id} must be called before an endpoint can be created again.

**Behavior**

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

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

**Usage Samples**

Here's an example of a request for registering a listener.

|  |
| --- |
| **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

**DELETE /hub/{id}**

**Description**

Clears the communication endpoint address that was set by creating the Hub.

**Behavior**

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.

**Usage Samples**

Here's an example of a request for un-registering a listener.

|  |
| --- |
| **Request** |
| DELETE /api/hub/42  Accept: application/json |
| **Response** |
| 204 |

## Publish Event to listener

**POST /client/listener**

**Description**

Clears the communication endpoint address that was set by creating the Hub.

Provides to a registered listener the description of the event that was raised. The /client/listener url is the callback url passed when registering the listener.

**Behavior**

Returns HTTP/1.1 status code 201 if the service is able to set the configuration.

**Usage Samples**

Here's an example of a notification received by the listener. In this example “EVENT TYPE” should be replaced by one of the notification types supported by this API (see Notification resources Models section) and EVENT BODY refers to the data structure of the given notification type.

|  |
| --- |
| **Request** |
| POST /client/listener  Accept: application/json  {  "event": {  EVENT BODY  },  "eventType": "EVENT\_TYPE"  } |
| **Response** |
| 201 |

For detailed examples on the general TM Forum notification mechanism, see the TMF REST Design Guidelines Part 1 document.

# Acknowledgements

## Document History

## Release History

|  |  |  |  |
| --- | --- | --- | --- |
| **Release Number** | **Date** | **Release led by:** | **Description** |
| Release 1.0 | 20-Jun-2018 | Pierre Gauthier  TM Forum  [pgauthier@tmforum.org](mailto:pgauthier@tmforum.org) | First Release of the Document. |
| Release 4.0.0 | 16-Jan-2019 | Jonathan Goldberg  Amdocs  [Jonathan.Goldberg@amdocs.com](mailto:Jonathan.Goldberg@amdocs.com) | Schema alignment for NaaS APIs |

### Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Release Number** | **Date** | **Release led by:** | **Description** |
| Release 18.5.0 | 16 Jan 2019 | Jonathan Goldberg  Amdocs  [Jonathan.Goldberg@amdocs.com](mailto:Jonathan.Goldberg@amdocs.com) | Schema alignment for NaaS APIs |

## Contributors to Document

|  |  |
| --- | --- |
| Pierre Gauthier | TM Forum |
| Kiyotaka Mizuno | NTT |
| Takayuki Nakamura | NTT |
| Jonathan Goldberg | Amdocs |