MEF Standard

MEF 124

LSO Cantata and LSO Sonata Trouble Ticket and Incident Management API - Developer Guide

February 2023

Disclaimer

The information in this publication is freely available for reproduction and use by any recipient and is believed to be accurate as of its publication date. Such information is subject to change without notice and MEF Forum (MEF) is not responsible for any errors. MEF does not assume responsibility to update or correct any information in this publication. No representation or warranty, expressed or implied, is made by MEF concerning the completeness, accuracy, or applicability of any information contained herein and no liability of any kind shall be assumed by MEF as a result of reliance upon such information.

The information contained herein is intended to be used without modification by the recipient or user of this document. MEF is not responsible or liable for any modifications to this document made by any other party.

The receipt or any use of this document or its contents does not in any way create, by implication or otherwise:

a) any express or implied license or right to or under any patent, copyright, trademark or trade secret rights held or claimed by any MEF member which are or may be associated with the ideas, techniques, concepts or expressions contained herein; nor
b) any warranty or representation that any MEF members will announce any product(s) and/or service(s) related thereto, or if such announcements are made, that such announced product(s) and/or service(s) embody any or all of the ideas, technologies, or concepts contained herein; nor
c) any form of relationship between any MEF member and the recipient or user of this document.

Implementation or use of specific MEF standards, specifications, or recommendations will be voluntary, and no Member shall be obliged to implement them by virtue of participation in MEF Forum. MEF is a non-profit international organization to enable the development and worldwide adoption of agile, assured and orchestrated network services. MEF does not, expressly or otherwise, endorse or promote any specific products or services.

Table of Contents

List of Contributing Members

The following members of the MEF participated in the development of this document and have requested to be included in this list.

Member
Amartus
Lumen Technologies
NEC/Netcracker
Proximus
Spirent Communications

Table 1. Contributing Members

1. Abstract

This standard is intended to assist implementation of the Trouble Ticketing functionality defined for the LSO Cantata and LSO Sonata Interface Reference Points (IRPs), for which requirements and use cases are defined in MEF 113 *Trouble Ticketing Requirements and Use Cases* [MEF113]. This standard consists of this document and complementary API definitions for Trouble Ticket Management and Trouble Ticket Notification.

This standard normatively incorporates the following files by reference as if they were part of this document, from the GitHub repository

https://github.com/MEF-GIT/MEF-LSO-Sonata-SDK

commit id: 729804b9e383db93e6fca923106473f2828e244f

- `productApi/troubleTicket/troubleTicketManagement.api.yaml`
- `productApi/troubleTicket/troubleTicketNotification.api.yaml`

https://github.com/MEF-GIT/MEF-LSO-Cantata-SDK

commit id: 5c9b397f9737a6d65931c6598bff1b58485e0a95

- `productApi/troubleTicket/troubleTicketManagement.api.yaml`
- `productApi/troubleTicket/troubleTicketNotification.api.yaml`

The Trouble Ticket API is defined using OpenAPI 3.0 [OAS-V3]

2. Terminology and Abbreviations

This section defines the terms used in this document. In many cases, the normative definitions of terms are found in other documents. In these cases, the third column is used to provide the reference that is controlling, in other MEF or external documents.

In addition, terms defined in the standards referenced below are included in this document by reference and are not repeated in the table below:

- MEF 55.1
- MEF 79
- MEF 80
- MEF 113

Term	Description	Reference
Application Program Interface	In the context of LSO, API describes one of the Management Interface Reference Points based on the requirements specified in an Interface Profile, along with a data model, the protocol that defines operations on the data and the encoding format used to encode data according to the data model. In this document, API is used synonymously with REST API	[MEF55.1]
Buyer	In the context of this document, denotes the organization or individual acting as the customer in a transaction over a Cantata (Customer <-> Service Provider) or Sonata (Service Provider <-> Partner) Interface	This document; adapted from [MEF80]
Incident	An entry within a Seller's tracking system created by the context of this document, denotes a situation that is not part of normal operationSeller, which contains information about a Situation in the Seller's network that has a possible negative impact on the operability of the network ona Product for one or more Buyers	[MEF113]
Issue	In the context of this document, denotes a problem with a Product as experienced by the Buyer that is not part of normal operation.	[MEF113]
Notification	A message sent from the Seller to the Buyer to inform about an event that has occurred in regard to a specific instance of a Ticket or an Incident	[MEF113]
Requesting Entity	The business organization that is acting on behalf of one or more Buyers. In the most common case, the Requesting Entity	[MEF79]

	represents only one Buyer and these terms are then synonymous	
Responding Entity	The business organization that is acting on behalf of one or more Sellers. In the most common case, the Responding Entity represents only one Seller and these terms are then synonymous	[MEF79]
REST API	Representational State Transfer. REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.	[REST]
Seller	In the context of this document, denotes the organization acting as the supplier in a transaction over a Cantata (Customer <-> Service Provider) or Sonata (Service Provider <-> Partner) Interface	This document; adapted from [MEF80]
Situation	In the context of this document, denotes a problem that is not part of normal operation in the Seller's network	[MEF113]
Ticket	An entry within a Seller's tracking system created by the Buyer (or a third party on behalf of the Buyer), which contains information about an Issue impacting normal operation of a Product, along with support interventions made by technical support staff, or third parties	[MEF113]
Trouble Ticketing	In the context of this document, denotes the management of both Tickets and Incidents	[MEF113]
Work Order	In the context of this document, denotes a set of tasks to be scheduled and performed under the responsibility of a Technician at a given location	[MEF113]

Table 2. Terminology

Term	Description	Reference
API	Application Program Interface	[MEF55.1]
REST API	Representational State Transfer API	[REST]

Table 3. Abbreviations

3. Compliance Levels

The key words **"MUST"**, **"MUST NOT"**, **"REQUIRED"**, **"SHALL"**, **"SHALL NOT"**, **"SHOULD"**, **"SHOULD NOT"**, **"RECOMMENDED"**, **"NOT RECOMMENDED"**, **"MAY"**, and **"OPTIONAL"** in this document are to be interpreted as described in BCP 14 (RFC 2119 [RFC2119], RFC 8174 [RFC8174]) when, and only when, they appear in all capitals, as shown here. All key words must be in bold text.

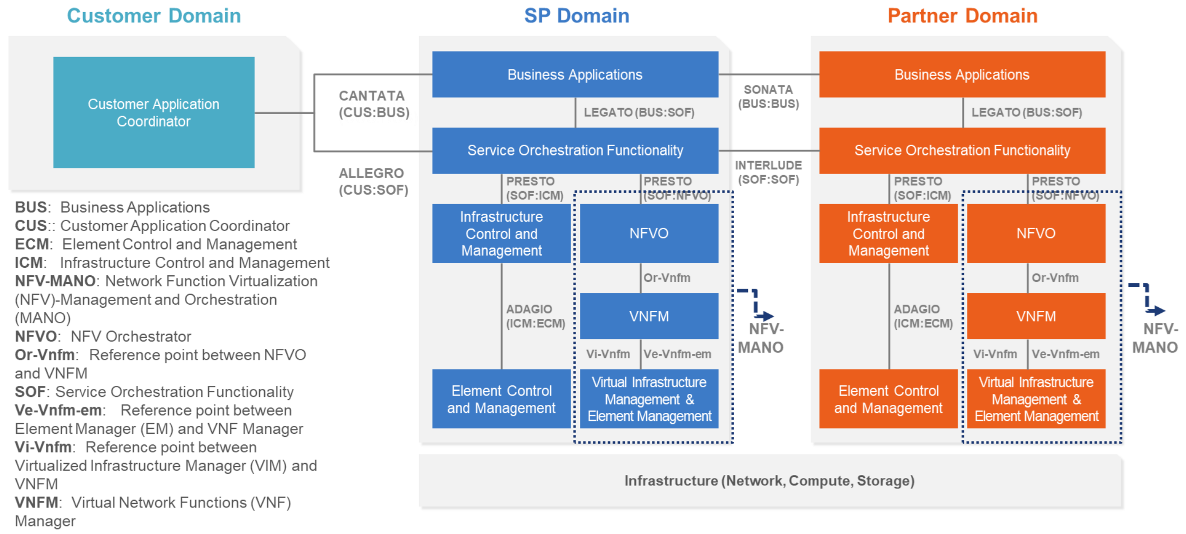
Items that are **REQUIRED** (contain the words **MUST** or **MUST NOT**) are labeled as **[Rx]** for required. Items that are **RECOMMENDED** (contain the words **SHOULD** or **SHOULD NOT**) are labeled as **[Dx]** for desirable. Items that are **OPTIONAL** (contain the words MAY or OPTIONAL) are labeled as **[Ox]** for optional.

A paragraph preceded by **[CRa]<** specifies a conditional mandatory requirement that **MUST** be followed if the condition(s) following the "<" have been met. For example, **" [CR1]<[D38]"** indicates that Conditional Mandatory Requirement 1 must be followed if Desirable Requirement 38 has been met. A paragraph preceded by **[CDb]<** specifies a Conditional Desirable Requirement that **SHOULD** be followed if the condition(s) following the "<" have been met. A paragraph preceded by **[COc]<**specifies a Conditional Optional Requirement that **MAY** be followed if the condition(s) following the "<" have been met.

4. Introduction

The Trouble Ticket API allows the Buyer to create, retrieve, and update Trouble Tickets as well as receive notifications and Trouble Tickets' updates. This allows managing issues and situations that are not part of normal operations of the Product provided by the Seller.

This standard specification document describes the Application Programming Interface (API) for Trouble Ticketing functionality of the LSO Cantata Interface Reference Point (IRP) and LSO Sonata IRP as defined in the *MEF 55.1 Lifecycle Service Orchestration (LSO): Reference Architecture and Framework* [MEF55.1]. The LSO Reference Architecture is shown in Figure 1 with both IRPs highlighted.



Customer Domain
SP Domain
Partner Domain
Customer Application Coordinator
CANTATA (CUS:BUS)
ALLEGRO (CUS:SOF)
Business Applications
LEGATO (BUS:SOF)
Service Orchestration Functionality
PRESTO (SOF:ICM)
PRESTO (SOF:NFVO)
Infrastructure Control and Management
NFVO
Or-Vnfm
VNFM
ADAGIO (ICM:ECM)
Vi-Vnfm
Ve-Vnfm-em
Element Control and Management
Virtual Infrastructure Management & Element Management
SONATA (BUS:BUS)
INTERLUDE (SOF:SOF)
NFV-MANO
Infrastructure (Network, Compute, Storage)
BUS: Business Applications
CUS:: Customer Application Coordinator
ECM: Element Control and Management
ICM: Infrastructure Control and Management
NFV-MANO: Network Function Virtualization (NFV)-Management and Orchestration (MANO)
NFVO: NFV Orchestrator
Or-Vnfm: Reference point between NFVO and VNFM
SOF: Service Orchestration Functionality
Ve-Vnfm-em: Reference point between Element Manager (EM) and VNF Manager
Vi-Vnfm: Reference point between Virtualized Infrastructure Manager (VIM) and VNFM
VNFM: Virtual Network Functions (VNF) Manager

Figure 1. The LSO Reference Architecture

Cantata and Sonata IRPs define pre-ordering and ordering functionalities that allow an automated exchange of information between business applications of the Buyer (Customer or Service Provider) and Seller (Service Provider or Partner) Domains. Those are:

- Product Catalog
- Address Validation
- Site Retrieval
- Product Offering Qualification
- Product Quote
- Product Inventory
- Product Ordering
- Trouble Ticketing
- Billing

The business requirements and use cases for Trouble Ticketing are defined in MEF W113 *Trouble Ticketing Requirements and Use Cases* [MEF113]. MEF W113 defines use cases that cover Trouble Ticket, Incident, Appointment, and WorkOrder. The scope of this API and Developer Guide covers the Trouble Ticket and Incident related use cases (based on the [TMF621] Trouble Ticket API). The Appointment and Work order use cases are covered by *LSO Cantata and LSO Sonata Appointment API Developer Guide* [MEF137].

This document is structured as follows:

- Chapter 4 provides an introduction to Trouble Ticketing and its description in a broader context of Cantata and Sonata and their corresponding SDKs.
- Chapter 5 gives an overview of endpoints, resource model and design patterns.
- Use cases and flows are presented in Chapter 6.
- And finally, Chapter 7 complements previous sections with a detailed API description.

4.1. Conventions in the Document

- Code samples are formatted using code blocks. When notation `<< some text >>` is used in the payload sample it indicates that a comment is provided instead of an example value and it might not comply with the OpenAPI definition.
- Model definitions are formatted as in-line code (e.g. `TroubleTicket`).
- In UML diagrams the default cardinality of associations is `0..1`. Other cardinality markers are compliant with the UML standard.
- In the API details tables and UML diagrams required attributes are marked with a `*` next to their names.
- In UML sequence diagrams `{{variable}}` notation is used to indicate a variable to be substituted with a correct value.

4.2. Relation to Other Documents

This API implements the Trouble Ticket related requirements and use cases that are defined in MEF 113 [MEF113]. The API definition builds on *TMF621 Trouble Ticket API REST Specification R19.0.1* [TMF621].

4.3. Approach

As presented in Figure 2. both Cantata and Sonata API frameworks consist of three structural components:

- Generic API framework
- Product-independent information (Function-specific information and Function-specific operations)
- Product-specific information (MEF product specification data model)

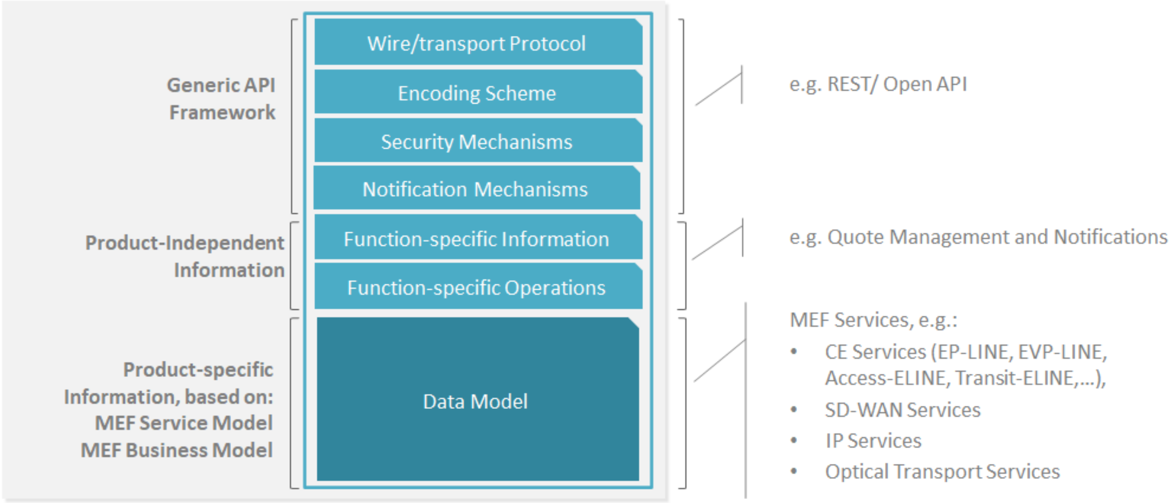

Generic API Framework
Product-Independent Information
Product-specific Information, based on: MEF Service Model MEF Business Model
Wire/transport Protocol
Encoding Scheme
Security Mechanisms
Notification Mechanisms
Function-specific Information
Function-specific Operations
Data Model
e.g. REST/ Open API
e.g. Quote Management and Notifications
MEF Services, e.g.:
CE Services (EP-LINE, EVP-LINE, Access-ELINE, Transit-ELINE,...),
SD-WAN Services
IP Services
Optical Transport Services

Figure 2. Cantata and Sonata API framework

The essential concept behind the framework is to decouple the common structure, information and operations from the specific product information content.
Firstly, the Generic API Framework defines a set of design rules and patterns that are applied across all Cantata or Sonata APIs.
Secondly, the product-independent information of the framework focuses on a model of a particular Cantata or Sonata functionality and is agnostic to any of the product specifications.
Finally, the product-specific information part of the framework focuses on MEF product specifications that define business-relevant attributes and requirements for trading MEF subscriber and MEF operator services.

The Trouble Ticket is product-agnostic in its nature and is not intended to carry any product-specific payloads. It only references products from the inventory by `id`. It operates using the Generic API Framework and the Function-specific Information and Operations.

4.4. High-Level Flow

Trouble Ticket is part of a broader Cantata and Sonata End-to-End flow. Figure 3. below shows a high-level diagram to get a good understanding of the whole process and Trouble Ticket's position within it.

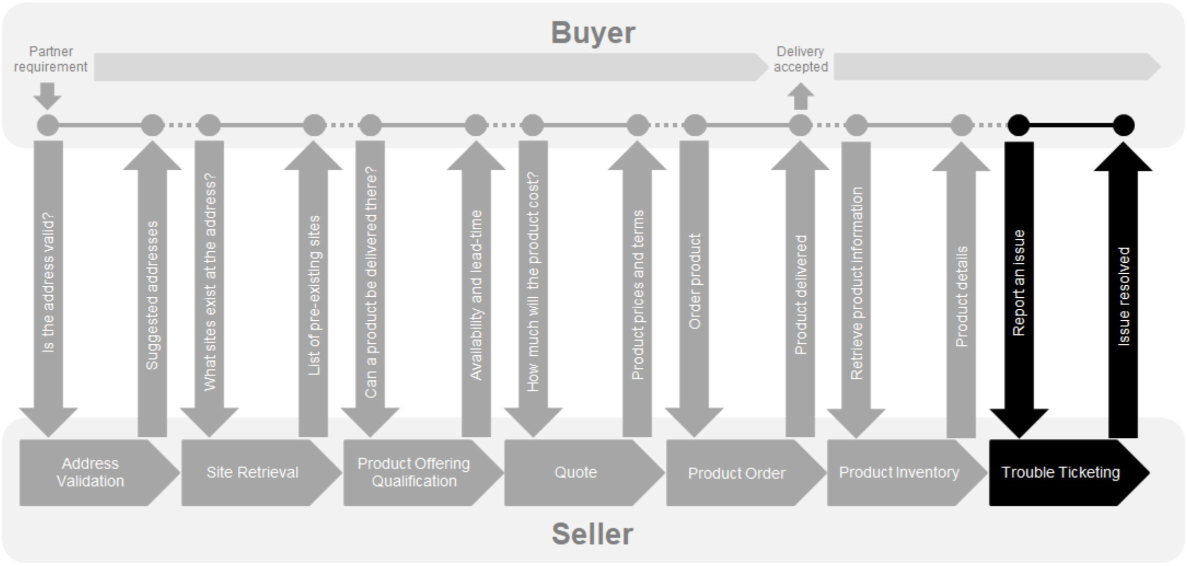


Figure 3. Cantata and Sonata End-to-End Function Flow

- Address Validation:
    - Allows the Buyer to retrieve address information from the Seller, including exact formats, for addresses known to the Seller.
- Site Retrieval:
    - Allows the Buyer to retrieve Geographic Site information including exact formats for Geographic Sites known to the Seller.
- Product Offering Qualification (POQ):
    - Allows the Buyer to check whether the Seller can deliver a product or set of products from among their product offerings at the geographic address or a Geographic Site specified by the Buyer; or modify a previously purchased product.
- Quote:
    - Allows the Buyer to submit a request to find out how much the installation of an instance of a Product Offering, an update to an existing Product, or a disconnect of an existing Product will cost.
- Product Order:
    - Allows the Buyer to request the Seller to initiate and complete the fulfillment process of an installation of a Product Offering, an update to an existing Product, or a disconnect of an existing Product at the address defined by the Buyer.
- Product Inventory:
    - Allows the Buyer to retrieve the information about the existing Product instances from Seller's Product Inventory.
- Trouble Ticketing:
    - Allows the Buyer to create, retrieve, and update Trouble Tickets as well as receive notifications about Incidents' and Trouble Tickets' updates. This allows managing issues and situations for a Product provided by the Seller.

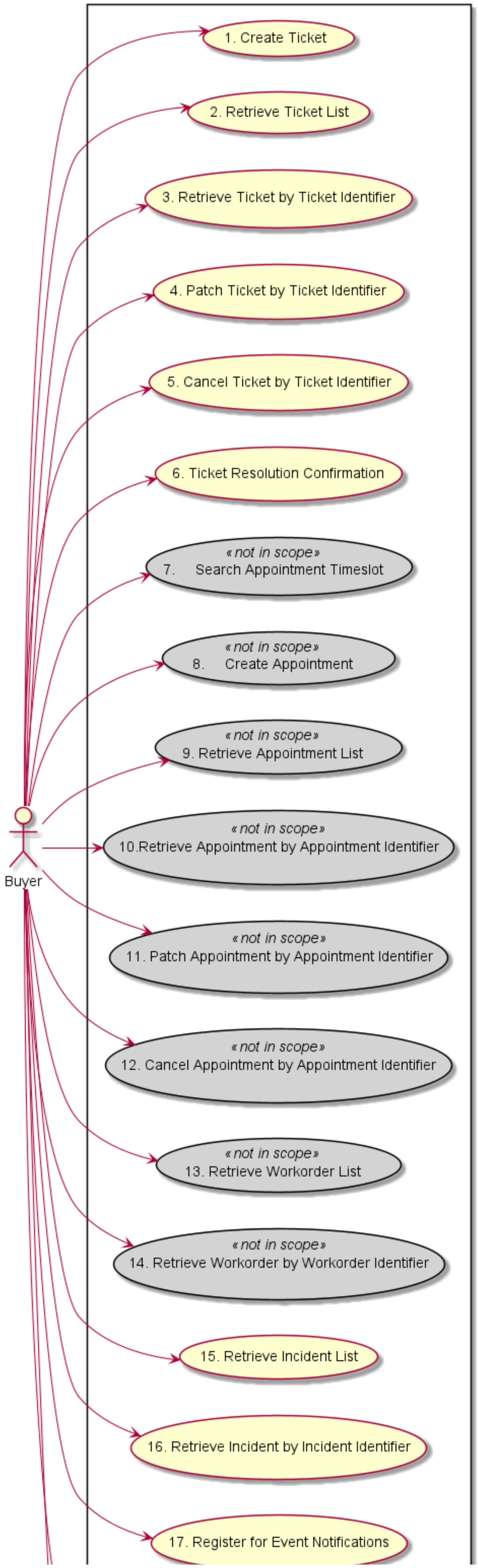
5. API Description

This section presents the API structure and design patterns. It starts with the high-level use cases diagram. Then it describes the REST endpoints with use case mapping. Next, it gives an overview of the API resource model.

5.1. High-level use cases

Figure 4 presents a high-level use case diagram as specified in MEF 113 [MEF113] in section 7. This picture aims to help understand the endpoint mapping. Use cases are described extensively in chapter 6.

Note: As stated earlier, the scope of this API does not cover the Appointment and WorkOrder related use cases. The diagram below lists all use cases defined in MEF 113 to highlight which of them are covered. For easier requirements matching this document keeps the original MEF 113 numbering. The remaining use cases are covered by *LSO Cantata and LSO Sonata Appointment API Developer Guide* [MEF137].


1. Create Ticket
2. Retrieve Ticket List
3. Retrieve Ticket by Ticket Identifier
4. Patch Ticket by Ticket Identifier
5. Cancel Ticket by Ticket Identifier
6. Ticket Resolution Confirmation
« not in scope »
7. Search Appointment Timeslot
« not in scope »
8. Create Appointment
« not in scope »
9. Retrieve Appointment List
« not in scope »
10.Retrieve Appointment by Appointment Identifier
« not in scope »
11. Patch Appointment by Appointment Identifier
« not in scope »
12. Cancel Appointment by Appointment Identifier
« not in scope »
13. Retrieve Workorder List
« not in scope »
14. Retrieve Workorder by Workorder Identifier
15. Retrieve Incident List
16. Retrieve Incident by Incident Identifier
17. Register for Event Notifications
Buyer

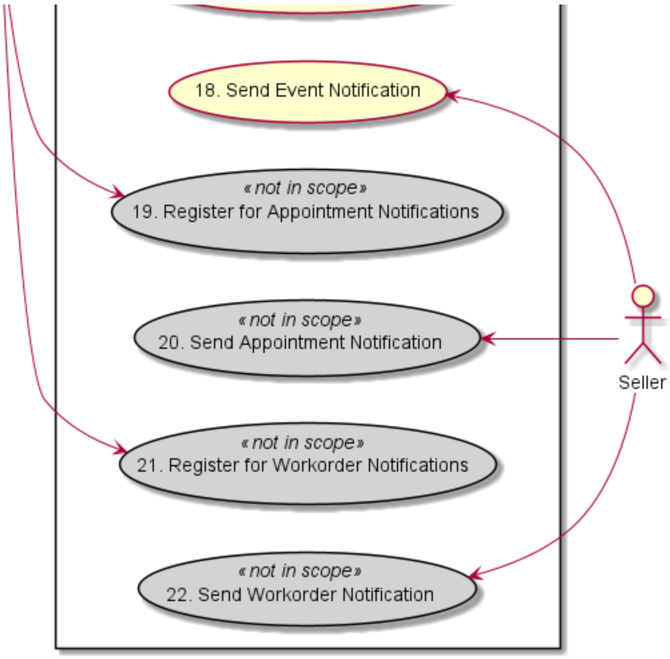

18. Send Event Notification
« not in scope »
19. Register for Appointment Notifications
« not in scope »
20. Send Appointment Notification
Seller
« not in scope »
21. Register for Workorder Notifications
« not in scope »
22. Send Workorder Notification

Figure 4: Use cases

5.2. API Endpoint and Operation Description

5.2.1. Seller side API Endpoints

Base URL for Cantata: `https://{{serverBase}}:{{port}}{{?/seller_prefix}}/mefApi/cantata/troubleTicket/v4/`

Base URL for Sonata: `https://{{serverBase}}:{{port}}{{?/seller_prefix}}}/mefApi/sonata/troubleTicket/v4/`

The following API endpoints are implemented by the Seller and allow the Buyer to create, retrieve, modify Trouble Tickets and register for Notifications. The endpoints and corresponding data model are defined in

`productApi/troubleTicket/troubleTicketManagement.api.yaml`.

API endpoint	**Description**	**MEF 113 Use Case mapping**
`POST /troubleTicket`	A request initiated by the Buyer to create a Ticket in the Seller's system to report an Issue experienced by the Buyer or their end user.	UC 1: Create Ticket
`GET /troubleTicket`	The Buyer requests a list of Tickets from the Seller based on a set of specified filter criteria. The Seller returns a summarized list of Tickets.	UC 2: Retrieve Ticket List

API endpoint	Description	MEF 113 Use Case mapping
`GET /troubleTicket/{{id}}`	The Buyer requests detailed information about a single Ticket based on a Ticket Identifier.	UC 3: Retrieve Ticket by Ticket Identifier
`PATCH /troubleTicket/{{id}}`	A request by the Buyer to patch/partial up-date a Ticket created by the Buyer in the Seller's system.	UC 4: Patch Ticket by Ticket Identifier
`POST /troubleTicket/{{id}}/cancel`	A request by the Buyer to cancel a Ticket created by the Buyer in the Seller's system.	UC 5: Cancel Ticket by Ticket Identifier
`POST /troubleTicket/{{id}}/close`	A request from the Buyer confirming whether they agree that a Ticket created by the Buyer in the Seller's system can be closed, since the reported Issue is no longer observed. This request is the action taken by a Buyer after receiving an Event Notification from the Seller with Notification Event Type `TroubleTicketResolvedEvent`.	UC 6: Ticket Resolution Confirmation
`POST /troubleTicket/{{id}}/reopen`	A request from the Buyer rejecting that a Ticket created by the Buyer in the Seller's system can be closed, because the reported Issue is still observed. This request is the action taken by a Buyer after receiving a Event Notification from the Seller with Notification Event Type `TroubleTicketResolvedEvent`.	UC 6: Ticket Resolution Confirmation
`POST /hub`	The Buyer requests to subscribe to notifications.	UC 17: Register for Event Notifications
`GET /hub/{{id}}`	A request initiated by the Buyer to retrieve the details of the notification subscription.	UC 17: Register for Event Notifications

API endpoint	Description	MEF 113 Use Case mapping
`DELETE /hub/{{id}}`	A request initiated by the Buyer to instruct the Seller to stop sending notifications.	UC 17: Register for Event Notifications

Table 4. Seller side mandatory API endpoints

[R1] The implementation **MUST** support API endpoints listed in Table 4. [MEF113 R1], [MEF113 R2]

API endpoint	Description	MEF 113 Use Case mapping
`GET /incident`	The Buyer requests a list of Incidents from the Seller based on a set of specified filter criteria. The Seller returns a summarized list of Incidents.	UC 15. Retrieve Incident List
`GET /incident/{{id}}`	The Buyer requests detailed information about a single Incident based on an Incident Identifier.	UC 16. Retrieve Incident by Incident Identifier

Table 5. Seller side optional API endpoints

[O1] The implementation **MAY** support API endpoints listed in Table 5. [MEF113 O1]

[CR1]<([O1]) If any of endpoints listed in Table 5 is supported, then all endpoints listed in Table 5 **MUST** be supported. [MEF113 [CR1]<[O1]]

5.2.2. Buyer side API Endpoints

Base URL for Cantata: `https://{{serverBase}}:{{port}}{{?/buyer_prefix}}/mefApi/cantata/troubleTicketNotification/v4/`

Base URL for Sonata: `https://{{serverBase}}:{{port}}{{?/buyer_prefix}}/mefApi/sonata/troubleTicketNotification/v4/`

The following API Endpoints are used by the Seller to post notifications to registered listeners. The endpoints and corresponding data model are defined in

`productApi/troubleTicket/troubleTicketNotification.api.yaml`

API Endpoint	Description	MEF 113 Use Case Mapping
`POST /listener/troubleTicketAttributeValueChangeEvent`	A request initiated by the Seller to notify the Buyer on `TroubleTicket` attribute value change.	UC 18: Send Event Notification
`POST /listener/troubleTicketStatusChangeEvent`	A request initiated by the Seller to notify the Buyer on `TroubleTicket.status` change.	UC 18: Send Event Notification
`POST /listener/troubleTicketResolvedEvent`	A request initiated by the Seller to notify the Buyer on `TroubleTicket` reaching the `resolved` status.	UC 18: Send Event Notification
`POST /listener/troubleTicketInformationRequiredEvent`	A request initiated by the Seller to notify the Buyer that and additional information is required for further Ticket processing	UC 18: Send Event Notification

Table 6. Buyer side mandatory API endpoints

[R2] The implementation **MUST** support API endpoints listed in Table 6. [MEF113 R2]

API Endpoint	Description	MEF 113 Use Case Mapping
`POST /listener/incidentCreateEvent`	A request initiated by the Seller to notify the Buyer on `Incident` creation	UC 18: Send Event Notification
`POST /listener/incidentAttributeValueChangeEvent`	A request initiated by the Seller to notify the Buyer on `Incident` attribute value change.	UC 18: Send Event Notification
`POST /listener/incidentStatusChangeEvent`	A request initiated by the Seller to notify the Buyer on `Incident.status` change.	UC 18: Send Event Notification

Table 7. Buyer side optional API endpoints

[O2] The implementation **MAY** support API endpoints listed in Table 7. [MEF113 O2]

[CR2]<([O1]]) If any of endpoints listed in Table 5 is supported, then the Seller **MUST** support all endpoints listed in Table 7. [MEF113 [CR2]<[O2]]

5.3. Specifying the Buyer ID and the Seller ID

A business entity willing to represent multiple Buyers or multiple Sellers must follow requirements of MEF 79 [MEF79] chapter 8.8, which states:

> For requests of all types, there is a business entity that is initiating an Operation (called a Requesting Entity) and a business entity that is responding to this request (called the Responding Entity). In the simplest case, the Requesting Entity is the Buyer and the Responding Entity is the Seller. However, in some cases, the Requesting Entity may represent more than one Buyer and similarly, the Responding Entity may represent more than one Seller.
>
> While it is outside the scope of this specification, it is assumed that the Requesting Entity and the Responding Entity are aware of each other and can authenticate requests initiated by the other party. It is further assumed that both the Buying Entity and the Requesting Entity know:
>
> a) the list of Buyers the Requesting Entity represents when interacting with this Responding Entity; and
> b) the list of Sellers that this Responding Entity represents to this Requesting Entity.

In the API the `buyerId` and `sellerId` are represented as query parameters in each operation defined in `troubleTicketManagement.api.yaml` and as attributes of events as described in `troubleTicketNotification.api.yaml`.

[R3] If the Requesting Entity has the authority to represent more than one Buyer the request **MUST** include `buyerId` query parameter that identifies the Buyer being represented [MEF79 R80]

[R4] If the Requesting Entity represents precisely one Buyer with the Responding Entity, the request **MUST NOT** specify the `buyerId` [MEF79 R81]

[R5] If the Responding Entity represents more than one Seller to this Buyer the request **MUST** include `sellerId` query parameter that identifies the Seller with whom this request is associated [MEF79 R82]

[R6] If the Responding Entity represents precisely one Seller to this Buyer, the request **MUST NOT** specify the `sellerId` [MEF79 R83]

[R7] If `buyerId` or `sellerId` attributes were specified in the request same attributes **MUST** be used in the notification payload.

5.4. Model Structural Validation

The structure of the HTTP payloads exchanged via Trouble Ticket API endpoints is defined using OpenAPI version 3.0.

[R8] Implementations **MUST** use payloads that conform to these definitions.

5.5. Security Considerations

There must be an authentication mechanism whereby a Seller can be assured who a Buyer is and vice-versa. There must also be authorization mechanisms in place to control what a particular Buyer or Seller is allowed to do and what information may be obtained. However, the definition of the exact security mechanism and configuration is outside the scope of this document. It is specified by a separate MEF Project (MEF 128 [MEF128]).

6. API Interactions and Flows

This section provides a detailed insight into the API functionality, use cases, and flows. It starts with Table 8 presenting a list and short description of all business use cases then presents the variants of end-to-end interaction flows, and in the following subchapters describes the API usage flow and examples for each of the use cases.

Table 8. lists the use cases supported by Trouble Ticket API (use case numbers as in MEF 113 for mapping):

Use Case #	Use Case Name	Use Case Description
1	Create Ticket	A request initiated by the Buyer to create a Ticket in the Seller's system to report an Issue experienced by the Buyer or their end-user.
2	Retrieve Ticket List	The Buyer requests a list of Tickets from the Seller based on a set of specified filter criteria. The Seller returns a summarized list of Tickets.
3	Retrieve Ticket by Ticket Identifier	The Buyer requests detailed information about a single Ticket based on a Ticket Identifier.
4	Patch Ticket by Ticket Identifier	A request by the Buyer to patch/partial update a Ticket based on a Ticket Identifier.
5	Cancel Ticket by Ticket Identifier	A request by the Buyer to cancel a Ticket based on a Ticket Identifier.
6	Ticket Resolution Confirmation	A reply from the Buyer confirming whether they agree that a Ticket can be closed, since the reported Issue is no longer observed. This reply is the action taken by a Buyer after receiving an Event Notification from the Seller with Event Notification Type TICKET_RESOLVED.
15	Retrieve Incident List	The Buyer requests a list of Incidents from the Seller based on a set of specified filter criteria. The Seller returns a summarized list of Incidents.

Use Case #	Use Case Name	Use Case Description
16	Retrieve Incident by Incident Identifier	The Buyer requests detailed information about a single Incident based on an Incident Identifier.
17	Register for Event Notifications	The Buyer requests to subscribe to Ticket and Incident Notifications.
18	Send Event Notification	Send Event Notification The Seller sends a notification regarding a Ticket or Incident to the Buyer

Table 8. Use cases description

MEF 113 defines use cases related to three domains:

- Trouble Ticket
- WorkOrder
- Appointment

Figure 5 presents an example of an end-to-end flow that shows dependencies between all the domains:

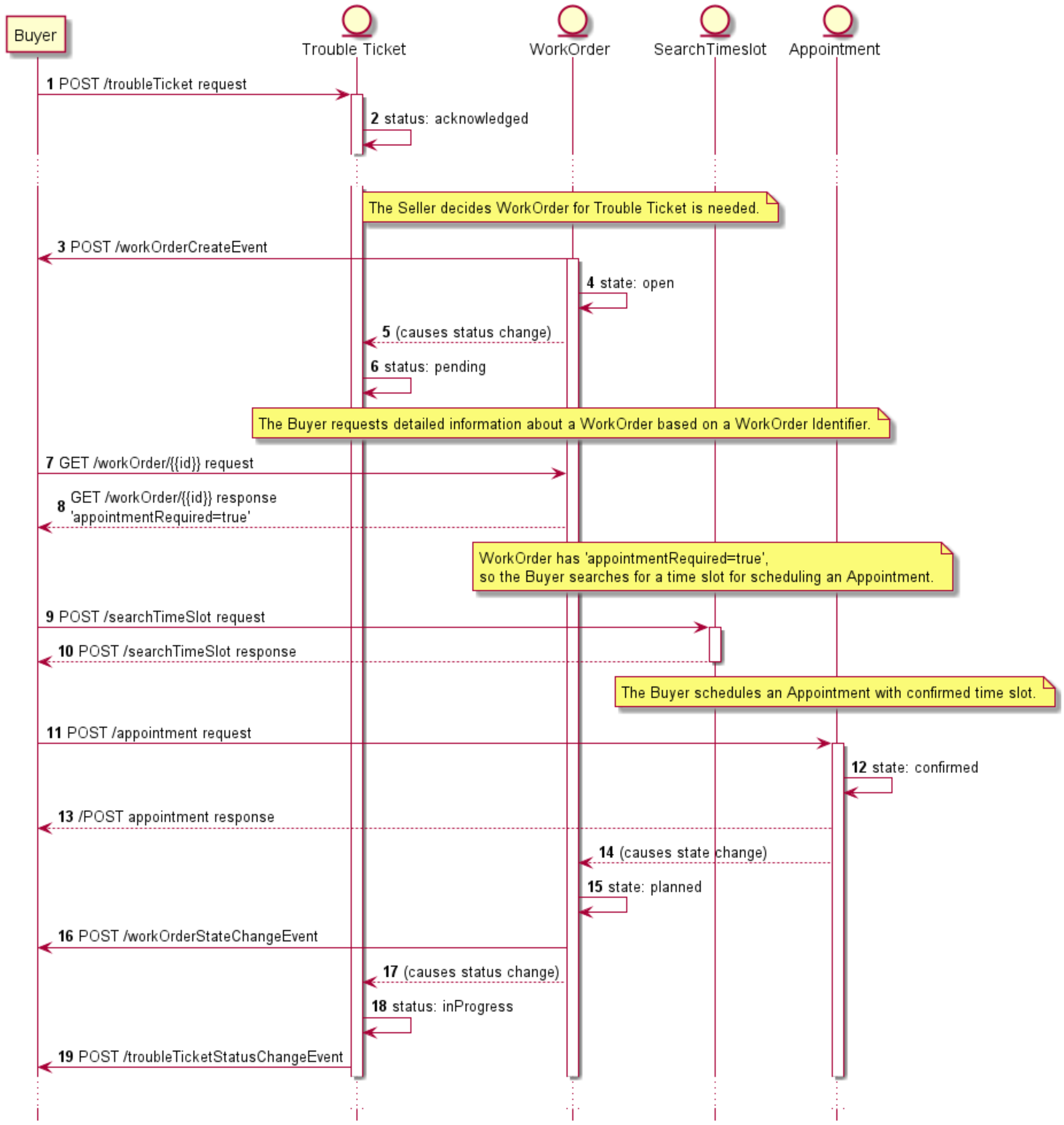

Buyer
Trouble Ticket
WorkOrder
SearchTimeslot
Appointment
1 POST /troubleTicket request
2 status: acknowledged
The Seller decides WorkOrder for Trouble Ticket is needed.
3 POST /workOrderCreateEvent
4 state: open
5 (causes status change)
6 status: pending
The Buyer requests detailed information about a WorkOrder based on a WorkOrder Identifier.
7 GET /workOrder/{{id}} request
8 GET /workOrder/{{id}} response 'appointmentRequired=true'
WorkOrder has 'appointmentRequired=true', so the Buyer searches for a time slot for scheduling an Appointment.
9 POST /searchTimeSlot request
10 POST /searchTimeSlot response
The Buyer schedules an Appointment with confirmed time slot.
11 POST /appointment request
12 state: confirmed
13 /POST appointment response
14 (causes state change)
15 state: planned
16 POST /workOrderStateChangeEvent
17 (causes status change)
18 status: inProgress
19 POST /troubleTicketStatusChangeEvent

Figure 5. End-to-End API flow with Workorder and Appointment

- (1) The Buyer experiences the issue in the network and creates the Trouble Ticket.
- (2) The Seller creates the Trouble Ticket and sets the status: `acknowledged`.
- The Seller decides that a WorkOrder with Appointment is needed to resolve the issue.
- The Seller creates a Workorder in state `open` (4) and sends a `workOrderCreateEvent` (3).
- (7-8) The Buyer requests detailed information about the WorkOrder.
- (9) The Buyer proposes time slots for scheduling an Appointment, if the WorkOrder requires the Appointment (the parameter set to `appointmentRequired=true`)
- (10) The Seller responds with the list of available time slots.
- (11) The Buyer schedules an Appointment with agreed time slot.
- (12) The Buyer sets the Appointment status to `confirmed`.
- (14-15) Appointment creation causes the WorkOrder state change to `planned`
- (17-18) WorkOrder state change to `planned` causes the Trouble Ticket status change back to `in_progress`.

The detailed business requirements of each of the use cases are described in sections 7 and 8 of MEF 113 [MEF113].

6.1. Use case 1: Create Ticket

This is the initial step for Trouble Ticket processing.

6.1.1. Interaction flow

The flow of this use case is very simple and is described in Figure 6.

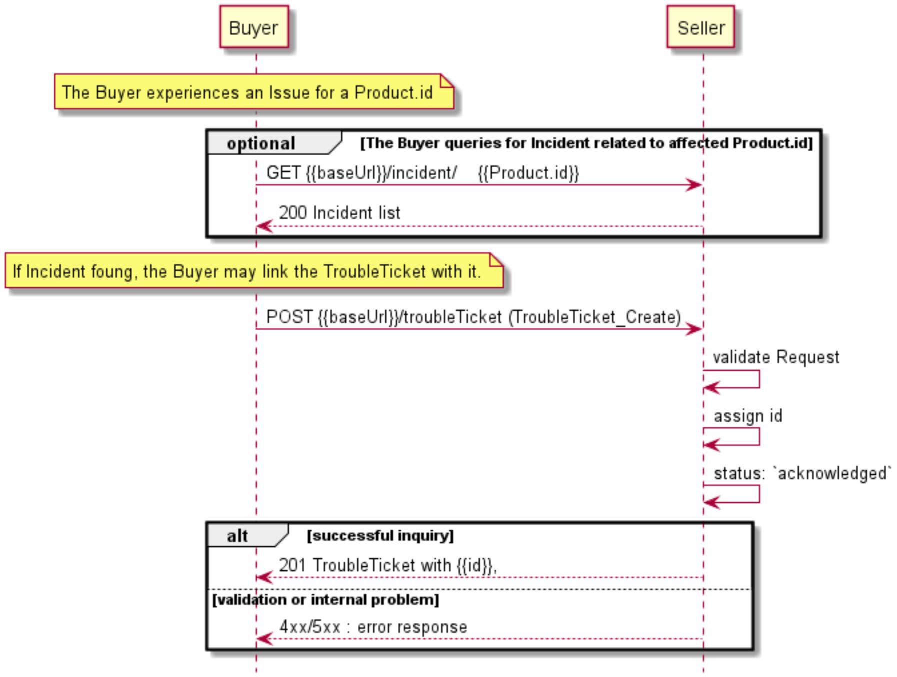

Buyer
Seller
The Buyer experiences an Issue for a Product.id
optional
[The Buyer queries for Incident related to affected Product.id]
GET {{baseUrl}}/incident/ {{Product.id}}
200 Incident list
If Incident foung, the Buyer may link the TroubleTicket with it.
POST {{baseUrl}}/troubleTicket (TroubleTicket_Create)
validate Request
assign id
status: `acknowledged`
alt
[successful inquiry]
201 TroubleTicket with {{id}},
[validation or internal problem]
4xx/5xx : error response

Figure 6: Use Case 1 - Trouble Ticket create request flow

The Buyer experiences an Issue with a Product (Identified by Product.id) and may decide to check if there is any Incident related to the affected Product. If yes, the Buyer may decide to link it with the new Ticket. The Buyer sends a request with a `TroubleTicket_Create` type in the body. The Seller performs request validation, assigns an `id`, and returns `TroubleTicket` type in the response body, with a `status` set to `acknowledged`. From this point, the Trouble Ticket is ready for further processing. The Buyer must track the progress of the process by subscribing for notifications (see chapter 6.9). The flow example with the use of Notifications is presented in Figure 7

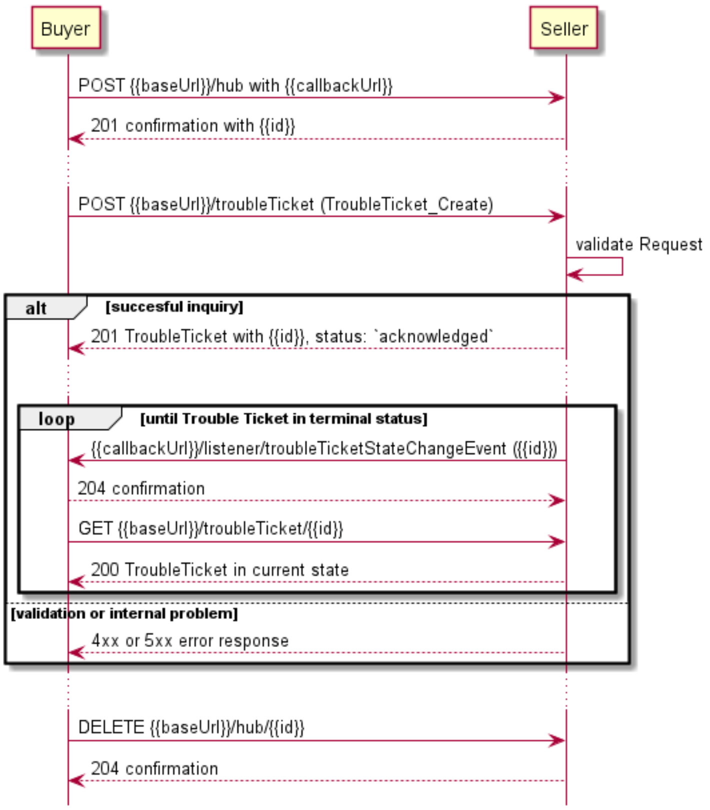

Buyer
Seller
POST {{baseUrl}}/hub with {{callbackUrl}}
201 confirmation with {{id}}
POST {{baseUrl}}/troubleTicket (TroubleTicket_Create)
validate Request
alt
[succesful inquiry]
201 TroubleTicket with {{id}}, status: `acknowledged`
loop
[until Trouble Ticket in terminal status]
{{callbackUrl}}/listener/troubleTicketStateChangeEvent ({{id}})
204 confirmation
GET {{baseUrl}}/troubleTicket/{{id}}
200 TroubleTicket in current state
[validation or internal problem]
4xx or 5xx error response
DELETE {{baseUrl}}/hub/{{id}}
204 confirmation

Figure 7: Trouble Ticket progress tracking - Notifications

Note: The context of notifications is not a part of the considered use case itself. It is presented to show the big picture of end-to-end flow. This applies also to all further use case flow diagrams with notifications.

6.1.2. Create Trouble Ticket - Request

Figure 8 presents the data model of the Trouble Ticket. The model of the request message (`TroubleTicket_Create`) is a subset of the `TroubleTicket` model and contains only attributes that can (or must) be set by the Buyer. The Seller then enriches the entity in the response with additional information. For visibility of these shared attributes, the `TroubleTicket_Common` has been introduced. Though, it is not to be used directly in the payload.

The full list of attributes is available in Section 7 and in the API specification which is an integral part of this standard.

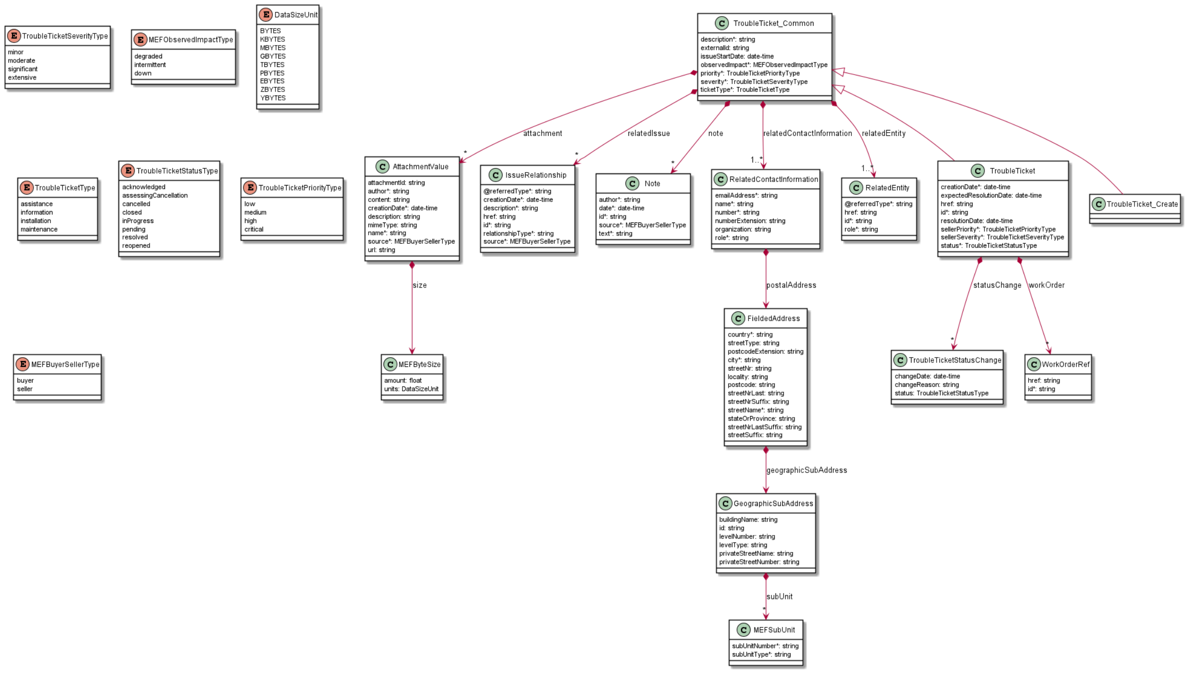


Figure 8: Create Trouble Ticket Model

The snippet below presents an example of the Create Trouble Ticket Request:

`TroubleTicket` Create

```
{
  "description": "Connection is lost",
  "externalId": "BuyerTicket-123",
  "issueStartDate": "2021-06-02T14:21:11.090Z",
  "priority": "critical",
  "severity": "extensive",
  "ticketType": "failure",
  "attachment": [
    {
      "attachmentId": "att-001",
      "author": "John Example",
      "creationDate": "2021-06-02T14:21:11.090Z",
      "description": "Print screen from the assurance system",
      "mimeType": "image/jpeg",
      "name": "Alarm",
      "url": "https://example.com/documents/00000000-0000-1111-2222-000000001111",
      "size": {
        "amount": 5.3,
        "units": "MBYTES"
      },
      "source": "buyer"
    }
  ],
  "note": [
    {
      "id": "note-1",
      "author": "John Example",
      "date": "2021-06-02T14:25:11.090Z",
      "source": "buyer",
      "text": "Couldn't reach the support on phone."
    }
  ],
  "relatedEntity": [ <<A relation to a Product that this Ticket refers to>>
    {
      "id": "01494079-6c79-4a25-83f7-48284196d44d",
      "role": "Issue Source",
      "@referredType": "Product"
    }
  ],
  "relatedContactInformation": [
    {
      "emailAddress": "john.example@example.com",
      "name": "John Example",
      "number": "+12-345-678-90",
      "organization": "Buyer Example Co.",
      "role": "reporterContact"
    }
  ]
}
```

[R9] The Buyer's Create request **MUST** include the following attributes: [MEF113 R31]

- `description`
- `observedImpact`
- `priority`
- `relatedContactInformation` item with a `role` set to `reporterContact`
- `relatedEntity` - (pointer to related Product instance)
- `severity`
- `ticketType`

Note: During the onboarding the Seller may require to provide an additional contact `role`.

Note: It is up to the Seller's discretion on how to react in case the Buyer provides a contact `role` that is not listed by this standard or agreed upon during the onboarding. Preferably the Seller should return an error with a message stating which `roles` are accepted. It may also be ignored

Note: The `relatedEntity` attribute is used to provide the related product `id`. It is done by setting the additional `@referredType` to `Product`. This follows the TMF pattern which enables compliance and allows referring also other potential types in MEF (e.g. `Service`). In this version, the only type that is mentioned in the implemented requirements document is the `Product` and to ease the request `RelatedEntity.@ReferredType` and the `relatedEntityType` in the filter criteria has a default value: `Product`.

[R10] If the `attachment` is provided, either the `attachment.url` or (`attachment.content` and `attachment.mimeType`) **MUST** be specified. [MEF113 R18], [MEF113 R19]

6.1.3. Create Trouble Ticket - Response

The Seller responds with a `TroubleTicket` type, which adds some attributes to the `TroubleTicket_Create` that was used in the Buyer's request.

Note: The term "Seller Response Code" used in the Business Requirements maps to HTTP response code, where `2xx` indicates *Success* and `4xx` or `5xx` indicate *Failure*.

The following snippet presents the Seller's response. It has the same structure as in the retrieve by identifier operation.

```
{
  "id": "00000000-4444-5555-6666-000000000987",
  "href": "{{baseUrl}}/troubleTicket/00000000-4444-5555-6666-000000000987",
  "creationDate": "2021-06-02T20:56:08.559Z",
  "expectedResolutionDate": "2021-06-03T20:56:08.559Z",
  "lastUpdate": "2021-06-02T20:56:08.559Z",
  "sellerPriority": "critical",
  "sellerSeverity": "extensive",
  "status": "acknowledged",
  "description": "Connection is lost", << as provided by the Buyer >>
  "externalId": "BuyerTicket-123", << as provided by the Buyer >>
  "issueStartDate": "2021-06-02T14:21:11.090Z", << as provided by the Buyer >>
  "priority": "critical", << as provided by the Buyer >>
  "severity": "extensive", << as provided by the Buyer >>
  "ticketType": "failure", << as provided by the Buyer >>
  "attachment": [
    { << as provided by the Buyer >>
      "attachmentId": "att-001",
      "author": "John Example",
      "creationDate": "2021-06-02T14:21:11.090Z",
      "description": "Print screen from the assurance system",
      "mimeType": "image/jpeg",
      "name": "Alarm",
      "url": "https://example.com/documents/00000000-0000-1111-2222-000000001111",
      "size": {
        "amount": 5.3,
        "units": "MBYTES"
      },
      "source": "buyer"
    }
  ],
  "note": [
    {<< as provided by the Buyer >>
      "id": "note-1",
      "author": "John Example",
      "date": "2021-06-02T14:25:11.090Z",
      "source": "buyer",
      "text": "Couldn't reach the support on phone."
    }
  ],
  "relatedEntity": [
    {<< as provided by the Buyer >>
      "id": "01494079-6c79-4a25-83f7-48284196d44d",
      "role": "Issue Source",
      "@referredType": "Product"
    }
  ],
  "relatedContactInformation": [
    {<< as provided by the Buyer >>
      "emailAddress": "john.example@example.com",
      "name": "John Example",
      "number": "+12-345-678-90",
      "organization": "Buyer Example Co.",
      "role": "reporterContact"
    },
    {<< a new item appended by the Seller >>
      "emailAddress": "Seller.TicketContact@example.com",
      "name": "Seller Ticket Contact",
      "number": "+98-765-432-10",
      "organization": "Seller Example Co.",
      "role": "sellerTicketContact"
    }
  ],
  "relatedIssue": [
    {
      "@referredType": "TroubleTicket",
      "id": "00000000-1234-4321-1111-00000000888",
      "creationDate": "2021-06-02T20:56:08.559Z",
      "description": "The issue is caused by.",
      "relationshipType": "caused by",
      "source": "seller"
    }
  ],
  "statusChange": [
    {
      "changeDate": "2021-06-02T20:56:08.560Z",
      "status": "acknowledged"
    }
  ]
}
```

The response to the create request does not contain all possible attributes, for example, the `resolutionDate` is valid only in the future lifecycle of the Trouble Ticket.

[R11] The Seller's response **MUST** include all and unchanged attributes' values as provided in the request. [MEF113 R33]

These attributes are indicated above with an appropriate comment: `<< as provided by the Buyer >>`.

[R12] The Seller **MUST** specify the following attributes in a response: [MEF113 R35]

- `creationDate`
- `id`
- `relatedContactInformation` - item with a `role` set to `sellerTicketContact`
- `sellerSeverity`
- `sellerPriority`
- `status`

[R13] The `status` of the Ticket in the Seller's response **MUST** be `acknowledged`. [MEF113 R34]

6.1.4. Trouble Ticket - Lifecycle

Figure 9 presents the Trouble Ticket state machine:

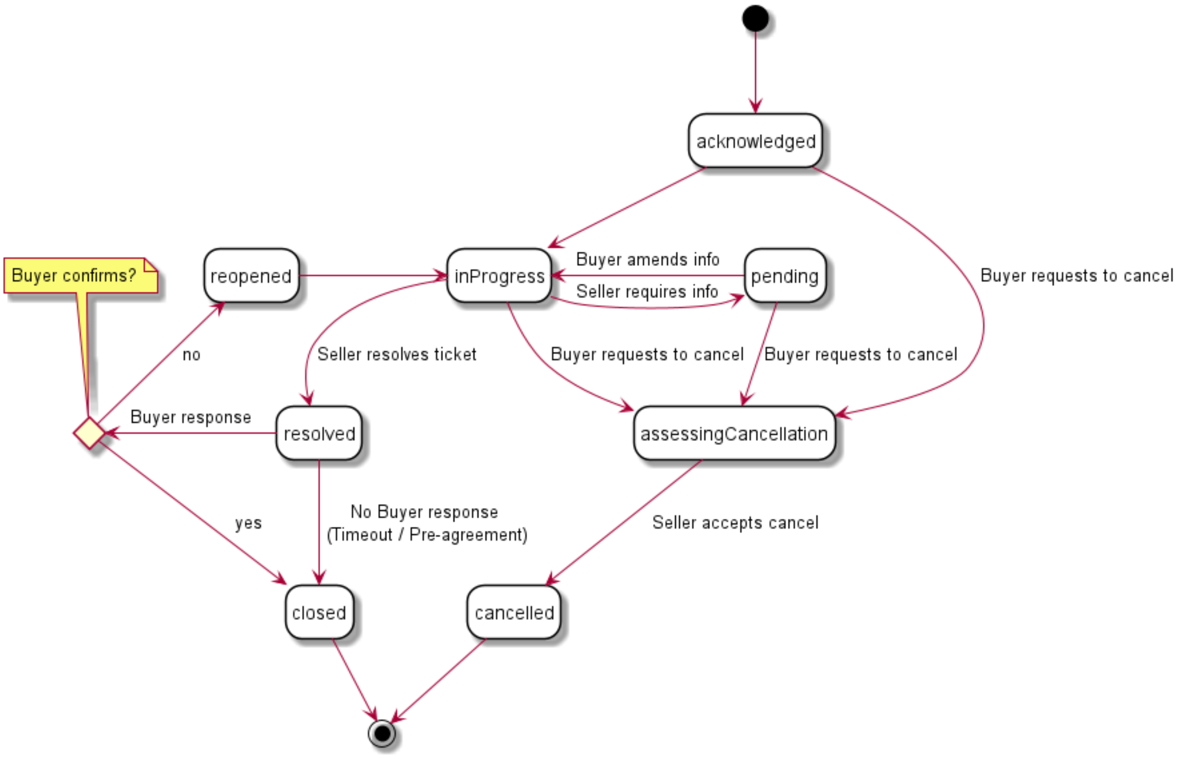


Figure 9: Trouble Ticket State Machine

After receiving the request, the Seller performs a validation of the message. If any problem is found an Error response is provided. If the validation passes a response is provided with `TroubleTicket` in `acknowledged` status. Then the Seller starts working on resolving the issue and moves the Trouble Ticket to `inProgress` state. From there, additional information might be required to proceed and the Trouble Ticket moves to `pending` until one is provided. The Trouble Ticket is set as `resolved` when the Seller claims the issue is fixed. From there the Buyer can either reopen or close the Ticket (use cases described in following sections). The Buyer may also request for a Trouble Ticket to be cancelled, while in `acknowledged`, `pending`, or `inProgress` state.

Table 9 presents the mapping between the API `status` names (aligned with TMF) and the MEF 113 naming, together with statuses' description.

status	MEF 113 name	Description
`acknowledged`	ACKNOWLEDGED	A request to create a Ticket was received and accepted by the Seller. The Ticket create request has been validated and a Ticket has been created by the Seller and allocated a unique `id`.
`inProgress`	IN_PROGRESS	The Ticket is in the process of being handled and investigated for resolution by the Seller.
`resolved`	RESOLVED	The Buyer's Issue described in the Ticket was resolved by the Seller. The Seller assumes that normal operation is re-established for the Buyer's product and i snow waiting for the Buyer to confirm that the Issue they reported is no longer observed.
`closed`	CLOSED	The Buyer has confirmed that the Issue they reported is no longer observed, or the pre-defined time frame (agreed upon between Buyer and Seller) for confirming that the Issue has been resolved has passed without a response by the Buyer. This is a terminal state.

status	MEF 113 name	Description
`reopened`	REOPENED	The Buyer has verified that the Issue described in the Ticket is still observed and has not been resolved satisfactorily. The Buyer rejects the Seller's request to close the Ticket. The Ticket has been reopened and is waiting for further actions from the Seller.
`pending`	PENDING	The Seller is waiting on the Buyer to provide additional information for the Ticket, or the Buyer to schedule an Appointment for the WorkOrder (linked to the Ticket) in order to continue processing the Ticket. This may result in the clock being stopped for the service level agreement until the Buyer has responded to the request.
`assessingCancellation`	ASSESSING_CANCELLATION	A request has been made by the Buyer to cancel the Ticket and is being assessed by the Seller to determine whether to just close the Ticket, or continue to resolve the Issue to prevent similar Create Ticket requests from other Buyers. If the Seller chooses to resolve the Issue, the Seller might create an Incident or an internal Ticket for the Issue, but that is outside the scope of this document. After the Seller has completed the assessment, the Seller updates the Ticket State to `cancelled`.
`cancelled`	CANCELLED	The Ticket has been successfully cancelled by the Buy-er. The Buyer will receive no further Event Notifications for the Ticket. This is a terminal state.

Table 9: Trouble Ticket statuses

[R14] The Seller **MUST** support all Trouble Ticket statuses and their associated transitions as described in Figure 9 and Table 9. [MEF113 R155]

[R15] If the Trouble Ticket was in `pending` status and an Appointment is created and the related WorkOrder moves to `planned` state, the Seller **MUST** update the Trouble Ticket status to `inProgress`. [MEF113 R91]

[R16] The Buyer **MUST** set the respective `source=buyer` attribute when adding any item to one of the following list: `note`, `attachment`, or `relatedIssue`. [MEF113 R8], [MEF113 R14], [MEF113 R23]

[R17] The Buyer **MUST NOT** set the respective `source=seller` attribute when adding any item to one of the following list: `note`, `attachment`, or `relatedIssue`. [MEF113 R9], [MEF113 R15], [MEF113 R24]

[R18] The Seller **MUST** set the `source=seller` when adding any item to one of the following list: `note`, `attachment`, or `relatedIssue`. [MEF113 R6], [MEF113 R12], [MEF113 R21]

[R19] The Seller **MUST NOT** set the `source=buyer` when adding any item to one of the following list: `note`, `attachment`, or `relatedIssue`. [MEF113 R7], [MEF113 R13], [MEF113 R22]

[R20] Any item in a `note` or `attachment` list **MUST NOT** be modified or deleted once added. [MEF113 R10], [MEF113 R16], [MEF113 R52], [MEF113 R56]

[O3] The Seller **MAY** append an item to `note`, `attachment`, or `relatedIssue` if required. [MEF113! O8], [MEF113! O9], [MEF113! O11]

[O4] The Seller **MAY** add, modify, or delete an item in `relatedContactInformation` with `role=sellerTechnicalContact` if the Ticket State is in `acknowledged`, `inProgress`, `reopened`, `pending` or `assessingCancellation`. [MEF113 O10]

[O5] The Seller **MAY** add or modify an item in `workOrder` list. [MEF113 O11]

[R21] The Seller **MUST NOT** modify or delete any items provided by the Buyer in following lists: `relatedContactInformation`, `note`, `attachment`, `relatedEntity`, or `relatedIssue`. [MEF113 R7], [MEF113 R37].

[R22] The Seller **MUST** add a `note` when any of the following Trouble Ticket attributes are updated: [MEF113 R36]

- `expectedResolutionDate`
- `relatedIssue`

6.2. Use Case 2: Retrieve Ticket List

[O6] The Buyer **MAY** retrieve a list of Trouble Tickets by using a `GET /troubleTicket` operation with desired filtering criteria. The attributes that are available to be used are: [MEF113 O12]

- `externalId`
- `priority`
- `sellerPriority`
- `severity`
- `sellerSeverity`
- `ticketType`
- `status`
- `observedImpact`
- `relatedEntityId`
- `relatedEntityType`
- `creationDate.gt`
- `creationDate.lt`
- `expectedResolutionDate.gt`
- `expectedResolutionDate.lt`
- `resolutionDate.gt`
- `resolutionDate.lt`

The Buyer may also ask for pagination with the use of the `offset` and `limit` parameters. The filtering and pagination attributes must be specified in URI query format RFC3986. Section 7.1.2. provides details about the implementation of pagination mechanism.

```
https://serverRoot/mefApi/sonata/troubleTicket/v2/troubleTicket?
status=inProgress&priority=critical&limit=10&offset=0
```

The example above shows a Buyer's request to get all Trouble Tickets that are in the `inProgress` status and with `critical` priority. Additionally, the Buyer asks only for a first (`offset=0`) pack of 10 results (`limit=10`) to be returned. The correct response (HTTP code `200`) in the response body contains a list of `TroubleTicket_Find` objects matching the criteria. To get more details (e.g. the item level information), the Buyer has to query a specific `TroubleTicket` by `id`.

[R23] The Seller **MUST** put the following attributes (if set) into the `TroubleTicket_Find` object in the response: [MEF113 R39]:

- `id`
- `externalId`
- `relatedEntity`
- `observedImpact`
- `priority`
- `sellerPriority`
- `severity`
- `sellerSeverity`

- ticketType
- status
- creationDate
- expectedResolutionDate
- resolutionDate

[R24] In case no items matching the criteria are found, the Seller **MUST** return a valid response with an empty list.

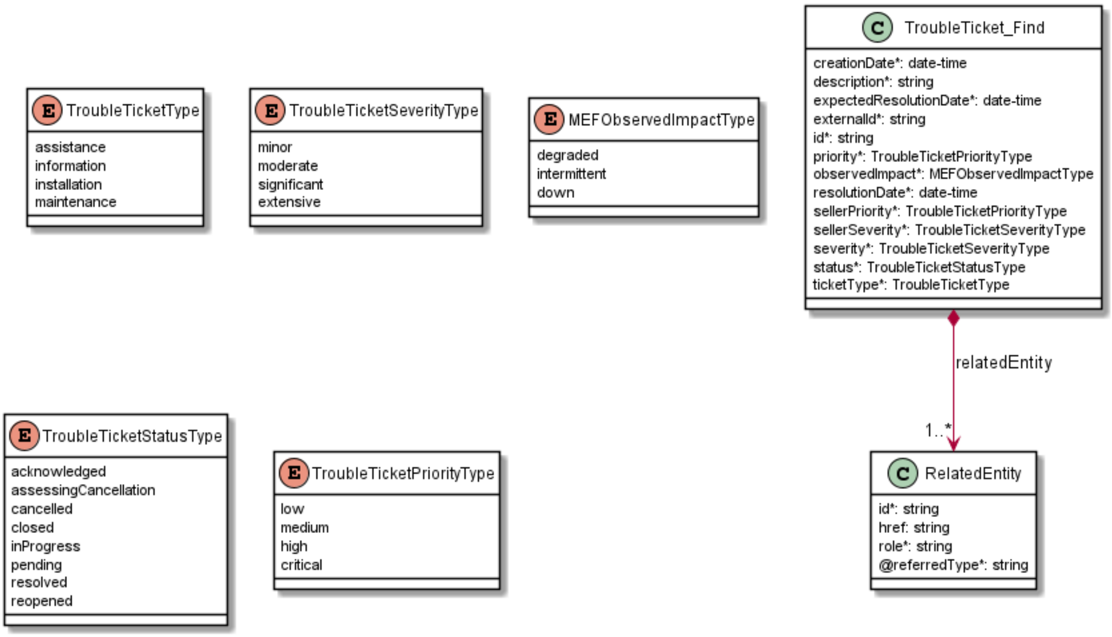


Figure 10: Use Case 2: Retrieve Ticket List - Model

6.3. Use Case 3: Retrieve Ticket by Ticket Identifier

The Buyer can get detailed information about the Trouble Ticket from the Seller by using a `GET /troubleTicket/{{id}}` operation.

[R25] In case `id` does not allow to find a `TroubleTicket` instance in Seller's system, an error response `Error404` **MUST** be returned. [MEF113 R42]

[R26] The Seller **MUST** put the following attributes into the `TroubleTicket` object in the response: [MEF113 R44]

- id
- relatedEntity
- description
- observedImpact
- priority
- sellerPriority
- severity
- sellerSeverity

- `ticketType`
- `status`
- `creationDate`
- `relatedContactInformation`

[R27] The Seller **MUST** provide all remaining optional attributes if they were previously set by the Buyer or the Seller. [MEF113 R45]

[R28] The Seller's response to a Retrieve Ticket by Ticket Identifier request **MUST** include the `resolutionDate` and a `note` added by the Seller describing how the Ticket was resolved if the `status` is `closed` or `resolved`. [MEF113 R46]

6.4. Use Case 4: Patch Ticket by Ticket Identifier

The update operation is realized with the use of the REST PATCH operation. For that purpose, a specialized type `TroubleTicket_Update` is provided. It consists of attributes limited to a subset that includes only the Buyer updateable attributes.

The PATCH usage recommendation follows TMF 621 json/merge (https://tools.ietf.org/html/rfc7386).

Figure 11 presents the model used in the PATCH request. The Seller responds with a `TroubleTicket` type.

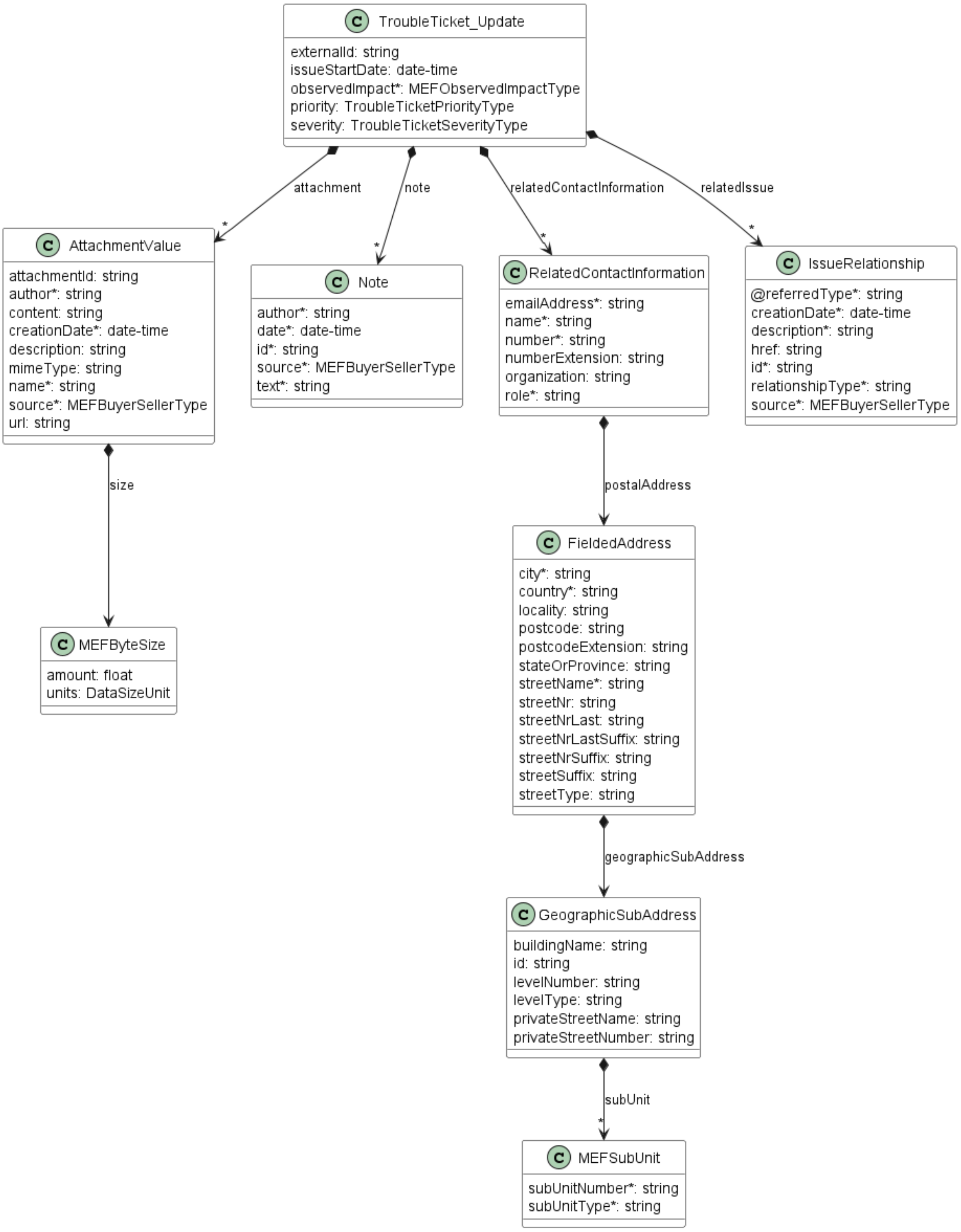


Figure 11: Patch request Model

[R29] The Buyer **MUST** include at least one of the following attributes of `TroubleTicket_Update` in the PATCH request: [MEF113 R48]

- `externalId`
- `priority`
- `severity`
- `issueStartDate`
- `observedImpact`
- `attachment` - append only

- `note` - append only
- `relatedContactInformation` - append or modify the Buyer settable contacts
- `relatedIssue`

[R30] The Buyer **MUST** add a `note` to a Trouble Ticket when any of the following attributes are patched: [MEF113 R49]

- `priority`
- `severity`
- `issueStartDate`
- `relatedIssue`

[R31] If the new item in the `attachment` list is provided, either the `attachment.url` or (`attachment.content` and `attachment.mimeType`) **MUST** be specified. [MEF113! R54]

[R32] The Buyer **MUST NOT** modify or delete any items provided by the Seller in following lists: `note`, `attachment`, `relatedContactInformation`, or `relatedIssue`. [MEF113 R51], [MEF113 R52]

Note: The Buyer can add or update items in the above-mentioned lists by providing a full list of existing items, and appending them with new ones or updating values of existing ones (where possible).

Note: As stated before, items to the `attachment` and `note` lists may only be added.

[R33] In case `id` does not allow to find a `TroubleTicket` that is to be updated in Seller's system, an error response `Error404` **MUST** be returned. [MEF113 R53]

[R34] The Seller **MUST** return an error (`Error422`) if attributes requested to be changed by the Buyer cannot be updated. [MEF113 R54]

[R35] The Seller **MUST** return an error (`Error422`) if the Ticket `state` is `closed`, `assessingCancellation` or `cancelled`. [MEF113 R55]

The example below shows a request to patch a `TroubleTicket` that was created in section 6.1.3. The first snippet provides the existing state of the `TroubleTicket`, showing only parts that are to be updated:

```
{
  ...
  "note": [
    {<< provided by the Buyer >>
      "id": "note-1",
      "author": "John Example",
      "date": "2021-06-02T14:25:11.090Z",
      "source": "buyer",
      "text": "Couldn't reach the support on phone."
    }
  ],
  "relatedContactInformation": [
    {<< provided by the Buyer >>
      "emailAddress": "john.example@example.com",
      "name": "John Example",
      "number": "+12-345-678-90",
      "organization": "Buyer Example Co.",
      "role": "reporterContact"
    },
    {<< a new item appended by the Seller >>
      "emailAddress": "Seller.TicketContact@example.com",
      "name": "Seller Ticket Contact",
      "number": "+98-765-432-10",
      "organization": "Seller Example Co.",
      "role": "sellerTicketContact"
    }
  ],
  ...
}
```

The request below aims to:

- add a new `note` (existing cannot be modified or deleted)
- change details of Buyer's `reporterContact`

```
{
  "note": [
    {<<previously existing>>
      "id": "note-1",
      "author": "John Example",
      "date": "2021-06-02T14:25:11.090Z",
      "source": "buyer",
      "text": "Couldn't reach the support on phone."
    },
    {<<added new note>>
      "id": "note-2",
      "author": "Kate Example",
      "date": "2021-06-02T19:25:11.090Z",
      "source": "buyer",
      "text": "Support reached after 5 hours"
    }
  ],
  "relatedContactInformation": [
    {<< update details of reporterContact >>
      "emailAddress": "Kate.example@example.com",
      "name": "Kate Example",
      "number": "+12-345-678-91",
      "organization": "Buyer Example Co.",
      "role": "reporterContact"
    },
    {<< provided by Seller - untouched >>
      "emailAddress": "Seller.TicketContact@example.com",
      "name": "Seller Ticket Contact",
      "number": "+98-765-432-10",
      "organization": "Seller Example Co.",
      "role": "sellerTicketContact"
    }
  ]
}
```

[R36] The Seller **MUST NOT** delete item from the `workOrder` list. [MEF113 R57]

[R37] If the Trouble Ticket status was `pending`, the Seller **MUST** update it to `inProgress`. [MEF113 R60]

6.5. Use case 5: Cancel Ticket by Ticket Identifier

The Buyer may request to cancel a Trouble Ticket by using `POST /troubleTicket/{{id}}/cancel` endpoint. This operation only requires providing the `id` in the path and has an empty `204` confirmation response.

The sequence diagram below presents this use case in detail.

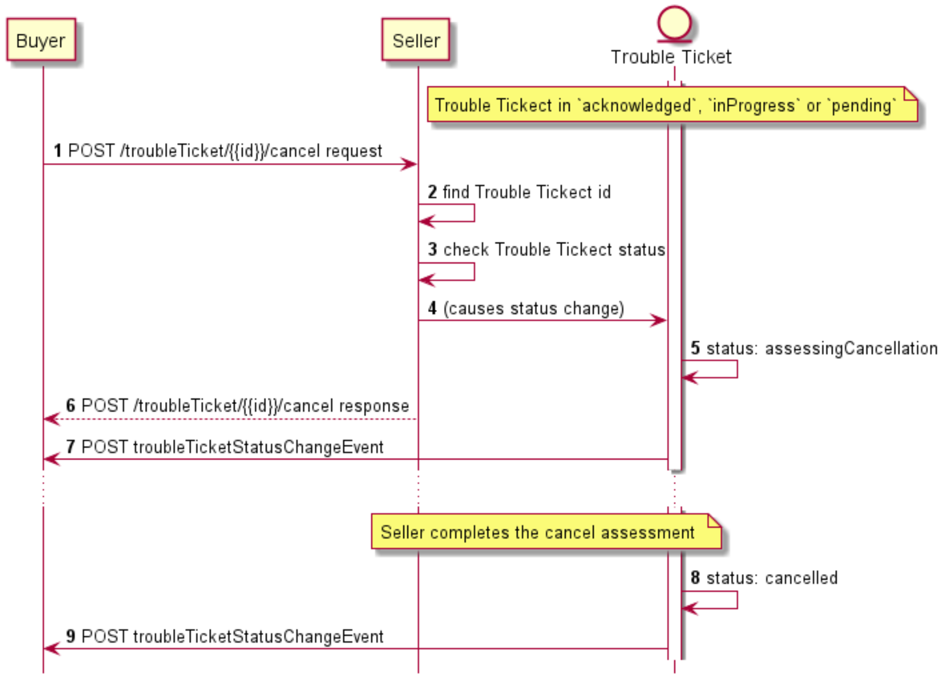

Buyer
Seller
Trouble Ticket
Trouble Tickect in `acknowledged`, `inProgress` or `pending`
1 POST /troubleTicket/{{id}}/cancel request
2 find Trouble Tickect id
3 check Trouble Tickect status
4 (causes status change)
5 status: assessingCancellation
6 POST /troubleTicket/{{id}}/cancel response
7 POST troubleTicketStatusChangeEvent
Seller completes the cancel assessment
8 status: cancelled
9 POST troubleTicketStatusChangeEvent

Figure 12: Cancel Trouble Ticket Flow

The Seller verifies the request, then searches for a Trouble Ticket to be cancelled by given `id`. If found, the status is verified (`acknowledged`, `inProgress` or `pending` allowed). If everything is verified correctly, the Seller moves the ticket to the `assessingCancellation` status, sends a successful response to a cancellation request followed by `troubleTicketStatusChangeEvent` and starts assessing the cancellation process for the ticket. After successful assessment, the ticket moves to `cancelled` status and another `troubleTicketStatusChangeEvent` is sent.

[R38] In case of a successful validation of the cancel request, the Seller **MUST** move the ticket to `assessingCancellation` status. [MEF113 R64]

[R39] In case `id` does not allow to find a `TroubleTicket` that is to be cancelled, an error response `Error404` **MUST** be returned. [MEF113 R62]

[R40] In case the `TroubleTicket` is in one of statuses: `resolved`, `closed`, `reopened`, `assessingCancellation`, or `cancelled` the Seller **MUST** return an error (`Error422`). [MEF113 R63]

6.6 Use Case 6: Ticket Resolution Confirmation

As shown in Figure 6, the Seller after resolving the Issue moves the Trouble Ticket to a `resolved` state. The Seller sends the `troubleTicketResolvedEvent` - a dedicated notification type. This is the point where the Buyer verifies the resolution and chooses to either close or reopen the Trouble Ticket. The Buyer uses one of the dedicated actions:

- `POST /troubleTicket/{{id}}/close`

- `POST /troubleTicket/{{id}}/reopen`

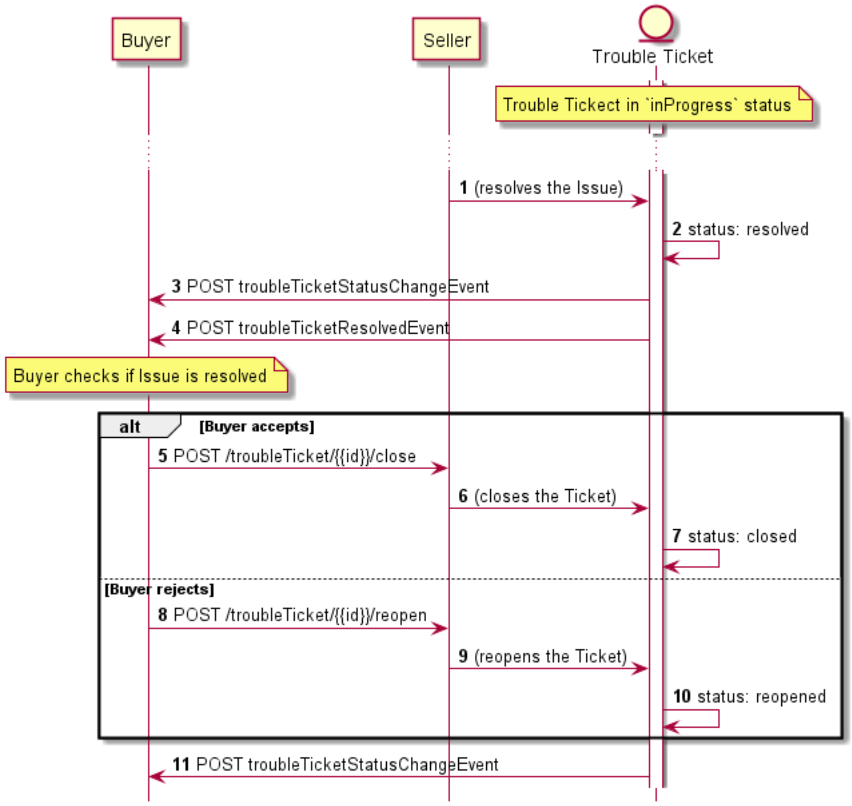

Buyer
Seller
Trouble Ticket
Trouble Tickect in `inProgress` status
1 (resolves the Issue)
2 status: resolved
3 POST troubleTicketStatusChangeEvent
4 POST troubleTicketResolvedEvent
Buyer checks if Issue is resolved
alt
[Buyer accepts]
5 POST /troubleTicket/{{id}}/close
6 (closes the Ticket)
7 status: closed
[Buyer rejects]
8 POST /troubleTicket/{{id}}/reopen
9 (reopens the Ticket)
10 status: reopened
11 POST troubleTicketStatusChangeEvent

Figure 13: Ticket Resolution Confirmation Flow

[R41] The Buyer **MUST** perform the `reopen` action if the Issue on which the Ticket was based has not been resolved in a satisfactory manner to the Buyer. [MEF113 R65]

[R42] The Buyer **MUST** perform the `close` action if the Issue on which the Ticket was based has been resolved in a satisfactory manner to the Buyer. [MEF113 R65]

[R43] If performing the `reopen` action, the Buyer **MUST** include a `reason` describing why the Buyer doesn't agree that the Trouble Ticket has been resolved in a satisfactory manner and is requesting the Trouble Ticket to be reopened. [MEF113 R66]

[R44] In case `id` does not allow to find a `TroubleTicket` that is to be reopened or closed, an error response `Error404` **MUST** be returned. [MEF113 R67]

[R45] If Buyer performs the `reopen` action, the Seller **MUST** change the Ticket `status` to `reopened`. [MEF113 R69]

[R46] If Buyer performs the `reopen` action, the Seller **MUST** add the `reason` (provided by the Buyer) to the `note` list of the Ticket with `note.source=buyer` and `note.author=closureRejection`. [MEF113 R68]

[R47] If Buyer performs the `close` action, the Seller **MUST** change the Ticket `status` to `closed`. [MEF113 R70]

Note: The Seller will return an error if the Buyer responds to the troubleTicketResolvedEvent after the Ticket was closed due to the expiration of the pre-agreed timeframe/timeout for the Buyer to confirm that the Issue on which the Ticket was based has been resolved satisfactorily.

6.7. Use Case 15: Retrieve Incident List

[O7] The Buyer **MAY** retrieve a list of Incidents by using a GET /incident operation with desired filtering criteria. The attributes that are available to be used are: [MEF113 O20]

- priority
- severity
- impact
- incidentType
- status
- relatedEntityId
- relatedEntityType
- creationDate.gt
- creationDate.lt
- situationStartDate.gt
- situationStartDate.lt
- expectedClosedDate.gt
- expectedClosedDate.lt
- closedDate.gt
- closedDate.lt

The example of making a request and using pagination is provided in section 6.2 Please refer to it as the rules also apply to this case.

[R48] The Seller **MUST** put the following attributes (if set) into the Incident_Find object in the response: [MEF113 R126]:

- id
- relatedEntity
- description
- priority
- severity
- impact
- incidentType
- status
- creationDate
- situationStartDate
- expectedClosedDate
- closedDate

[R49] In case no items matching the criteria are found, the Seller **MUST** return a valid response with an empty list. [MEF 113* R127]

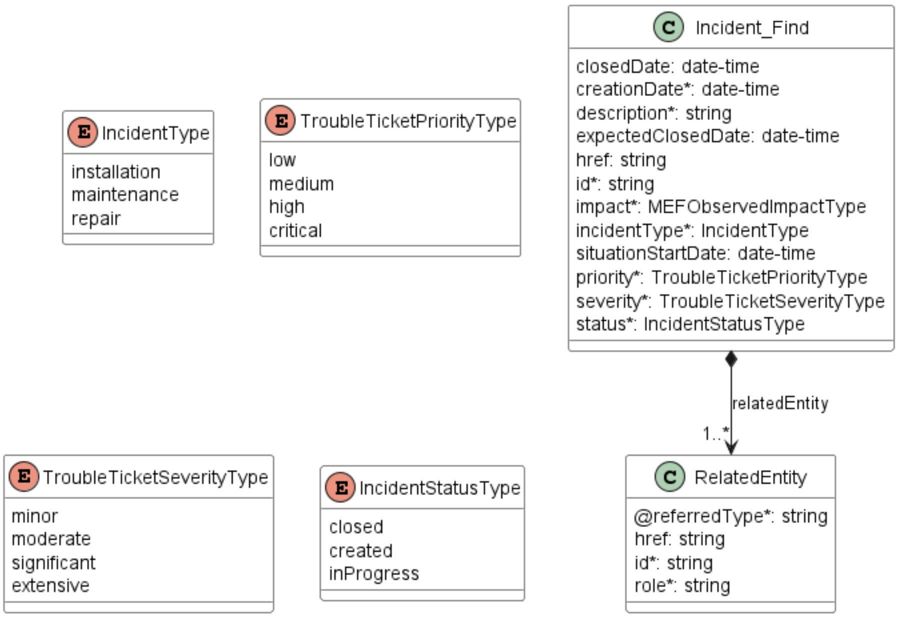


Figure 14: Use Case 15: Retrieve Incident List - Model

6.8. Use Case 16: Retrieve Incident by Incident Identifier

The Buyer can get detailed information about the Incident from the Seller by using a `GET /incident/{{id}}` operation.

[R50] In case `id` does not allow to find an `Incident` instance, an error response `Error404` **MUST** be returned. [MEF113 R129]

[R51] The Seller **MUST** put the following attributes into the `Incident` object in the response: [MEF113 R131]

- `id`
- `relatedEntity`
- `description`
- `priority`
- `severity`
- `impact`
- `incidentType`
- `status`
- `situationStartDate`
- `creationDate`
- `relatedContactInformation` - items with `role` equal to `incidentContact`

[R52] The Seller **MUST** provide all remaining optional attributes if they are set. [MEF113 R132]

[R53] The Seller's response to a Retrieve Incident by Incident Identifier request **MUST** include the `closedDate` if the `status` is `closed`. [MEF113 R133]

Table 10 presents the mapping between the API `status` names and the MEF 113 naming, together with their description.

status	MEF 113 name	Description
`created`	CREATED	A new Incident has been created and allocated a unique `id`.
`inProgress`	IN_PROGRESS	The Incident is in the process of being handled by the Seller.
`closed`	CLOSED	The Situation described in the Incident was closed by the Seller. This is a terminal state.

Table 10: Incident states

Figure 15 presents the Incident state machine:

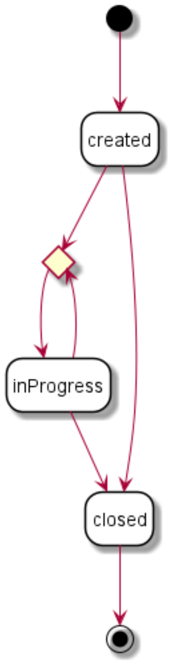


Figure 15: Incident State Machine

[R54] The Seller **MUST** support all Incident statuses and their associated transitions as described in Figure 15 and Table 10. [MEF113 R167]

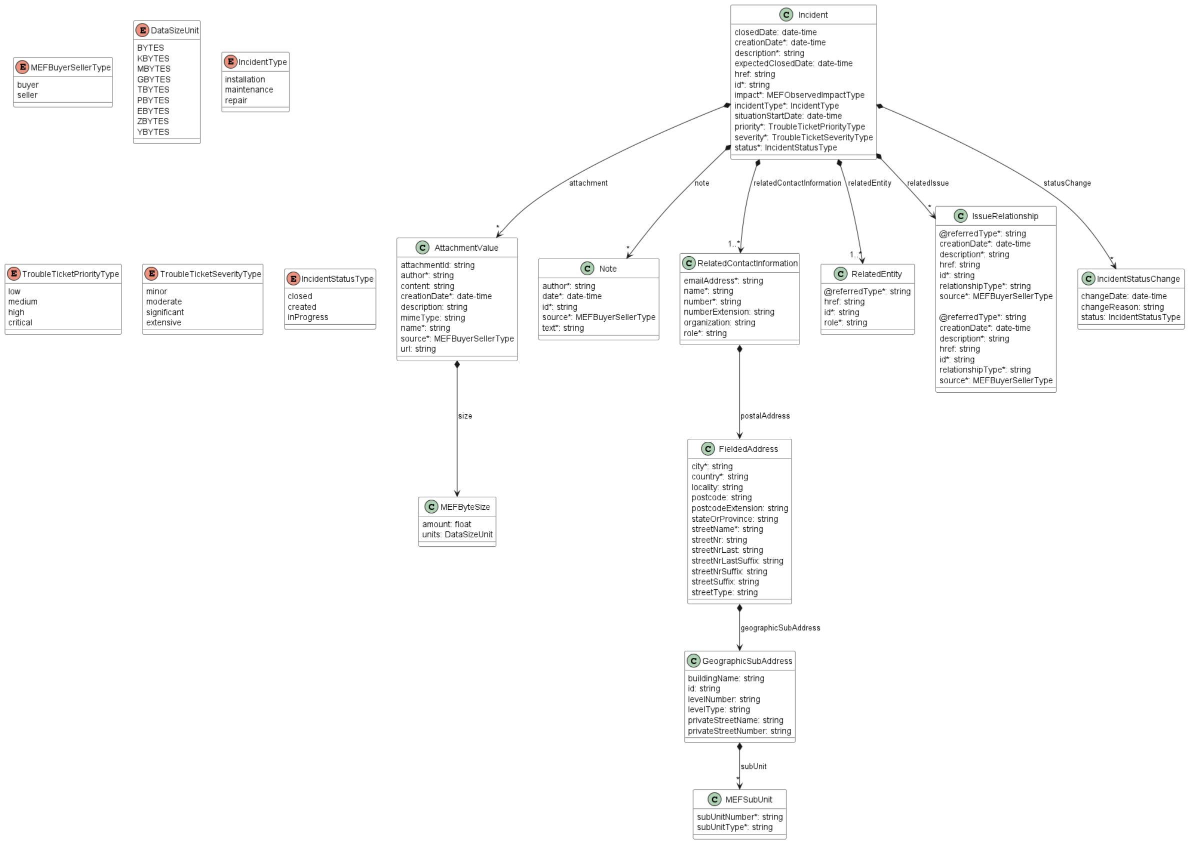


Figure 16: Use Case 16: Incident Model

```
{
  "id": "00001111-4321-6666-7777-000000003333",
  "href": "{{baseUrl}}/incident/00001111-4321-6666-7777-000000003333",
  "attachment": [
    {
      "attachmentId": "att-002",
      "author": "Kate Example",
      "creationDate": "2022-01-02T14:21:11.090Z",
      "description": "Print screen from the assurance system",
      "mimeType": "image/jpeg",
      "name": "Alarm",
      "url": "https://example.com/documents/00000000-5555-4444-3333-222211110000",
      "size": {
        "amount": 2.6,
        "units": "MBYTES"
      },
      "source": "seller"
    }
  ],
  "creationDate": "2022-01-12T23:09:44.814Z",
  "description": "Hardware failure",
  "expectedClosedDate": "2022-01-13T23:09:44.814Z",
  "impact": "down",
  "incidentType": "repair",
  "situationStartDate": "2022-01-12T23:09:44.814Z",
  "priority": "critical",
  "relatedContactInformation": [
    {
      "emailAddress": "Incident.Contact@example.com",
      "name": "Incident Contact",
      "number": "+98-765-432-10",
      "organization": "Seller Example Co.",
      "role": "incidentContact"
    }
  ],
  "relatedEntity": [
    {
      "id": "01494079-6c79-4a25-83f7-48284196d44d",
      "role": "Affected Product",
      "@referredType": "Product"
    }
  ],
  "relatedIssue": [
    {
      "@referredType": "TroubleTicket",
      "creationDate": "2022-01-12T23:09:44.815Z",
      "description": "Reported failure is causing referred Trouble Ticket",
      "id": "00000000-4444-5555-6666-000000000987",
      "relationshipType": "causes",
      "source": "seller"
    }
  ],
  "severity": "extensive",
  "status": "created"
}
```

6.9. Use case 17: Register for Event Notifications

[R55] The Seller **MUST** support Event Notifications. [MEF113 R134]

[R56] The Seller **MUST** support all of `TroubleTicketEventType`: [MEF113 R135]

- `troubleTicketAttributeValueChangeEvent`
- `troubleTicketInformationRequiredEvent`
- `troubleTicketResolvedEvent`
- `troubleTicketStatusChangeEvent`

[R57] The Buyer **MUST** support and register for all `TroubleTicketEventType`. [MEF113 R136]

To register for notifications the Buyer uses the `registerListener` operation from the API: `POST /hub`. The request model contains only 2 attributes:

- `callback` - mandatory, to provide the callback address the events will be notified to,
- `query` - optional, to provide the required types of event.

The usage of a combination of these attributes fulfills the [MEF113 R137], [MEF113 R138], [MEF113 R139] requirements.

By using a simple request:

```
{
  "callback": "https://buyer.com/listenerEndpoint"
}
```

The Buyer subscribes for notification of all types of events. Those are:

- `troubleTicketAttributeValueChangeEvent`
- `troubleTicketInformationRequiredEvent`
- `troubleTicketResolvedEvent`
- `troubleTicketStatusChangeEvent`
- `incidentCreateEvent`
- `incidentAttributeValueChangeEvent`
- `incidentStatusChangeEvent`

If the Buyer wishes to receive only notification of a certain type, a `query` must be added:

```
{
  "callback": "https://buyer.com/listenerEndpoint",
  "query": "eventType=troubleTicketResolvedEvent"
}
```

If the Buyer wishes to subscribe to 2 different types of events, there are 2 possible syntax variants [TMF630]:

```
eventType=troubleTicketResolvedEvent,troubleTicketStatusChangeEvent
```

or

```
eventType=troubleTicketResolvedEvent&eventType=troubleTicketStatusChangeEvent
```

The `query` formatting complies to RCF3986 RFC3986. According to it, every attribute defined in the Event model (from notification API) can be used in the `query`. However, this standard requires only `eventType` attribute to be supported.

[R58] `eventType` is the only attribute that the Seller **MUST** support in the query.

The Seller responds to the subscription request by adding the `id` of the subscription to the message that must be further used for unsubscribing.

```
{
  "id": "00000000-0000-0000-0000-000000000678",
  "callback": "https://buyer.com/listenerEndpoint",
  "query": "eventType=troubleTicketResolvedEvent"
}
```

Example of a final address that the Notifications will be sent to (for Sonata, `troubleTicketResolvedEvent`):

- `https://buyer.com/listenerEndpoint/mefApi/sonata/troubleTicketNotification/v2/listener/troubleTicketResolvedEvent`

6.10. Use case 18: Send Event Notification

Notifications are used to asynchronously inform the Buyer about the respective objects and attributes changes. The Seller's synchronous response to a Trouble Ticket create requests are considered to act as a Create Notification so there is no explicit respective Create Notification type. The next notification must be sent when the state changes compared to the previously sent one.

[R59] The Seller **MUST** send Notifications of `eventTypes` to Buyers who have registered for them. [MEF113 R141]

[R60] The Seller **MUST NOT** send Notifications for `eventTypes` to Buyers who have not registered for them. [MEF113 R140]

The Figure below shows all entities involved in the Notification use cases.

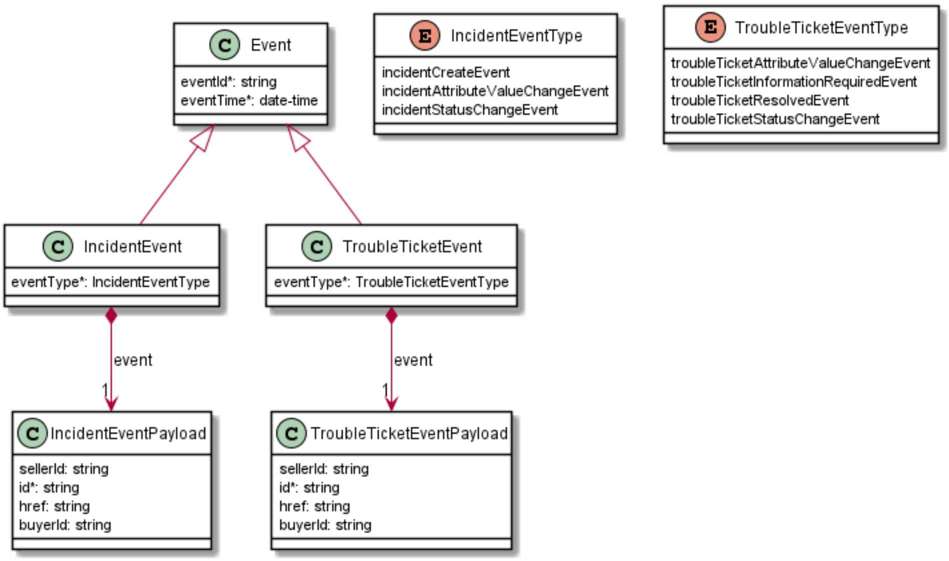


Figure 17: Use Case 18. Notification Data Model

The following snippet presents an example of `troubleTicketResolvedEvent`

```
{
  "eventId": "event-001",
  "eventType": "troubleTicketResolvedEvent",
  "eventTime": "2021-06-03T15:56:08.559Z",
  "event": {
    "id": "00000000-4444-5555-6666-000000000987"
  }
}
```

Note: the body of the event carries only the source object's `id`. The Buyer needs to query it later by `id` to get details.

To stop receiving events, the Buyer has to use the `unregisterListener` operation from the `DELETE /hub/{id}` endpoint. The `id` is the identifier received from the Seller during the listener registration.

The table below presents the mapping between the API Notification types' names and the ones in MEF 113 together with event descriptions. The inconsistencies are caused by API naming convention and using the TMF event types as the base for this API.

API name	MEF 113 name	Description

API name	MEF 113 name	Description
troubleTicketAttributeValueChangeEvent	TICKET_UPDATE	The Seller settable attributes for a Ticket were updated by the Seller. Note: Buyer initiated Ticket updates due to Patch operation will not trigger a troubleTicketAttributeValueChangeEvent
troubleTicketInformationRequiredEvent	TICKET_STATE_CHANGE	A Ticket status was changed by the Seller.
troubleTicketResolvedEvent	TICKET_INFO_REQUIRED	The Seller requires more information from the Buyer for a Ticket to continue processing a Ticket. The details on what information is needed from the Buyer will be provided via a Ticket note. The Ticket status is pending. Note: The Buyer uses the Patch operation to provide more information for a Ticket.
troubleTicketStatusChangeEvent	TICKET_RESOLVED	The Seller is informing the Buyer the Ticket is resolved and the Buyer to verify that the Issue on which the Ticket was based is no longer observed. The Ticket status is resolved. Note: The Buyer confirms if the Issue has been resolved satisfactorily or not using close or reopen operations
incidentCreateEvent	INCIDENT_CREATE	A new Incident was created by the Seller.
incidentAttributeValueChangeEvent	INCIDENT_UPDATE	An open Incident was updated by the Seller.
incidentStatusChangeEvent	INCIDENT_STATE_CHANGE	An Incident status was changed by the Seller.

Table 11. Notification types mapping

[R61] The Seller **MUST** send a `troubleTicketAttributeValueChangeEvent` whenever the Seller updates any of the following Ticket attributes: [MEF113 R156]

- `sellerSeverity`
- `sellerPriority`
- `expectedResolutionDate`
- `note`
- `attachment`
- `relatedContactInformation`
- `relatedIssue`
- `workOrder` - including updates to a Referenced WorkOrder

[R62] The Seller **MUST** send a `troubleTicketStatusChangeEvent` whenever a Ticket `status` change occurs. [MEF113 R157]

[R63] Whenever the Ticket `status` is changed to `pending`, the Seller **MUST** add a `note` to the Ticket to inform the Buyer about what additional information is required for the Ticket or for the Buyer to schedule an Appointment to continue processing the Ticket. [MEF113 R159]

[R64] The Seller **MUST** send a `troubleTicketInformationRequiredEvent` whenever the Ticket `status` has been changed to `pending` and the `appointmentRequired` attribute for all WorkOrders linked to the Ticket are `false`. [MEF113 R160]

[R65] If the `appointmentRequired` attribute for a Workorder is `true`, the Seller **MUST** set the `status` of the Ticket associated to the Workorder to `pending`. [MEF113 R158]

[R66] The Seller **MUST** send an `troubleTicketResolvedEvent` whenever the Ticket `status` is changed to `resolved`. [MEF113 R161]

[R67] The Seller **MUST** send an `incidentCreateEvent` whenever a new Incident has been created. [MEF113 R168]

[R68] The Seller **MUST** send a `incidentAttributeValueChangeEvent` whenever the Seller updates any of the Incident attributes (excluding `status`) [MEF113 R169]

[R69] The Seller **MUST** send a `incidentStatusChangeEvent` whenever an Incident `status` change occurs. [MEF113 R170]

[R70] When the Incident `status` moves to `inProgress`, the Seller **MUST** set the `expectedClosedDate`. [MEF113 R171]

[R71] The Seller **MUST NOT** send an `IncidentEvent` to a Buyer for an Incident impacting a Product that the Seller has not activated on behalf of the Buyer. [MEF113 R172]

7. API Details

7.1. API patterns

7.1.1. Indicating errors

Erroneous situations are indicated by appropriate HTTP responses. An error response is indicated by HTTP status 4xx (for client errors) or 5xx (for server errors) and the appropriate response payload. The Product Order API uses the error responses as depicted and described below.

Implementations can use HTTP error codes not specified in this standard in compliance with rules defined in RFC 7231 [RFC7231]. In such a case, the error message body structure might be aligned with the `Error`.

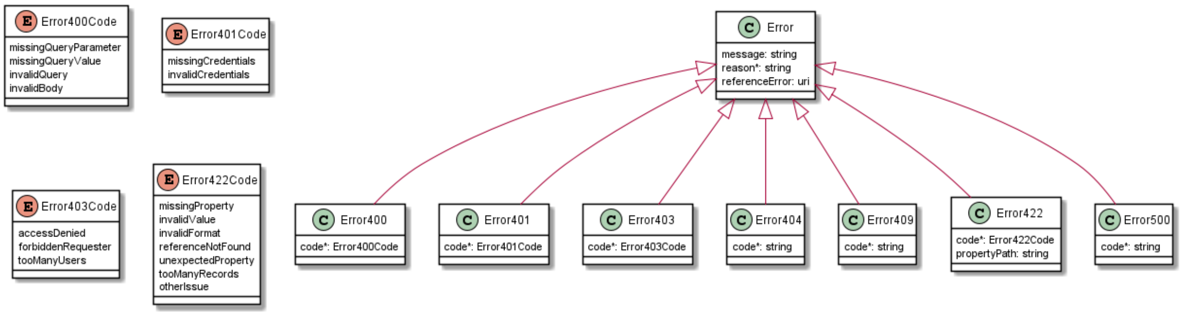


Figure 18. Data model types to represent an erroneous response

7.1.1.1. Type Error

Description: Standard Class used to describe API response error Not intended to be used directly. The `code` in the HTTP header is used as a discriminator for the type of error returned in runtime.

Name	Type	Description
message	string	Text that provides mode details and corrective actions related to the error. This can be shown to a client user.
reason*	string	Text that explains the reason for the error. This can be shown to a client user.
referenceError	uri	URL pointing to documentation describing the error

7.1.1.2. Type Error400

Description: Bad Request. (https://tools.ietf.org/html/rfc7231#section-6.5.1)

Inherits from:

- Error

Name	Type	Description
code*	Error400Code	One of the following error codes: - missingQueryParameter: The URI is missing a required query-string parameter - missingQueryValue: The URI is missing a required query-string parameter value - invalidQuery: The query section of the URI is invalid. - invalidBody: The request has an invalid body

7.1.1.3. `enum` Error400Code

Description: One of the following error codes:

- missingQueryParameter: The URI is missing a required query-string parameter
- missingQueryValue: The URI is missing a required query-string parameter value
- invalidQuery: The query section of the URI is invalid.
- invalidBody: The request has an invalid body

7.1.1.4. Type Error401

Description: Unauthorized. (https://tools.ietf.org/html/rfc7235#section-3.1)

Inherits from:

- Error

Name	Type	Description
code*	Error401Code	One of the following error codes: - missingCredentials: No credentials provided. - invalidCredentials: Provided credentials are invalid or expired

7.1.1.5. `enum` Error401Code

Description: One of the following error codes:

- missingCredentials: No credentials provided.
- invalidCredentials: Provided credentials are invalid or expired

7.1.1.6. Type Error403

Description: Forbidden. This code indicates that the server understood the request but refuses to authorize it. (https://tools.ietf.org/html/rfc7231#section-6.5.3)

Inherits from:

- Error

Name	Type	Description
code*	Error403Code	This code indicates that the server understood the request but refuses to authorize it because of one of the following error codes: - accessDenied: Access denied - forbiddenRequester: Forbidden requester - tooManyUsers: Too many users

7.1.1.7. enum Error403Code

Description: This code indicates that the server understood the request but refuses to authorize it because of one of the following error codes:

- accessDenied: Access denied
- forbiddenRequester: Forbidden requester
- tooManyUsers: Too many users

7.1.1.8. Type Error404

Description: Resource for the requested path not found. (https://tools.ietf.org/html/rfc7231#section-6.5.4)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - notFound: A current representation for the target resource not found

7.1.1.9. Type Error409

Description: Conflict (https://datatracker.ietf.org/doc/html/rfc7231#section-6.5.8)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - conflict: The client has provided a value whose semantics are not appropriate for the property.

7.1.1.10. Type Error422

The response for HTTP status `422` is a list of elements that are structured using the `Error422` data type. Each list item describes a business validation problem. This type introduces the `propertyPath` attribute which points to the erroneous property of the request, so that the Buyer may fix it easier. It is highly recommended that this property should be used, yet remains optional because it might be hard to implement.

Description: Unprocessable entity due to a business validation problem. (https://tools.ietf.org/html/rfc4918#section-11.2)

Inherits from:

- Error

Name	Type	Description
code*	Error422Code	One of the following error codes: - missingProperty: The property the Seller has expected is not present in the payload - invalidValue: The property has an incorrect value - invalidFormat: The property value does not comply with the expected value format - referenceNotFound: The object referenced by the property cannot be identified in the Seller system - unexpectedProperty: Additional property, not expected by the Seller has been provided - tooManyRecords: the number of records to be provided in the response exceeds the Seller's threshold. - otherIssue: Other problem was identified (detailed information provided in a reason)
propertyPath	string	A pointer to a particular property of the payload that caused the validation issue. It is highly recommended that this property should be used. Defined using JavaScript Object Notation (JSON) Pointer (https://tools.ietf.org/html/rfc6901).

7.1.1.11. `enum` Error422Code

Description: One of the following error codes:

- missingProperty: The property the Seller has expected is not present in the payload
- invalidValue: The property has an incorrect value
- invalidFormat: The property value does not comply with the expected value format
- referenceNotFound: The object referenced by the property cannot be identified in the Seller system
- unexpectedProperty: Additional property, not expected by the Seller has been provided
- tooManyRecords: the number of records to be provided in the response exceeds the Seller's threshold.
- otherIssue: Other problem was identified (detailed information provided in a reason)

7.1.1.12. Type Error500

Description: Internal Server Error. (https://tools.ietf.org/html/rfc7231#section-6.6.1)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - internalError: Internal server error - the server encountered an unexpected condition that prevented it from fulfilling the request.

7.1.1.13. Type Error501

Description: Not Implemented. Used in case Seller is not supporting an optional operation (https://tools.ietf.org/html/rfc7231#section-6.6.2)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - notImplemented: Method not supported by the server

7.1.2. Response pagination

A response to retrieve a list of results (e.g. `GET /productOfferingQualification`) can be paginated. The Buyer can specify following query attributes related to pagination:

- `limit` - number of expected list items
- `offset` - offset of the first element in the result list

The Seller returns a list of elements that comply with the requested `limit`. If the requested `limit` is higher than the supported list size the smaller list result is returned. In that case, the size of the result is returned in the header attribute `X-Result-Count`. The Seller can indicate that there are additional results available using:

- `X-Total-Count` header attribute with the total number of available results
- `X-Pagination-Throttled` header set to `true`

[R72] Seller **MUST** use either `X-Total-Count` or `X-Pagination-Throttled` to indicate that the page was truncated and additional results are available.

7.2. Management API Data model

Figure 19 presents the whole Trouble Ticket Management data model the data types, requirements related to them, and mapping to MEF 113 specifications are discussed later in this section.

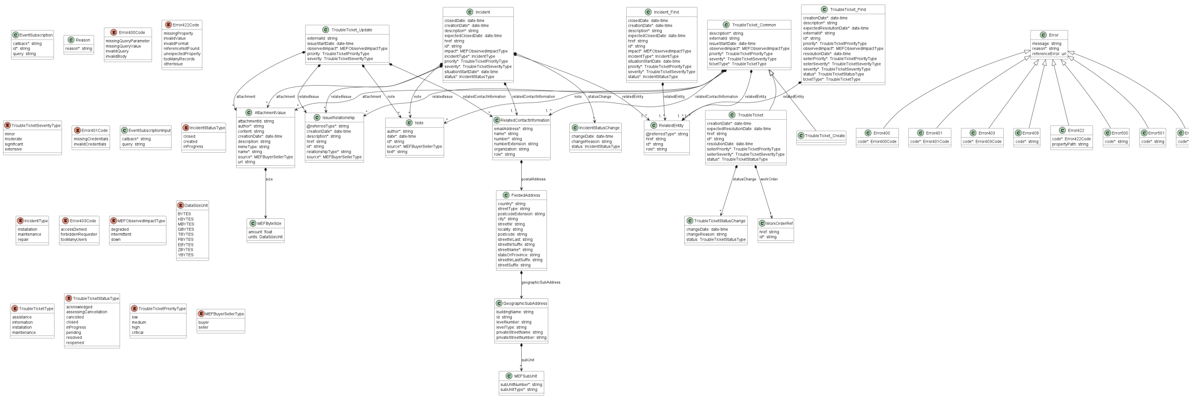


Figure 19: Trouble Ticket Management Data Model

7.2.1. TroubleTicket

7.2.1.1. Type TroubleTicket_Common

Description: A Trouble Ticket is a record of an issue that is created, tracked, and managed by a Trouble Ticket management system Skipped properties: id,href

Name	Type	Description	MEF 11.
attachment	AttachmentValue[]	Attachments to the Ticket, such as a file, screen shot or embedded content. Attachments may be added but may not be modified or deleted (for historical reasons).	Attachm

Name	Type	Description	MEF 11.
description*	string	Summarized description of the Issue the Buyer is experiencing.	Descript
externalId	string	Identifier provided by the Buyer to allow the Buyer to use as a search attribute in Retrieve Ticket List.	Buyer Ticket Identifie
issueStartDate	date-time	The date indicating when the Buyer first observed the Issue, to provide the Seller with additional insight.	Issue Sta Date
note	Note[]	A set of comments or information associated to the Ticket. This list can be empty. Notes may be added but may not be modified or deleted (for historical reasons).	Notes
observedImpact*	MEFObservedImpactType	The type of impact observed by the Buyer.	Observe Impact
priority*	TroubleTicketPriorityType	The priority of the Trouble Ticket and how quickly the issue should be resolved. Example: Critical, High, Medium, Low. The value is set by the ticket management system considering the severity, ticket type etc...	Priority

Name	Type	Description	MEF 11
relatedContactInformation*	RelatedContactInformation[]	Party playing a role for this Trouble Ticket. The 'role' is to specify the type of contact as specified in MEF 113: Reporter Contact ('role=reporterContact') - REQUIRED in the request Buyer Technical Contacts ('role=buyerTechnicalContact') Seller Ticket Contact ('role=sellerTicketContact') Seller Technical Contact ('role=sellerTechnicalContact')	Reporter Contact, Buyer Technica Contacts Seller Ticket Contact, Seller Technica Contacts
relatedEntity*	RelatedEntity[]	An entity that is related to the ticket such as a bill, a product, etc. The entity against which the ticket is associated.	Product Identifie
relatedIssue	IssueRelationship[]	A list of Related Issue relationships. Represents relationships to other Trouble Tickets and Incidents.	Related Tickets A Incidents
severity*	TroubleTicketSeverityType	The severity or impact (ITIL) of the Issue as evaluated by the Buyer.	Severity
ticketType*	TroubleTicketType	The presumed cause of the Trouble Ticket as evaluated by the Buyer.	Type

7.2.1.2. Type TroubleTicket_Create

Description: A Trouble Ticket is a record of an issue that is created, tracked, and managed by a Trouble Ticket management system The modeling pattern introduces the `Common` supertype to aggregate attributes that are common to both `TroubleTicket` and `TroubleTicket_Create`. It this case the Create type has a subset of attributes of the response type and does not introduce any new, thus the `Create` type has an empty definition.

Inherits from:

- TroubleTicket_Common

7.2.1.3. Type TroubleTicket

Description: A Trouble Ticket is a record of an issue that is created, tracked, and managed by a Trouble Ticket management system

Inherits from:

- TroubleTicket_Common

Name	Type	Description	MEF 113
creationDate*	date-time	The date on which the Trouble Ticket was created	Ticket Creation Date
expectedResolutionDate	date-time	The date provided by the Seller to indicate when the Ticket is expected to be resolved	Target Resolved Date
href	string	Hyperlink, a reference to the Trouble Ticket entity	Not represented in MEF 113
id*	string	Unique (within the Seller Ticket domain) identifier for the Ticket.	Ticket Identifier
resolutionDate	date-time	The date the Ticket status was set to resolved by the Seller	Resolved Date
sellerPriority*	TroubleTicketPriorityType	The priority (ITIL) is based on the assessment of the impact and urgency of how quickly the Ticket should be resolved after evaluation by the Seller of the impact of the Issue on the Buyer.	Seller Priority

Name	Type	Description	MEF 113
sellerSeverity*	TroubleTicketSeverityType	The severity or impact (ITIL) of the Issue on the Buyer as evaluated by the Seller.	Seller Severity
status*	TroubleTicketStatusType	The current status of the Trouble Ticket	Ticket State
statusChange	TroubleTicketStatusChange[]	The status change history that is associated to the ticket. Populated by the Seller.	Not represented in MEF 113
workOrder	WorkOrderRef[]	A reference to a set of WorkOrders to be performed under the responsibility of Seller technician(s) to resolve the Ticket.	Workorders

7.2.1.4. Type TroubleTicket_Find

Description: This class represents a single list item for the response of `listTroubleTicket` operation.

Name	Type	Description	MEF 113
creationDate*	date-time	The date on which the Trouble Ticket was created	Ticket Creation Date
description*	string	Summarized description of the Issue the Buyer is experiencing.	Description

Name	Type	Description	MEF 113
expectedResolutionDate*	date-time	The date provided by the Seller to indicate when the Ticket is expected to be resolved	Target Resolved Date
externalId*	string	Additional identifier coming from an external system	Buyer Ticket Identifier
id*	string	Unique identifier of the Trouble Ticket	Ticket Identifier
priority*	TroubleTicketPriorityType	The priority (ITIL) is based on the assessment of the impact and urgency of how quickly the Ticket should be resolved as evaluated by the Buy-er. The Priority is used by the Seller to determine the order in which Tickets get resolved across Buyers.	Priority
relatedEntity*	RelatedEntity[]	An entity that is related to the ticket such as a bill, a product, etc. The entity against which the ticket is associated.	Product Identifier
observedImpact*	MEFObservedImpactType	The type of impact observed by the Buyer.	

Name	Type	Description	MEF 113
resolutionDate*	date-time	The date the Ticket status was set to resolved by the Seller	Resolved Date
sellerPriority*	TroubleTicketPriorityType	The priority (ITIL) is based on the assessment of the impact and urgency of how quickly the Ticket should be resolved after evaluation by the Seller of the impact of the Issue on the Buyer.	Seller Priority
sellerSeverity*	TroubleTicketSeverityType	The severity or impact (ITIL) of the Ticket on the Buyer as evaluated by the Seller.	Seller Severity
severity*	TroubleTicketSeverityType	The severity or impact (ITIL) of the Ticket as evaluated by the Buyer.	Severity
status*	TroubleTicketStatusType	The current status of the Trouble Ticket	Not represented in MEF 113
ticketType*	TroubleTicketType	The presumed cause of the Trouble Ticket as evaluated by the Buyer.	Type

7.2.1.5. Type TroubleTicket_Update

Description: A Trouble Ticket is a record of an issue that is created, tracked, and managed by a Trouble Ticket management system

Name	Type	Description	MEF 1
attachment	AttachmentValue[]	Attachments to the Ticket, such as a file, screen shot or embedded content.	Attachn
externalId	string	Additional identifier coming from an external system	Buyer T Identifi
issueStartDate	date-time	The date indicating when the Buyer first observed the Issue, to provide the Seller with additional insight.	issueSta
note	Note[]	A set of comments or information associated to the Ticket. This list can be empty. Notes may be added but may not be modified or deleted (for historical reasons).	Notes
priority	TroubleTicketPriorityType	The priority of the Trouble Ticket and how quickly the issue should be resolved. Example: Critical, High, Medium, Low. The value is set by the ticket management system considering the severity, ticket type etc...	Priority
relatedContactInformation	RelatedContactInformation[]	Party playing a role for this quote. If `instantSyncQuote=false` then the Buyer MUST specify Buyer Contact Information ('role=buyerContactInformation') and the Seller MUST specify Seller Contact Information ('role=sellerContactInformation')	Reporte Contact Buyer Technic Contact Seller T Contact Seller Technic Contact
relatedIssue	IssueRelationship[]	A list of Related Issue relationships. Represents relationships to other Trouble Tickets and Incidents.	Related Tickets Inciden

Name	Type	Description	MEF 1
severity	TroubleTicketSeverityType	The severity of the issue. Indicate the implication of the issue on the expected functionality e.g. of a system, application, service etc...	Not represe MEF 11

7.2.1.6. enum TroubleTicketPriorityType

Description: Possible values for the priority of the Trouble Ticket

Value
low
medium
high
critical

7.2.1.7. Type IssueRelationship

Description: Represents relationships to other Trouble Tickets and Incidents

Name	Type	Description	MEF 113
@referredType*	string	The type of the referred Issue (Incident or TroubleTicket)	Related Ticket Or Incident Type
creationDate*	date-time	The date the relationship was created	Relation Creation Date
description*	string	A description of the reason for the Relation Source to set the relationship	Relation Reason Description
href	string	Reference of the Trouble Ticket or Incident	Not represented in MEF 113
id*	string	Unique identifier of the referenced Issue (Trouble Ticket od Incident)	Related Ticket Or Incident Identifier

Name	Type	Description	MEF 113
relationshipType*	string	Type of the Trouble Ticket relationship can be blocks, depends on, duplicates, causes, etc...	Relation Type
source*	MEFBuyerSellerType	Indicates if this Related Issue was added by the Buyer or the Seller.	Relation Source

7.2.1.8. enum TroubleTicketSeverityType

Description: Possible values for the severity of the Trouble Ticket

Value
minor
moderate
significant
extensive

7.2.1.9. enum MEFObservedImpactType

Description: An enumeration of the possible values of impact observed by the Buyer.

- degraded: When the Product is impacted and not meeting the Product specifications.
- intermittent: When the Product is not operational as intended on an intermittent basis.
- down: When the Product is non-operational.

Value
degraded
intermittent
down

7.2.1.10. Type TroubleTicketStatusChange

Description: Holds the status notification reasons and associated date the status changed, populated by the server

Name	Type	Description	MEF 113
changeDate	date-time	The date and time the status changed.	Not represented in MEF 113

Name	Type	Description	MEF 113
changeReason	string	The reason why the status changed.	Not represented in MEF 113
status	TroubleTicketStatusType	Reached status	Not represented in MEF 113

7.2.1.11. `enum` TroubleTicketStatusType

Description: Possible values for the status of the Trouble Ticket

status	MEF 113 name	Description
`acknowledged`	ACKNOWLEDGED	A request to create a Ticket was received and accepted by the Seller. The Ticket create request has been validated and a Ticket has been created by the Seller and allocated a unique `id`.
`assessingCancellation`	ASSESSING_CANCELLATION	A request has been made by the Buyer to cancel the Ticket and is being assessed by the Seller to determine whether to just close the Ticket, or continue to resolve the Issue to prevent similar Create Ticket requests from other Buyers. If the Seller chooses to resolve the Issue, the Seller might create an Incident or an internal Ticket for the Issue, but that is outside the scope of this document. After the Seller has completed the assessment, the Seller updates the Ticket State to `cancelled`.
`cancelled`	CANCELLED	The Ticket has been successfully cancelled by the Buy-er. The Buyer will receive no further Event Notifications for the Ticket. This is a terminal state.

status	MEF 113 name	Description
`closed`	CLOSED	The Buyer has confirmed that the Issue they reported is no longer observed, or the pre-defined time frame (agreed upon between Buyer and Seller) for confirming that the Issue has been resolved has passed without a response by the Buyer. This is a terminal state.
`inProgress`	IN_PROGRESS	The Ticket is in the process of being handled and investigated for resolution by the Seller.
`pending`	PENDING	The Seller is waiting on the Buyer to provide additional information for the Ticket, or the Buyer to schedule an Appointment for the WorkOrder (linked to the Ticket) in order to continue processing the Ticket. This may result in the clock being stopped for the service level agreement until the Buyer has responded to the request.
`reopened`	REOPENED	The Buyer has verified that the Issue described in the Ticket is still observed and has not been resolved satisfactorily. The Buyer rejects the Seller's request to close the Ticket. The Ticket has been reopened and is waiting for further actions from the Seller.
`resolved`	RESOLVED	The Buyer's Issue described in the Ticket was resolved by the Seller. The Seller assumes that normal operation is re-established for the Buyer's product and i snow waiting for the Buyer to confirm that the Issue they reported is no longer observed.

7.2.1.12. `enum` TroubleTicketType

Description: Possible values for the type of the Trouble Ticket:

- assistance: Requesting help for a situation (not a failure) requiring attention that is not categorized.
- information: Buyer is requesting information on the Issue
- installation: Related to installation issue. Provisioning is complete, but Product is not operational.
- maintenance: Any scheduled or non-scheduled maintenance related Issue.

Value	MEF 113
assistance	ASSISTANCE
information	INFORMATION
installation	INSTALLATION
maintenance	MAINTENANCE

7.2.1.13. Type Reason

Description: An object to convey a reason for the operation.

Name	Type	Description	MEF 113
reason*	string	A text description of why given operation was requested.	Closure Rejection Reason

7.2.1.14. Type WorkOrderRef

Description: A reference to an WorkOrder resource.

Name	Type	Description	MEF 113
href	string	Hyperlink to the referenced WorkOrder.	Not represented in MEF 113
id*	string	Identifier of the referenced WorkOrder.	Workorder Identifier

7.2.2. Incident

7.2.2.1. Type Incident

Description: An Incident is a record of an issue that is not part of normal operation in the Seller's network that has a possible negative impact on the operability of the network on one or more Buyers.

Name	Type	Description	MEF

Name	Type	Description	MEF
attachment	AttachmentValue[]	Attachments to the Ticket, such as a file, screenshot, or embedded content. Attachments may be added but may not be modified or deleted (for historical reasons).	Attach
closedDate	date-time	The date the Incident status was set to closed by the Seller	Incide Close
creationDate*	date-time	The date on which the Incident was created	Incide Creati Date
description*	string	Description of the Incident	Descri
expectedClosedDate	date-time	The date provided by the Seller to indicate when the Incident is expected to be closed.	Incide Expec Close
href	string	Hyperlink, a reference to the Incident entity	Not repres in ME
id*	string	Unique (within the Seller domain) identifier for the Incident.	Incide Identi
impact*	MEFObservedImpactType	The type of impact observed by the Buyer.	Incide Impac
incidentType*	IncidentType	The presumed cause of the Incident as evaluated by the Seller.	Incide Type
note	Note[]	A set of unstructured comments or information associated to the Incident. Notes may be added but may not be modified or deleted (for historical reasons).	Incide Notes
priority*	TroubleTicketPriorityType	The priority (ITIL) is based on the assessment of the impact and urgency of how quickly the Incident should be resolved after evaluation by the Seller of the impact of the Incident.	Incide Priorit

Name	Type	Description	MEF
relatedContactInformation*	RelatedContactInformation[]	Party playing a role in this Incident. The 'role' is to specify the type of contact as specified in MEF 113: Incident Contact ('role=incidentContact') - REQUIRED to be set by the Seller Incident Technical Contact ('role=incidentTechnicalContact')	Incide Conta Incide Techn Conta
relatedEntity*	RelatedEntity[]	A set of identifiers of the Products on which the Incident could have an impact on the normal operation.	Produ Identi
relatedIssue	IssueRelationship[]	A list of Related Issue relationships. Represents relationships to other Trouble Tickets and Incidents.	Incide Relate Ticket Incide
severity*	TroubleTicketSeverityType	The severity or impact (ITIL) of the Incident as evaluated by the Seller.	Incide Severi
situationStartDate*	date-time	The date when the situation was first identified, for example via error logs.	
status*	IncidentStatusType	The current status of the Incident	Incide State
statusChange	IncidentStatusChange[]	The status change history that is associated to the Incident. Populated by the Seller.	Not repres in ME

7.2.2.2. Type Incident_Find

Description: This class represents a single list item for the response of `listIncident` operation.

Name	Type	Description	MEF 113
closedDate	date-time	The date the Incident status was set to closed by the Seller	Incident Closed Date
creationDate*	date-time	The date on which the Incident was created	Incident Creation Date

Name	Type	Description	MEF 113
description*	string	Description of the Incident	Description
expectedClosedDate	date-time	The date provided by the Seller to indicate when the Incident is expected to be closed.	Incident Expected Closed Date
href	string	Hyperlink, a reference to the Incident entity	Not represented in MEF 113
id*	string	Unique (within the Seller domain) identifier for the Incident.	Incident Identifier
impact*	MEFObservedImpactType	The type of impact observed by the Buyer.	Incident Impact
incidentType*	IncidentType	The presumed cause of the Incident as evaluated by the Seller.	Incident Type
situationStartDate	date-time	The date when the Incident was first identified, for example via error logs.	Situation Start Date
priority*	TroubleTicketPriorityType	The priority (ITIL) is based on the assessment of the impact and urgency of how quickly the Incident should be resolved after evaluation by the Seller of the impact of the Incident.	Incident Priority
relatedEntity*	RelatedEntity[]	An entity that is related to the Incident such as a service, a product, etc. The entity which the Incident is associated with.	Product Identifier

Name	Type	Description	MEF 113
severity*	TroubleTicketSeverityType	The severity or impact (ITIL) of the Incident as evaluated by the Seller.	Incident Severity
status*	IncidentStatusType	The current status of the Incident	Incident State

7.2.2.3. `enum` IncidentType

Description: Possible values for the type of the Incident:

- maintenance: Any scheduled or non-scheduled maintenance related Incident.
- repair: Any non-scheduled Situation requiring repair by the Seller.
- installation: Any installation related Situation requiring action by the Seller.

Value	MEF 113
maintenance	MAINTENANCE
repair	REPAIR
installation	INSTALLATION

7.2.2.4. `enum` IncidentStatusType

Description: Possible values for the status of the Incident

status	MEF 113 name	Description
`closed`	CLOSED	The Situation described in the Incident was closed by the Seller. This is a terminal state.
`created`	CREATED	A new Incident has been created and allocated a unique `id`.
`inProgress`	IN_PROGRESS	The Incident is in the process of being handled by the Seller.

7.2.2.5. Type IncidentStatusChange

Description: Holds the status notification reasons and associated date the status changed, populated by the server

Name	Type	Description	MEF 113
changeDate	date-time	The date and time the status changed.	Not represented in MEF 113

Name	Type	Description	MEF 113
changeReason	string	The reason why the status changed.	Not represented in MEF 113
status	IncidentStatusType	Reached status	Not represented in MEF 113

7.2.3. Common

Types described in this subsection are shared among two or more Cantata and Sonata APIs.

7.2.3.1. Type AttachmentValue

Description: Complements the description of an element (for instance a product) through video, pictures...

Name	Type	Description	MEF 113
attachmentId	string	locally unique identifier to distinguish items from the Attachment list.	Not represented in MEF 113
author*	string	The name of the person or organization who added the Attachment.	Attachment Author
content	string	The actual contents of the attachment object, if embedded, encoded as base64. Either url or (content and mimeType) attributes MUST be provided during creation.	Content
creationDate*	date-time	The date the Attachment was added.	Attachment Date
description	string	A narrative text describing the content of the attachment	Description
mimeType	string	Attachment mime type such as extension file for video, picture and document	Mime Type
name*	string	The name of the attachment	Attachment Name
size	MEFByteSize	The size of the attachment.	Size

Name	Type	Description	MEF 113
source*	MEFBuyerSellerType	Indicates if the attachment was added by the Buyer or the Seller.	Attachment Source
url	string	URL where the attachment is located. Either url or (content and mimeType) attributes MUST be provided during creation.	URL

7.2.3.2. enum DataSizeUnit

Description: The unit of measure in the data size.

Value
BYTES
KBYTES
MBYTES
GBYTES
TBYTES
PBYTES
EBYTES
ZBYTES
YBYTES

7.2.3.3. Type FieldedAddress

Description: A type of Address that has a discrete field and value for each type of boundary or identifier down to the lowest level of detail. For example "street number" is one field, "street name" is another field, etc. Reference: MEF 79 (Sn 8.9.2)

Name	Type	Description	MEF 113
city*	string	The city that the address is in	City
country*	string	Country that the address is in	Country
geographicSubAddress	GeographicSubAddress	Additional fields used to specify an address, as detailed as possible.	Not represented in MEF 113

Name	Type	Description	MEF 113
locality	string	The locality that the address is in	Locality
postcode	string	Descriptor for a postal delivery area, used to speed and simplify the delivery of mail (also known as zip code)	Postal Code
postcodeExtension	string	An extension of a postal code. E.g. the part following the dash in a US urban property address	Postal Code Extension
stateOrProvince	string	The State or Province that the address is in	State Or Province
streetName*	string	Name of the street or other street type	Street Name
streetNr	string	Number identifying a specific property on a public street. It may be combined with streetNrLast for ranged addresses. MEF 79 defines it as required however as in certain countries it is not used we make it optional in API.	Street Number
streetNrLast	string	Last number in a range of street numbers allocated to a property	Street Number Last
streetNrLastSuffix	string	Last street number suffix for a ranged address	Street Number Suffix Last
streetNrSuffix	string	The first street number suffix	Street Number Suffix
streetSuffix	string	A modifier denoting a relative direction	Street Suffix

Name	Type	Description	MEF 113
streetType	string	The type of street (e.g., alley, avenue, boulevard, brae, crescent, drive, highway, lane, terrace, parade, place, tarn, way, wharf)	Street Type

7.2.3.4. Type GeographicSubAddress

Description: Additional fields used to specify an address, as detailed as possible.

Name	Type	Description	MEF 113
buildingName	string	Allows for identification of places that require building name as part of addressing information	Building Name
id	string	Unique Identifier of the subAddress	Not represented in MEF 113
levelNumber	string	Used where a level type may be repeated e.g. BASEMENT 1, BASEMENT 2	Level Number
levelType	string	Describes level types within a building	Level Type
privateStreetName	string	"Private streets internal to a property (e.g. a university) may have internal names that are not recorded by the land title office	Private Street Name
privateStreetNumber	string	Private streets numbers internal to a private street	Private Street Number
subUnit	MEFSubUnit[]	Representation of a MEFSubUnit It is used for describing subunit within a subaddress e.g.BERTH, FLAT, PIER, SUITE, SHOP, TOWER, UNIT, WHARF.	Not represented in MEF 113

7.2.3.5. enum MEFBuyerSellerType

Description: An enumeration with buyer and seller values.

Value	MEF 113
buyer	BUYER
seller	SELLER

7.2.3.6. Type MEFByteSize

Description: A size represented by value and Byte units

Name	Type	Description	MEF 113
amount	float	Numeric value in a given unit	Value
units	DataSizeUnit	Byte Unit	Unit

7.2.3.7. Type MEFGeographicPoint

Description: A MEFGeographicPoint defines a geographic point through coordinates. Reference: MEF 79 (Sn 8.9.5)

Inherits from:

- RelatedPlaceRefOrValue

Name	Type	Description	MEF 113
spatialRef*	string	The spatial reference system used to determine the coordinates (e.g. "WGS84"). The system used and the value of this field are to be agreed during the onboarding process.	Spatial Reference
x*	string	The latitude expressed in the format specified by the `spacialRef`	Latitude
y*	string	The longitude expressed in the format specified by the `spacialRef`	Longitude
z	string	The elevation expressed in the format specified by the `spacialRef`	Elevation

7.2.3.8. Type MEFSubUnit

Description: Allows for sub unit identification

Name	Type	Description	MEF 113

Name	Type	Description	MEF 113
subUnitNumber*	string	The discriminator used for the subunit, often just a simple number but may also be a range.	Sub Unit Name
subUnitType*	string	The type of subunit e.g.BERTH, FLAT, PIER, SUITE, SHOP, TOWER, UNIT, WHARF.	Sub Unit Type

7.2.3.9. Type Note

Description: Extra information about a given entity. Only useful in processes involving human interaction. Not applicable for automated process.

Name	Type	Description	MEF 113
author*	string	Author of the note	Note Author
date*	date-time	Date of the note	Note Date
id*	string	Identifier of the note within its containing entity (may or may not be globally unique, depending on provider implementation)	Not represented in MEF 113
source*	MEFBuyerSellerType	Indicates if this Note was added by the Buyer or Seller.	Note Source
text*	string	Text of the note	Note Text

7.2.3.10. Type RelatedContactInformation

Description: Contact data for a person or organization that is involved in a given context. It is specified by the Seller (e.g. Seller Contact Information) or by the Buyer.

Name	Type	Description	MEF 113
emailAddress*	string	Email address	Contact email Address
name*	string	Name of the contact	Contact Name
number*	string	Phone number	Contract Phone Number

Name	Type	Description	MEF 113
numberExtension	string	Phone number extension	Contract Phone Number Extension
organization	string	The organization or company that the contact belongs to	Contact Organization
postalAddress	FieldedAddress	Identifies the postal address of the person or office to be contacted.	Contact Postal Address
role*	string	A role the party plays in a given context.	Not represented in MEF 113

7.2.3.11. Type RelatedEntity

Description: A reference to an entity, where the type of the entity is not known in advance.

Name	Type	Description	MEF 113
@referredType*	string	The actual type of the target instance when needed for disambiguation.	Not represented in MEF 113
href	string	Reference of the related entity.	Not represented in MEF 113
id*	string	Unique identifier of a related entity.	Product Identifier
role*	string	The role of an entity.	Not represented in MEF 113

7.2.4. Notification registration

Notification registration and management are done through `/hub` API endpoint. The below sections describe data models related to this endpoint.

7.2.4.1. Type EventSubscriptionInput

Description: This class is used to register for Notifications.

API name	MEF 113 name	Description

API name	MEF 113 name	Description
`troubleTicketAttributeValueChangeEvent`	TICKET_UPDATE	The Seller settable attributes for a Ticket were updated by the Seller. Note: Buyer initiated Ticket updates due to Patch operation will not trigger a `troubleTicketAttributeValueChangeEvent`
`troubleTicketInformationRequiredEvent`	TICKET_STATE_CHANGE	A Ticket `status` was changed by the Seller.
`troubleTicketResolvedEvent`	TICKET_INFO_REQUIRED	The Seller requires more information from the Buyer for a Ticket to continue processing a Ticket. The details on what information is needed from the Buyer will be provided via a Ticket `note`. The Ticket `status` is `pending`. Note: The Buyer uses the Patch operation to provide more information for a Ticket.
`troubleTicketStatusChangeEvent`	TICKET_RESOLVED	The Seller is informing the Buyer the Ticket is resolved and the Buyer to verify that the Issue on which the Ticket was based is no longer observed. The Ticket `status` is `resolved`. Note: The Buyer confirms if the Issue has been resolved satisfactorily or not using close or reopen operations
`incidentCreateEvent`	INCIDENT_CREATE	A new Incident was created by the Seller.
`incidentAttributeValueChangeEvent`	INCIDENT_UPDATE	An open Incident was updated by the Seller.
`incidentStatusChangeEvent`	INCIDENT_STATE_CHANGE	An Incident `status` was changed by the Seller.

Name	Type	Description

Name	Type	Description
callback*	string	This callback value must be set to *host* property from Buyer Notification API (tr This property is appended with the base path and notification resource path specifie which notification is sent. E.g. for "callback": "http://buyer.com/listenerEndpoint", will be sent to: `http://buyer.com/listenerEndpoint/mefApi/sonata/troubleTicketNotification/v2/list
query	string	This attribute is used to define to which type of events to register to. Example: "que troubleTicketStatusChangeEvent". To subscribe for more than one event type, put t `eventType=troubleTicketStatusChangeEvent,troubleTicketResolvedEvent`. The p 'TroubleTicketEventType' in troubleTicketNotification.api.yaml. An empty query is ending in subscription for all event types.

7.2.4.2. Type EventSubscription

Description: Sets the communication endpoint address the service instance must use to deliver notification information

Name	Type	Description	MEF 113
callback*	string	The value provided by the Buyer in `EventSubscriptionInput` during notification registration	Notification Target Information
id*	string	An identifier of the event subscription assigned by the Seller when a resource is created.	Not represented in MEF 113
query	string	This attribute is used to define notification registration constraints.	List of Notification Event Types, Action

7.3. Notification API Data model

Figure 20 presents the Trouble Ticket Management Notification data model.

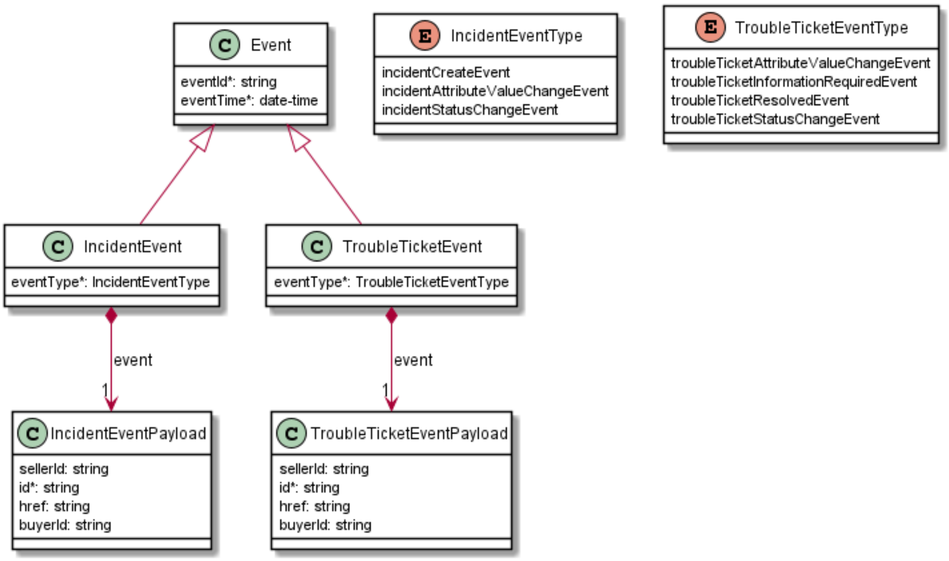

Event
eventId*: string
eventTime*: date-time
IncidentEventType
incidentCreateEvent
incidentAttributeValueChangeEvent
incidentStatusChangeEvent
TroubleTicketEventType
troubleTicketAttributeValueChangeEvent
troubleTicketInformationRequiredEvent
troubleTicketResolvedEvent
troubleTicketStatusChangeEvent
IncidentEvent
eventType*: IncidentEventType
TroubleTicketEvent
eventType*: TroubleTicketEventType
event
1
event
1
IncidentEventPayload
sellerId: string
id*: string
href: string
buyerId: string
TroubleTicketEventPayload
sellerId: string
id*: string
href: string
buyerId: string

Figure 20. Trouble Ticket Management Notification Data Model

This data model is used to construct requests and responses of the API endpoints described in Section 5.2.2.

7.3.1. Type Event

Description: Event class is used to describe information structure used for notification.

Name	Type	Description	MEF 113
eventId*	string	Id of the event	Not represented in MEF 113
eventTime*	date-time	Datetime when the event occurred	Not represented in MEF 113

7.3.2. Type TroubleTicketEvent

Description:

Inherits from:

- Event

Name	Type	Description	MEF 113
eventType*	TroubleTicketEventType	Indicates the type of the event.	Notification Type
event*	TroubleTicketEventPayload	A reference to the object that is source of the notification.	Not represented in MEF 113

7.3.3. `enum` TroubleTicketEventType

Description: Type of the Trouble Ticket event.

API name	MEF 113 name	Description
troubleTicketAttributeValueChangeEvent	TICKET_UPDATE	The Seller settable attributes for a Ticket were updated by the Seller. Note: Buyer initiated Ticket updates due to Patch operation will not trigger a troubleTicketAttributeValueChangeEvent
troubleTicketInformationRequiredEvent	TICKET_STATE_CHANGE	A Ticket status was changed by the Seller.
troubleTicketResolvedEvent	TICKET_INFO_REQUIRED	The Seller requires more information from the Buyer for a Ticket to continue processing a Ticket. The details on what information is needed from the Buyer will be provided via a Ticket note. The Ticket status is pending. Note: The Buyer uses the Patch operation to provide more information for a Ticket.
troubleTicketStatusChangeEvent	TICKET_RESOLVED	The Seller is informing the Buyer the Ticket is resolved and the Buyer to verify that the Issue on which the Ticket was based is no longer observed. The Ticket status is resolved. Note: The Buyer confirms if the Issue has been resolved satisfactorily or not using close or reopen operations

7.3.4. Type TroubleTicketEventPayload

Description: The identifier of the Trouble Ticket being subject of this event.

Name	Type	Description	MEF 113
sellerId	string	The unique identifier of the organization that is acting as the Seller. MUST be specified in the request only when requester entity represents more than one Seller.	Seller

Name	Type	Description	MEF 113
id*	string	ID of the Trouble Ticket attributed by quoting system	Not represented in MEF 113
href	string	Hyperlink to access the Trouble Ticket	Not represented in MEF 113
buyerId	string	The unique identifier of the organization that is acting as the a Buyer. MUST be specified in the request only when the responding represents more than one Buyer.	Buyer

7.3.5. Type IncidentEvent

Description:

Inherits from:

- Event

Name	Type	Description	MEF 113
eventType*	IncidentEventType	Indicates the type of the event.	Notification Type
event*	IncidentEventPayload	A reference to the object that is source of the notification.	Not represented in MEF 113

7.3.6. Type IncidentEventPayload

Description: The identifier of the Incident being subject of this event.

Name	Type	Description	MEF 113
sellerId	string	The unique identifier of the organization that is acting as the Seller. MUST be specified in the request only when requester entity represents more than one Seller.	Seller
id*	string	ID of the Incident attributed by quoting system	Not represented in MEF 113

Name	Type	Description	MEF 113
href	string	Hyperlink to access the Incident	Not represented in MEF 113
buyerId	string	The unique identifier of the organization that is acting as the a Buyer. MUST be specified in the request only when the responding represents more than one Buyer.	Buyer

7.3.7. `enum` IncidentEventType

Description: Type of the Incident event.

API name	MEF 113 name	Description
`incidentCreateEvent`	INCIDENT_CREATE	A new Incident was created by the Seller.
`incidentAttributeValueChangeEvent`	INCIDENT_UPDATE	An open Incident was updated by the Seller.
`incidentStatusChangeEvent`	INCIDENT_STATE_CHANGE	An Incident `status` was changed by the Seller.

8. References

- [OAS-v3] Open API 3.0, February 2020
- [MEF55.1] MEF 55.1, Lifecycle Service Orchestration (LSO): Reference Architecture and Framework, February 2021
- [MEF79] MEF 79, Address, Service Site, and Product Offering Qualification Management, Requirements and Use Cases, November 2019
- [MEF80] MEF 80, Quote Management Requirements and Use Cases, July 2021
- [MEF113] MEF 113 Trouble Ticketing Business Requirements and Use Cases, July 2022
- [MEF128] MEF 128, LSO API Security Profile, July 2022
- [MEF137] MEF 137 LSO Cantata and LSO Sonata Appointment Management API - Developer Guide, October 2022
- [REST] Chapter 5: Representational State Transfer (REST) Fielding, Roy Thomas, Architectural Styles and the Design of Network-based Software Architectures (Ph.D.).
- [RFC2119] RFC 2119, Key words for use in RFCs to Indicate Requirement Levels, by S. Bradner, March 1997
- [RFC3986] RFC 3986 Uniform Resource Identifier (URI): Generic Syntax, January 2005
- [RFC8174] RFC 8174, Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words, by B. Leiba, May 2017, Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.
- [TMF621] TMF 621, Trouble Ticket API REST Specification R19.0.1, November 2019
- [TMF630] TMF 630 TMF630 API Design Guidelines 4.2.0

Appendix A Acknowledgments

Mike **BENCHECK**

Michał **ŁĄCZYŃSKI**

Jack **PUGACZEWSKI**

Patrick **ROOSEN**

Karthik **SETHURAMAN**