Clearing Inter-Platform API Specification

Stand: **03.11.2026**

Version: **1.2.0**

Notice

Copyright © by the Authors 2025 and Arbeitskreis Schnittstellen und Prozesse. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works.

License might be changed, and the document might be published under open-source license at a later point in time based on agreements with “Arbeitskreis Schnittstellen und Prozesse”.

Authors

BITConEx GmbH	Contact Halid Zehic Tel.: +49 (0) 160 / 7839658 halid.zehic@bitconex.de
holos supply GmbH	Contact Rene Schäflein Tel.: +49 (0) 1522 278 17 18 rene.schaeflein@holossupply.de
ARBEITSKREIS SCHNITTSTELLEN & PROZESSE	Contact Kontaktformular

1 Introducing connected platforms 2
2 Version compatibility 3
2.1 Compatibility matrix 3
2.2 Compatibility rules 3
3 Syncing tickets between platforms 5
3.1 Trouble Ticket Sync Event 5
3.1.1 Create Ticket 6
3.1.2 Set Status 6
3.1.3 Set Resolved 7
3.1.4 Change Severity 7
3.1.5 Add a Note 8
3.1.6 Change Data 8
3.2 Request Ticket Cancellation 9
3.3 Request Customer Migration 10
3.4 Access control 13
4 Syncing carriers between platforms 13
4.1 Carrier Sync Event 15
4.2 Handling of Non-Clearing Organizations (NCOs) 18
5 Attachments 19
6 Checking tickets 20
6.1 Access Control 20
7 Operational Exception Guidelines 21
7.1 Failure Handling Principles 21
7.1.1 Technical errors 21
7.1.2 Logical errors 21
7.1.3 Illegal requests 22
7.2 Unexpected situations 22
8 Customer Migration Between Platforms 22
8.1 Preconditions 23
8.2 Process overview 23
8.3 Migration error handling 24
9 Open Issues 24
9.1 Unexpected situations 25

9.2 Automatic cancellation 25
9.3 Other Topics 25
10 Change Log 25
10.1 Version 1.0 to Version 1.1 26
10.2 Version 1.1 to Version 1.1.1 26
10.3 Version 1.1.1 to 1.2 26

1 Introducing connected platforms

The Inter-Platform API specification defines the mechanisms by which clearing platforms exchange data within a multi-platform network.

A clearing platform is a system that fully implements the official Clearing Specification, which consists of:

- [MAIN-1] Clearing_Platform_REST_API_Specification
- [REF-4] Clearing_Platform_Metadata
- [REF-6] Clearing Platform Specification Test Cases.xlsx
- [REF-7] Clearing_Platform_NCO_handling_V.1

The following documents are also part of this specification

- [REF-1] Inter-Platform_API_V1.1.yaml
- [REF-2] Inter Platform Specification Test Cases.xlsx

A platform can only participate in inter-platform communication if it has successfully implemented this specification. This ensures interoperability and minimizes the likelihood of rule validation exceptions during data synchronization.

> ***Note**: The successful implementation of the clearing specification must be demonstrated in an appropriate and verifiable way before onboarding into the inter-platform network. There is a test plan that must be verified by at least one of the existing platforms.*

All platforms within the multi-platform network need to know each other.

For the distribution of the relevant information, there is no dedicated API. Instead, this process is handled informally by the UAG Clearing group.

- Each platform is identified by a unique ITU Carrier Code.
- Each platform must provide all other platforms with:
  - the base URL of its Inter-Platform API
  - a set of OAuth2 credentials for secure communication

2 Version compatibility

2.1 Compatibility matrix

This matrix defines which Clearing Platform versions are supported by each Inter Platform version.

Version	Referenced	Compatible	Notes
1.0	1.1	1.1, 1.2, 1.2.1, 1.2.2	Baseline mapping
1.1	1.2.2	1.2.2	Added unsupportedScenarios support
1.1.1	1.2.2	1.2.2	Clarified that supported scenario validation is only performed for the receiver role during create ticket processing.
1.2	1.3	1.2.2, 1.3	Replaced hourly carrier polling with event driven carrier synchronization via carrierSyncEvent.

- Version: Inter-Platform specification version.
- Referenced: Clearing Platform version used as the baseline/reference for this Inter Platform version.
- Compatible: Additional Clearing Platform versions validated as compatible for this Inter Platfom version.
- Notes: Important context for compatibility decisions.

Note: Only listed version combinations are supported. Unlisted combinations are not supported/not validated.

2.2 Compatibility rules

- Inter Platform is pinned to a specific Clearing Platform version.
- Clearing Platform updates that do not change models referenced by Inter Platform are compatible without Inter Platform version bump
- Clearing Platform updated that do change models require an Inter Platform update

Example:

Clearing Platform - Active version: c.n Next version: c.n+1

Inter Platform - Active version: i.n Next version: i.n+1

- **Case A:** Clearing c.n+1 has no changes in Inter-referenced models
    - Inter i.n remains compatible.
    - No Inter API version bump required.
    - Matrix can be extended:
        - i.n -> c.n, c.n+1 (compatible by unchanged referenced models)

- **Case B**: Clearing c.n+1 changes a referenced model
    - Inter i.n becomes incompatible (or conditionally incompatible).
    - Release new Inter version (e.g. i.n+1) aligned to Clearing c.n+1
    - Matrix update:
        - i.n -> c.n
        - i.n+1 -> c.n+1

3 Syncing tickets between platforms

3.1 Trouble Ticket Sync Event

For syncing tickets between platforms there is just one sync method

```
/listener/troubleTicketSyncEvent:
  post:
    tags:
      - NotificationListener
    summary: Client listener for ClearingTicket sync event
    description: Client listener for receiving the ClearingTicket sync
        notification
    operationId: listenToTicketSyncEvent
    requestBody:
      description: The event data
      content:
        application/json;charset=utf-8:
          schema:
            $ref: '#/components/schemas/ClearingTicketEvent'
      required: true
    responses:
      '200':
        description: OK - Notified
        content: {}
    x-codegen-request-body-name: data
```

The request body of type `ClearingTicketEvent` contains

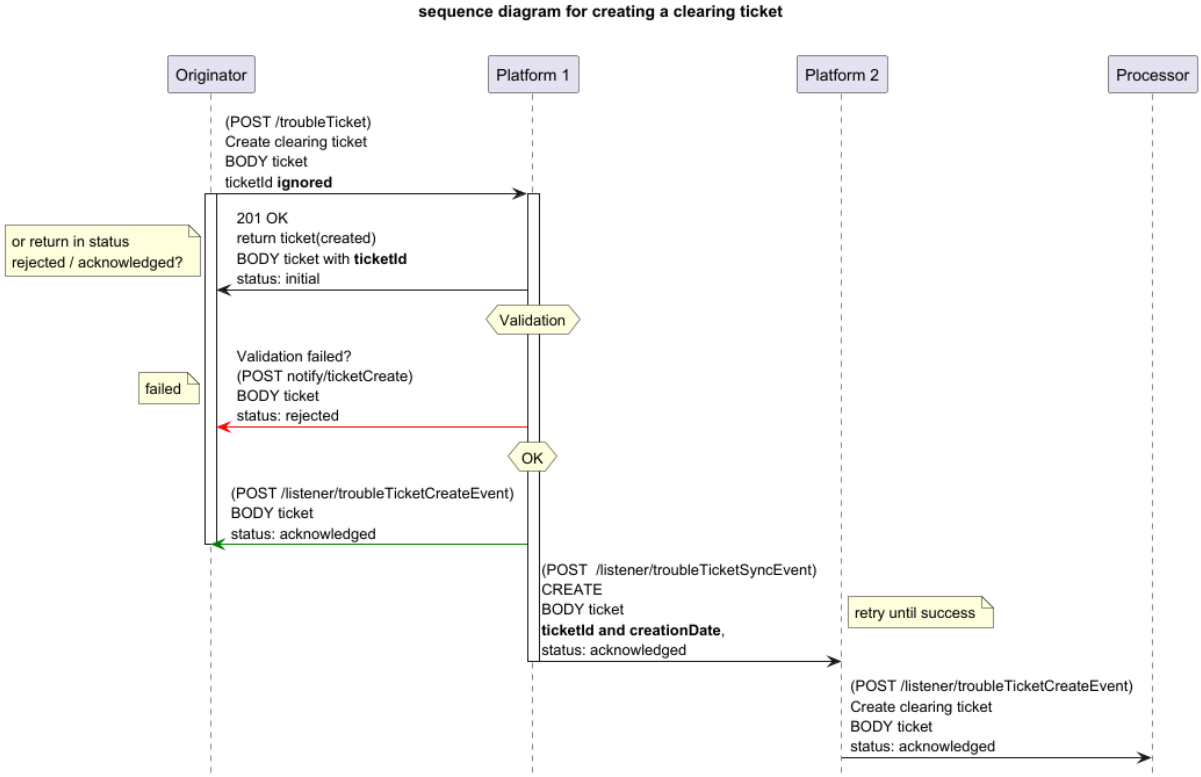
- The **initiator** of the request (ITU-Code)
- the entire **clearing ticket** – schema according to the most recent specification (see [MAIN-1] [REF-4])
- the **event type** (enum)
    - **CREATE**
    - **STATUS**
    - **RESOLVED**
    - **NOTE**
    - **DATA**

- **SEVERITY**

Reason: The tickets on either platform must be **exactly** in sync.

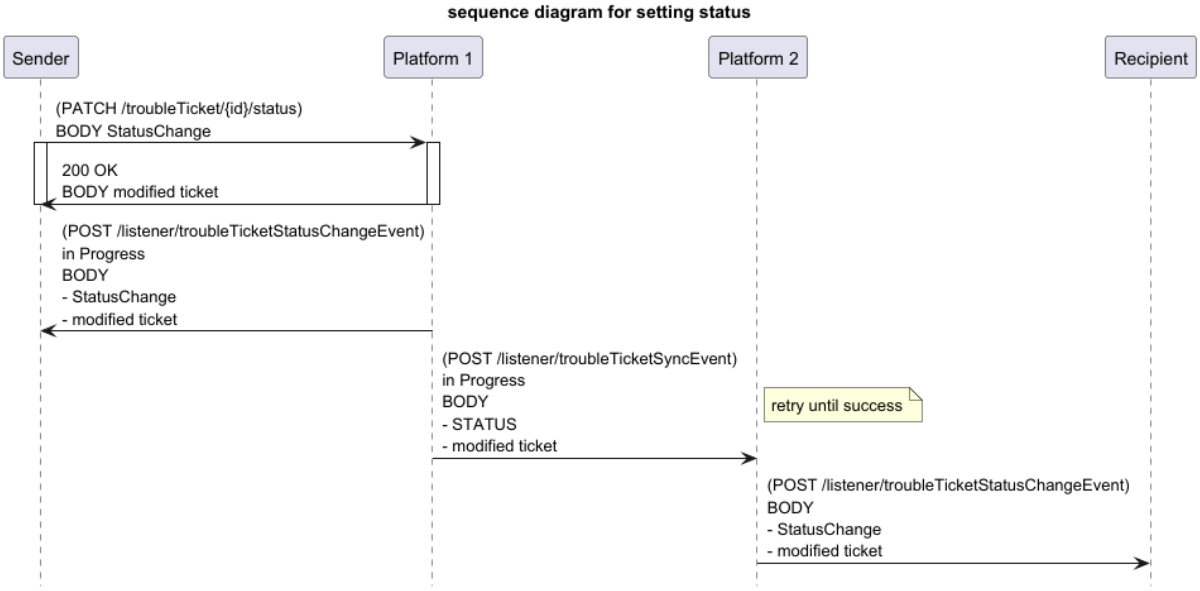
Benefit: We can just take the ticket as is and replace it on the target platform while still knowing the reason for this notification. This helps to notify the connected partners properly.

3.1.1 Create Ticket



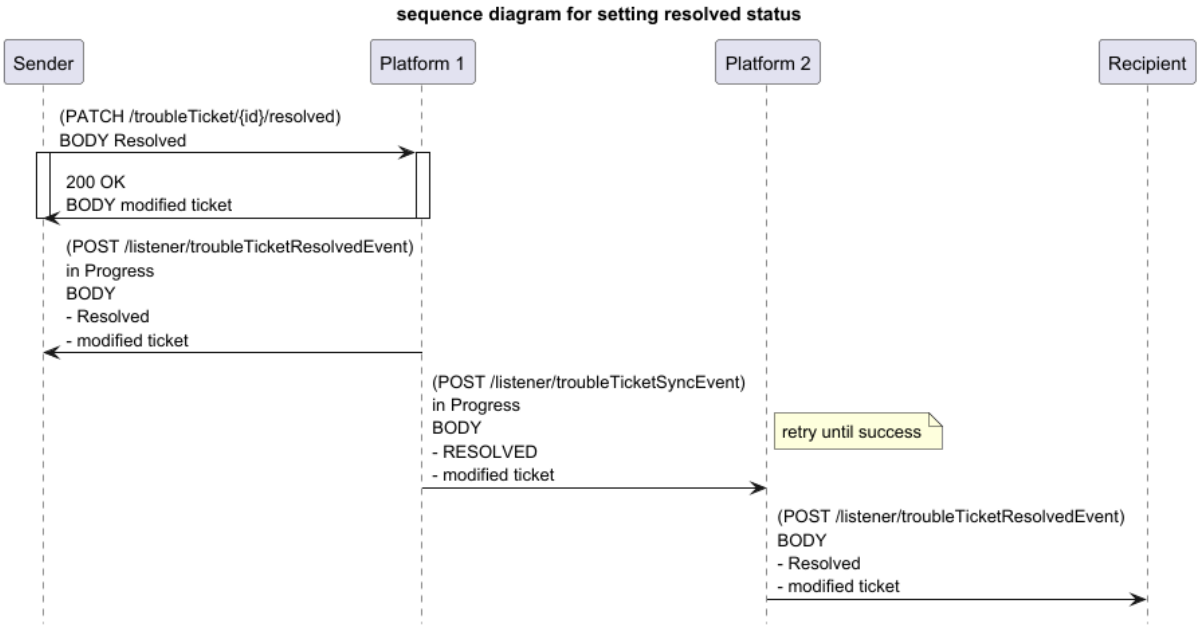
- Validation of supported scenarios shall only be performed for the receiver role of the ticket.

3.1.2 Set Status

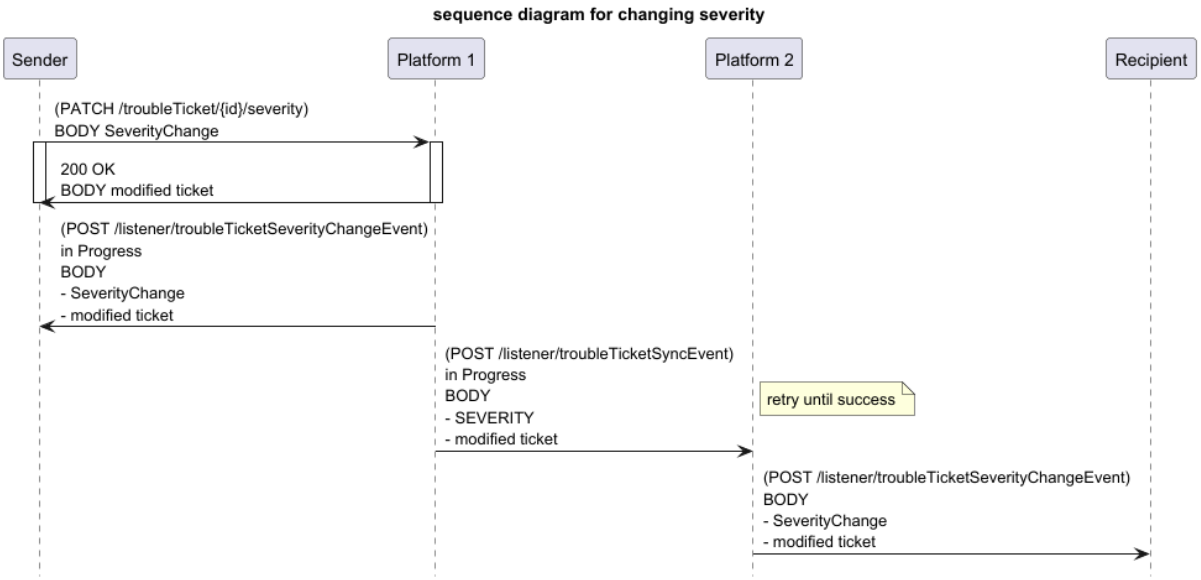


Note: There is also no sync event for Final, since Final is not an official lifecycle event.

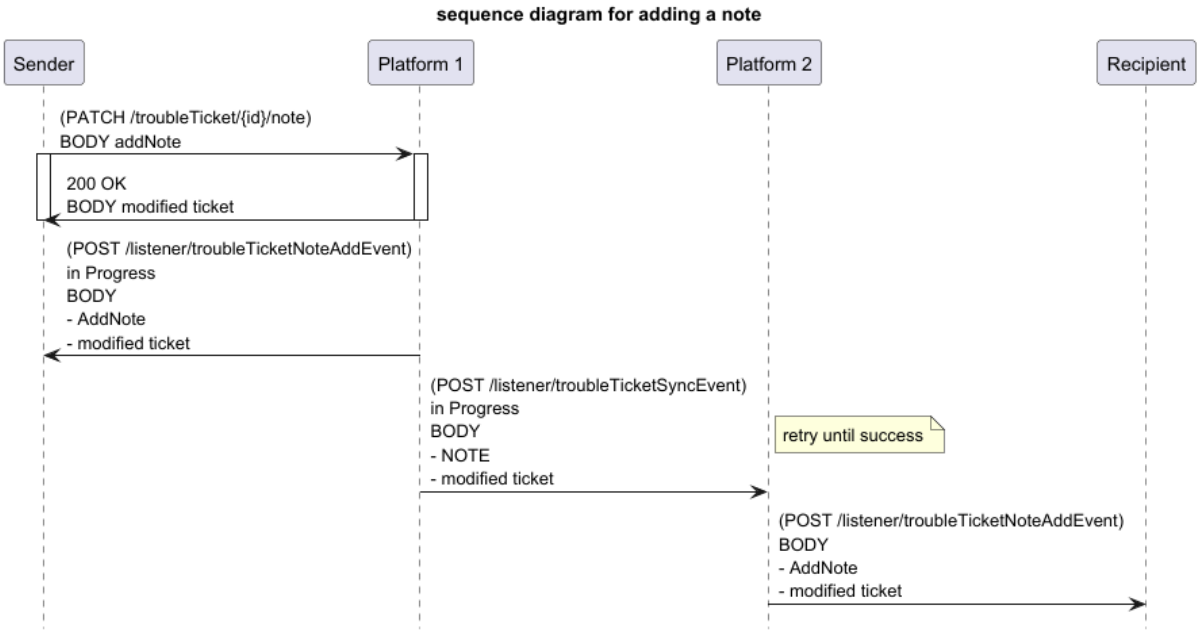
3.1.3 Set Resolved



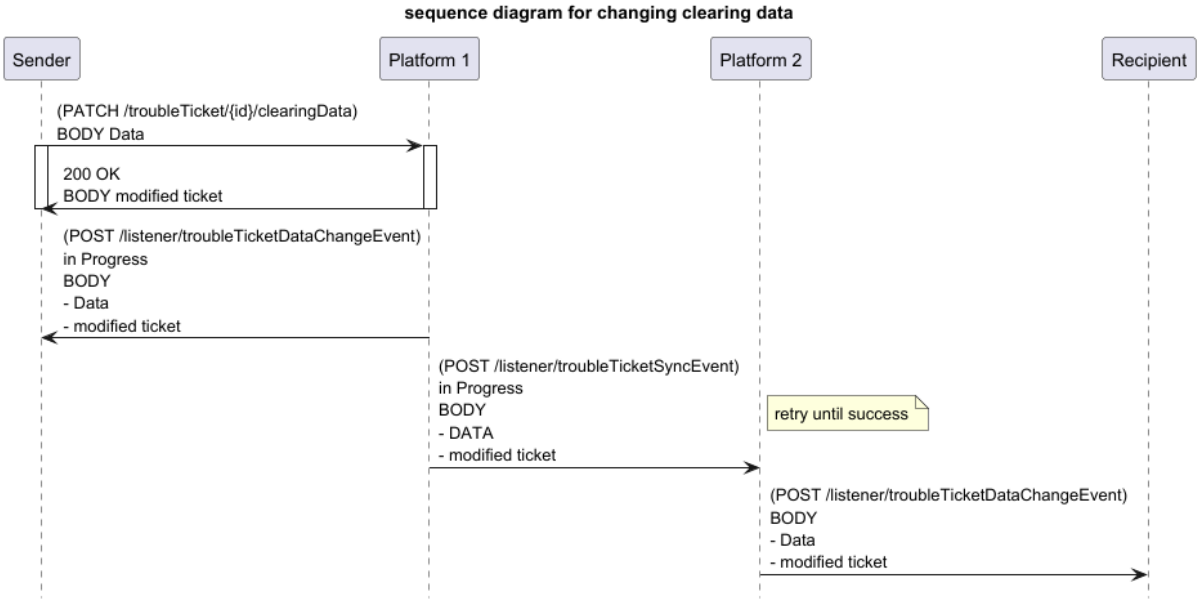
3.1.4 Change Severity



3.1.5 Add a Note



3.1.6 Change Data



3.2 Request Ticket Cancellation

Cancellation ticket requests allow a processor platform to formally request the cancellation of a ticket on the connected platform during problem resolution or escalation.

```
/listener/troubleTicketCancelRequest/{id}:
  post:
    tags:
      - IpcNotificationListener
    summary: Request cancellation of a ClearingTicket
    description: This operation allows a processor to request cancellation of a
ClearingTicket.
    operationId: requestTicketCancellation
    parameters:
      - name: id
        in: path
        description: Identifier of the ClearingTicket to be cancelled
        required: true
        schema:
          type: string
      - name: originatorItuCode
        in: query
        description: ITU code of the ticket's originator
        required: true
        schema:
          type: string
```

```
      - name: reason
        in: query
        description: Optional reason for the cancellation request
        required: false
        schema:
          type: string
    responses:
      '202':
        description: OK - Cancellation request accepted for processing
        content: {}
      '400':
        description: Bad request (e.g., ticket cannot be cancelled due to current
state)
        content: {}
      '404':
        description: Ticket not found
        content: {}
      '422':
        description: Unprocessable Entity (e.g., already cancelled or duplicate
request)
        content:
          application/json;charset=utf-8:
            schema:
              $ref: '#/components/schemas/SyncError'
```

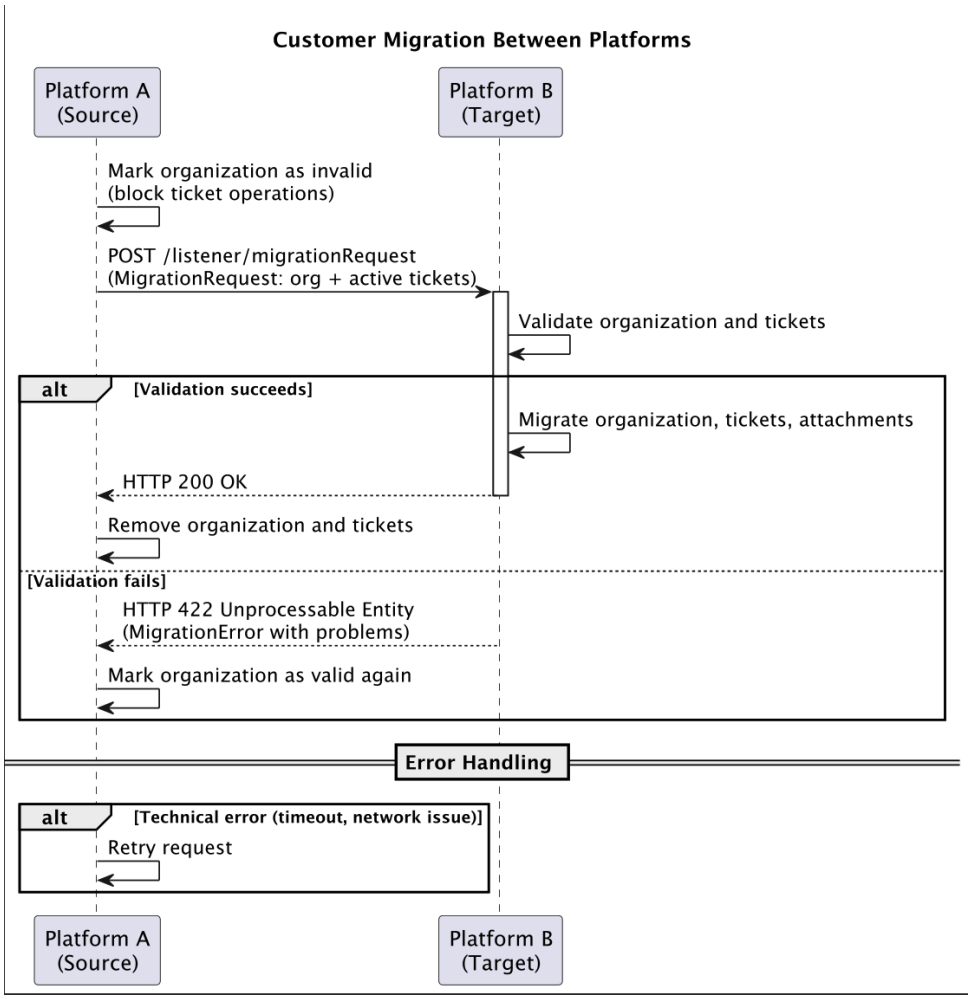
Parameters explanation:

- **`id`**: Identifier of the ClearingTicket to be cancelled.
- **`originatorItuCode`**: ITU code of the ticket's originator.
- **`reason`**: cancellation reason.

Post-Request Handling:

Upon receiving the cancellation request at /listener/troubleTicketCancelRequest/{id}, the originator platform **must change the ticket state to "cancelled"** and subsequently **sync the updated ticket state with the processor platform** via the established ticket synchronization mechanism.

3.3 Request Customer Migration



Migration request allows a platform to formally transfer an organization, including all its active tickets and related attachments, to another platform.
This process is used when a customer changes its hosting platform. It ensures that the complete state of the organization is moved in a consistent way, while preserving ticket IDs for traceability across platforms.

```
/listener/migrationRequest:
  post:
    tags:
      - IpcNotificationListener
```

```
    summary: Initiate a migration request for an organization
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/MigrationRequest'
    responses:
      '200':
        description: Migration succeeded. All tickets and attachments migrated,
          ticket IDs remain unchanged.
      '422':
        description: Migration failed due to validation errors. Returns list of
detected problems.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MigrationError'
```

The request body of type **MigrationRequest** contains:

- **The organization** to be migrated.
- **All active tickets** belonging to that organization (not closed or cancelled), each provided in full according to the most recent **ClearingTicket** specification (see [MAIN-1] and [REF-4]).
- **All attachments** related to those tickets, transferred together during the synchronous migration process.

Reason: The migration must guarantee that the complete state of the organization (tickets + attachments) is transferred consistently in a single operation.

Benefit: The receiving platform can directly replace the full dataset (organization, tickets, and attachments) and maintain traceability through unchanged ticket IDs. This ensures that the migrated organization is immediately usable on the target platform.

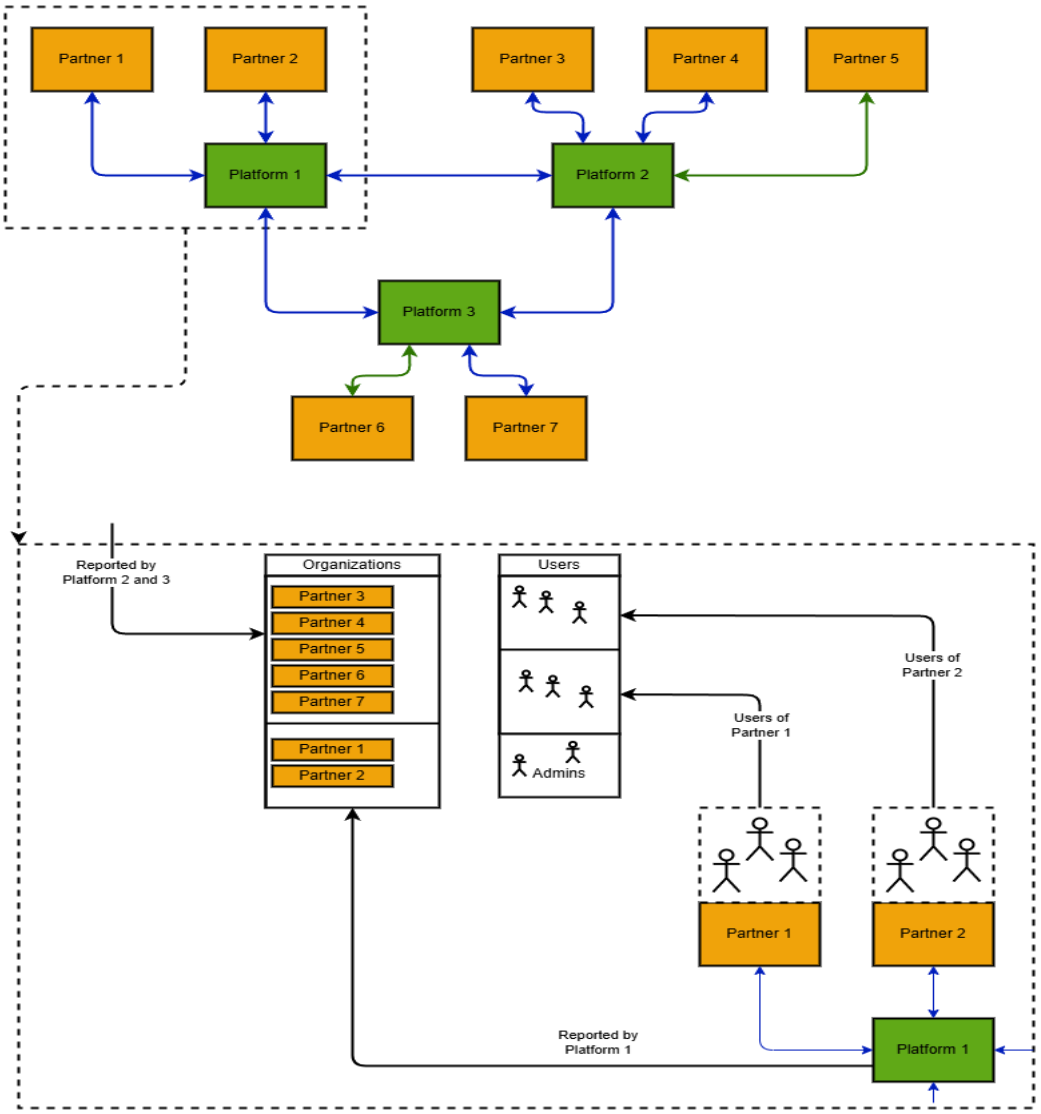
3.4 Access control

A platform may only execute any action on the ticket if either the **originator** or the **processor** of the ticket is registered on that platform.
If the caller is not a party to the ticket, the request must be rejected with an appropriate response `422 Unprocessed Entity`.

4 Syncing carriers between platforms

Each platform owns data about its own partners and their users. It needs to share its information about their partners with all other platforms while users are only known within the respective platform, strictly separated for each partner.

> ***Note**: The concept of a carrier in the Inter-Platform API refers to the high-level registration of a partner (identified by an ITU Carrier Code) and the associated platform they are hosted on. This is distinct from the full carrier or partner structure defined in [REF-4]).*



Carrier synchronization between platforms is event driven.

Whenever a carrier is created, updated or deleted on one platform, the originating platform must notify all other connected platforms immediately via the carrier sync event.

The following listener endpoint is used for this purpose:

- /listener/carrierSyncEvent

The event contains the carrier id, the type of change and, for create and update events, the full carrier object.

The following endpoints are available for lookup, recovery and reconciliation purpose:

- /carrier List of carriers on the platform
- /carrier/{id} Obtain information about carrier

4.1 Carrier Sync Event

For syncing carriers between platforms there is just one sync method

```
  /listener/carrierSyncEvent:
    post:
      tags:
        - IpcNotificationListener
      summary: Client listener for carrier sync event
      description: Client listener for receiving carrier create, update or delete
          notifications.
      operationId: listenToCarrierSyncEvent
      requestBody:
        description: The event data
        content:
          application/json;charset=utf-8:
            schema:
              $ref: '#/components/schemas/CarrierEvent'
        required: true
      responses:
        '200':
          description: OK - Notified
          content: {}
...
```

The request body of type `CarrierEvent` contains

- The **initiator** of the request (ITU-Code)
- the **event type** (enum)
  - **CREATE**
  - **UPDATE**
  - **DELETE**
- The event time
- The id of the affected carrier
- For CREATE and UPDATE events: the full carrier object

```
Carrier:
      type: object
      description: ItuCarrier
      required:
        - id
      properties:
        id:
          type: string
          description: Unique identifier of the ItuCarrier - DEU.XXXXX
        tradingName:
          type: string
          description: Name that the Carrier (unit) trades under
        validFrom:
          type: string
          format: date
          description: Validity of this carrier on this platform starts with
          'validFrom'
        validTo:
          type: string
          format: date
          description: Validity of this carrier on this platform ends with
          (including) 'validTo'
        flag:
          type: string
          description: Mark a special organization e.g. as demo or test
          organization
        unsupportedScenarios:
          description: List of all unsupported scenarios
          type: array
          items:
            description: Scenario ids like 1.01, no main scenario ids or wildcards
          are allowed.
            type: string
```

Processing rules:

- On CREATE, the receiving platform must create or activate the transmitted carrier entry.
- On UPDATE, the receiving platform must update the stored carrier data accordingly.
- On DELETE, the receiving platform must remove or deactivate the carrier entry accordingly.

If a carrier referenced by a ticket is unknown on the receiving platform, the platform must actively check whether the locally stored carrier data is out of date. For this purpose, the platform may call: /carrier/{id}.

If a GET /carrier/{id} call returns 404 Not Found, the calling platform must assume that the carrier has been removed or de-registered on the remote platform.

Since carrier synchronization is event driven, there is no need to wait for a scheduled full sync. The receiving platform must update its local carrier state immediately based on the lookup result and handle any dependent data accordingly.

As a fallback and reconciliation mechanism, platforms may additionally perform a full carrier synchronization via GET /carrier, for example after restart, recovery from communication errors or suspected event loss. Such a full synchronization is optional and does not replace the event driven mechanism.

4.2 Handling of Non-Clearing Organizations (NCOs)

Currently, only registered clearing platforms and their associated carriers participate in inter-platform ticket synchronization. However, scenarios arise where **Non-Clearing Organizations (NCOs =** partners or customers not connected to any clearing platform) are involved.

Read more about **NCOs** in: REF_7

For Inter-Platform communication, NCOs are fully ignored.
This means:

- NCOs are **never synced** between platforms.
- They are only visible to other carriers on the platform where they are introduced.

- hence, they might exist on several platforms simultaneously

As soon as a carrier, which was introduced as NCO on any platform, will then register on a platform

- If the carrier was **already** introduced as NCO on the **receiving platform**, it is just **transitioned** into a regular carrier synchronized between the platforms. It is then up to the platform how to deal with their active tickets.
- If the carrier was **not** introduced as NCO on the **receiving platform**, the registration is done regularly.
- In either case, the registration of the carrier on the receiving platform must happen according to the rules in chapter 4 Syncing carriers between platforms.
- for all **other platforms**, where the carrier has been introduced as NCO: with the first synchronization the carrier must immediately be removed as **NCO.** It is then up to the platform to deal with their active tickets, as they can no longer be processed.

Note: To avoid any problems with active tickets, ideally the NCO should get rid of any active ticket beforehand and actively inform all platforms about the plan to become a registered carrier.

5 Attachments

Attachments also need to be accessed via API using the method

`/attachment/{id}`

It is up to the implementation whether to:

- Always retrieve an attachment from hosting platform
- Cache a copy of all those attachments locally.

If the requested attachment is not available — whether due to deletion, virus check failure, or any other issue — the platform must treat the attachment as **missing** and will not be able to render it to its customer.

6 Checking tickets

To ensure data consistency across platforms, each system must be able to inspect the current state of a specific ticket as maintained by the other platform. This is especially useful for troubleshooting synchronization issues or resolving conflicts during escalations.

The Inter-Platform API provides the following endpoint:

- `/troubleTicket/{id}` - Retrieves a single ticket identified by its unique ID.

Note: Bulk ticket query will not be supported in current version of the Inter-Platform API .

6.1 Access Control

A platform may only request a ticket if either the **originator** or the **processor** of the ticket is registered on that platform.
If the caller is not a party to the ticket, the request must be rejected with an appropriate response `403 Forbidden`.

7 Operational Exception Guidelines

7.1 Failure Handling Principles

Errors encountered during platform communication can be categorized into the following types:

- **Technical errors** include timeouts, connection failures, and HTTP status codes in the 4xx (except 422) and 5xx ranges. These typically indicate network issues, temporary unavailability, or unexpected server errors.
- **Logical errors** request was syntactically correct but semantically invalid due to violation of Clearing specification rules.
- **Illegal requests** are errors caused by invalid request or even payload syntax.

7.1.1 Technical errors

On technical errors platform **must retry** the request. The idea is that the error may disappear as soon as some action is taken by the receiver of the request.

If retries are unsuccessful after a certain number of attempts or time interval, the platform must inform the other platform via email and the issue must be resolved by involved parties through coordination.

> *The number of retries and the interval between retries are not specified in this document and shall be defined operationally*

In such cases, the following escalation procedure shall be applied:

1. Email notification to the other platform
2. The affected platform must resolve the problem within a reasonable amount of time
3. If affected platform is not agreeing to solve the problem at all or in time: escalate to Working group.

7.1.2 Logical errors

On logical errors the receiving platform must respond with HTTP status code **422 (Unprocessable Entity)** to indicate that the request was syntactically correct but violated clearing specification or ticket state conflicts and the reason for the error must be provided in the reason field. The platform

must cancel the ticket automatically (if originator) or request cancellation of the ticket as described in Request Ticket Cancellation, both with a proper "cancel reason" e.g. "We are unable to process the ticket due to a logical issue between platforms. Please try again later."

In such cases, the following escalation procedure shall be applied:

1. Email notification to the other platform
2. Attempt resolution of the problem through direct contact and clarification between the involved platforms, including manual reconciliation of ticket states if cancellation synchronisation fails.
3. If none of the platforms are agreeing to solve the problem escalate to Working group.

7.1.3 Illegal requests

- Requests considered **illegal** (e.g., invalid parameters or operations) must be **immediately rejected** and response for such requests must be HTTPS status code 400 (Bad Request)
- If the illegal request is **expected per specification**, the receiving platform shall handle it according to the spec.
- If the illegal request is **unexpected**, the platform shall follow the **Unexpected situations** defined.

7.2 Unexpected situations

- When cancellation and resolved states conflict, the **"cancelled"** state must prevail and platforms must allow switching ticket state from resolved to cancelled in this case.
- For unexpected error, ticket will be automatically cancelled same as for logical errors

8 Customer Migration Between Platforms

The migration of an organization from one clearing platform to another is a coordinated process between the two platforms

- the platform which will be left by the migrating organization: ***Platform A***
- the platform the migrating organization will move to: ***Platform B***

The goal is to ensure that all active tickets belonging to the migrating organization are transferred.

8.1 Preconditions

- There must be an official request of the migrating organization stating an earliest date.
- The migration must be administratively and informally agreed between the two platform operators, e.g. via email exchange. Hence, the receiving platform must authorize the organization's migration.
- As a best practice, a cutover window (e.g., Friday afternoon → Monday morning) is defined to minimize operational risk.

8.2 Process overview

At the scheduled time, Platform A will mark the migration organization as invalid to avoid ticket operations during the migration. Then

Platform A will call the `/listener/migrationRequest` API of Platform B synchronously,
see chapter 3.3 Request Customer Migration.
The request specifies the organization and the list of all *active* tickets, where active means they are not *closed* or *cancelled.*

The receiving Platform B validates the request and checks if migration is allowed, and all tickets are valid.

If **check succeeds**, Platform B will then proceed with the migration of the organization, its tickets, and their attachments and then return with HTTP status code `200 OK`. Ticket IDs remain unchanged, preserving traceability across platforms.
On successful migration, Platform A will then proceed with removing the organization and all its tickets. From this point on, the organization is hosted exclusively on the new (target) platform. All new tickets involving the organization are created and maintained on the receiving platform only.

If **check fails**, Platform B returns with HTTP status code `422 Unprocessable entity` along with a list of **all** detected problems. Platform A will then proceed with just marking the migrating organization as valid again.

8.3 Migration error handling

The migration is triggered by an operator via a synchronous request.

So, there will be an instant feedback, if the request fails

- Technical errors, e.g., communication timeouts are just retried.
- Logical errors, e.g., invalid organization, invalid ticket(s) result in a HTTP status code `422 Unprocessable entity`. The reported errors need to be clarified bilaterally between the platform operators. After solving the problems that led to this error, the migration is to be rescheduled.

9 Open Issues

Open issues are not part of the current specification and are tracked separately.

Immediately after the current version of the specification has been implemented the open issues need to be resolved and the next implementation must take care of the adjustments.

9.1 Unexpected situations

Even if the specification tries to cover all possible exceptions there is still room for unexpected situations. Here is a short list of those situations:

- An illegal request is indicated by the receiving platform but the calling platform claims that the request is legal according to the rules. Here we again need an escalation strategy
- ...

9.2 Automatic cancellation

Currently, if cancellation synchronization between platforms fails (e.g., due to a 422 error or a subsequent sync failure), ticket states must be reconciled manually by the involved platforms as part of the escalation procedure Cancellation reconciliation.

Open Issue: Define and implement an automated reconciliation mechanism to detect and align mismatched cancellation states between platforms, thereby eliminating the need for manual intervention in these cases.

9.3 Other Topics

This specification does not handle the scenario, where a ticket is updated from both ends at the same time. How is a collision handled? To me a collision may not even be noticed in this specification. (This can happen, if the communication of the platform is interrupted).

10 Change Log

10.1 Version 1.0 to Version 1.1

Introduction of new attribute unsupportedScenarios.

- This new attribute contains a list of scenario ids which are not supportd by a Carrier. See: 3 Syncing carriers between platforms
- Added x-compatibility to define a machine readable version matrix and policy for Inter Platform/Clearing Platform compatibility and required version bumps on referenced model changes.

10.2 Version 1.1 to Version 1.1.1

Clarified that supported scenario validation in inter platform communication applies only to the receiver role of the ticket.

See: 3.1.1 - Create Ticket

10.3 Version 1.1.1 to 1.2

Added carrierSyncEvent to support immediate synchronization of carrier create, update and delete operations between platforms. Hourly polling via /carrier is no longer the primary synchronization mechanism and remains only for lookup and reconciliation.

See: 4.1 Carrier_Sync_Event