Annex F (Interface Specifications)

Annex to the EETS Domain Statement concerning the Danish Kilometer Tolling Scheme

Version: 0.6

Date: 17 May 2023

TABLE OF CONTENTS

1 DOCUMENT HISTORY

Version	Date	Comment	Initials
0.5	10 February 2023	Draft version published to the EETS Provider for information and review purposes as part of the accreditation procedure.	PETV
0.6	17 May 2023	Updated draft version published to the EETS Provider reflecting the current requirements for the accreditation procedure. Final version of the document will be published once the KmToll Law is passed in Danish Parliament.	PETV

2 DEFINITIONS AND ABBREVIATIONS

All definitions in the EETS Domain Statement shall have the same meaning in this Annex.

In addition to the definitions in the EETS Domain Statement the following definitions shall apply for this Annex:

ADU	Application Data Unit
APDU	Application Protocol Data Unit
API	Application Programming Interface
APIM	API Management
ASN	Abstract Syntax Notation
CCC	Compliance Check Communication
CRL	Certificate Revocation List
DSRC	Dedicated Short-Range Communications (DSRC in this context refers to CEN DSRC)
EETS	European Electronic Tolling Service
FQDN	Fully Qualified Domain Name
HTTPS	Hypertext Transfer Protocol Secure
ISO	International Organisation for Standardisation
KmToll	Kilometer Tolling Scheme in Denmark
OBE	On Board Equipment
REST	Representational State Transfer
TC	Toll Charger
TSP	Toll Service Provider

3 INTRODUCTION

This Annex describes the interfaces between EETS Toll Service Providers[1] (TSPs) and Toll Charger (TC) in the Danish Kilometre Tolling (KmToll) scheme. The purpose for this Annex is to get familiar with the interface design and understand the overall requirements and architecture to get started. This Annex is intended for technical personnel that understand APIs and know how to integrate with APIs.

It is not the intention that this specification describes in detail each field that is exposed in the API endpoint, as this will be documented using the OpenAPI v. 3.1 (link) using REST API. This will help reducing the maintenance of this documentation when APIs are going through updates (version-ing).

The architecture of the KmToll scheme is complaint with the ISO 17573 Electronic Fee Collection (EFC) set of standards. DS ISO 17573-1:2019 defines the roles in an EFC system as shown below:

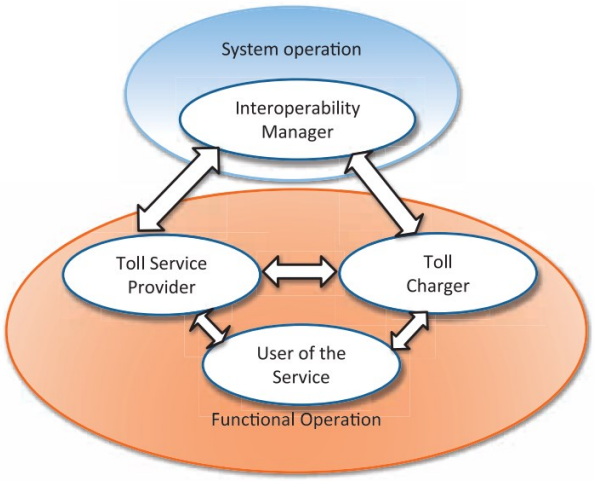


Figure 1. Roles in an EFC scheme as defined in DS ISO 17573-1

Sund & Bælt will be the Toll Charger. Regular Users will buy a Tolling service from EETS Toll Service Providers which are accredited onto the Scheme. Occasional users (or users who are unable or unwilling to enter inter a service contract with a TSP) will be able to buy a Toll Ticket from a website operated by S&B. Occasional users fall outside the scope of this document. The Interoperability Manager is a divided role with responsibilities split between S&B, The Danish Road Directorate (Vejdirektoratet) and the Danish Tax Ministry (Skatteministeriet).

This architecture is realised in the diagram shown below, which shows the interfaces between the various system actors. This is adapted from the system architecture diagram in ISO12855.

Data exchange will as far as is practical follow the requirements of standard prEN 16986:2023[2], which in turn is based on the underlying ISO 12855:2022 toolbox standard. These standards define the data formats for exchange between the various actors in a tolling system. It is essential that TSPs wishing to connect to the Danish KmToll system are familiar with these standards as they provide the underlying data and messaging formats to be used.

[1] In this document we will use the term TSP. TSPs are sometimes referred to as EETS Providers (EPs)
[2] Note: At the time of writing, a new version of EN 16986 is in the process of ratification. It is expected that this ratification will be concluded within the timescales of this project. However, until it is ratified, the prEN noted above will be used. Any references to EN 16986 in this document refer to the current prEN 16986:2023 version.

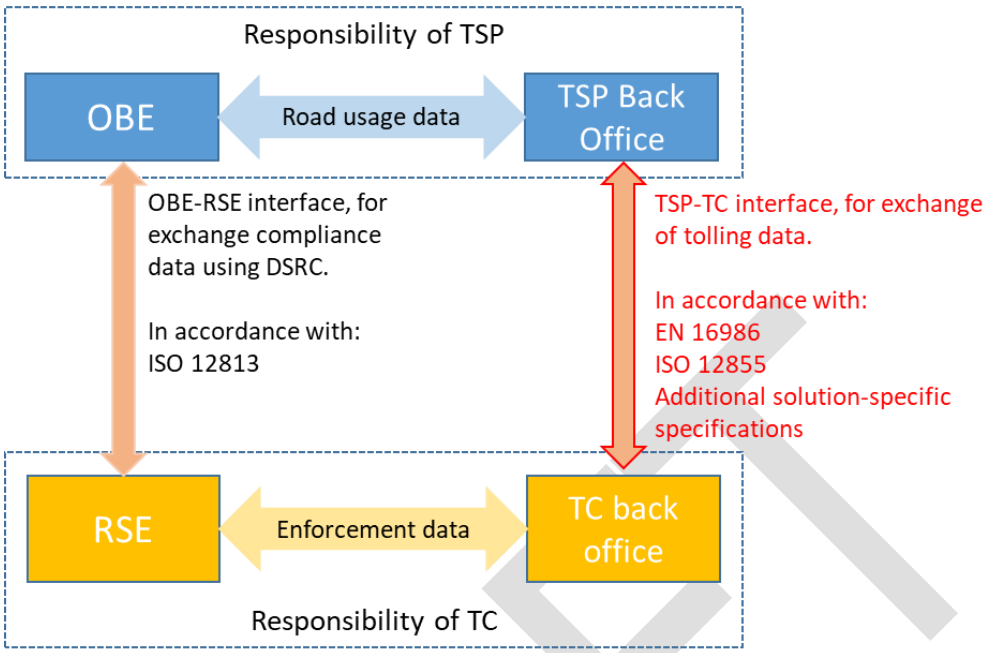


Figure 2: Interfaces in the Danish KmToll scheme

The On-Board Equipment (OBE) and TSP Back Office are the responsibility of the TSP, so the interface which carries the road usage data between them, is outside the scope of this document[3]. The interface between the TC back office and the Road Side Equipment (RSE, used for enforcement) is the responsibility of the TC, so is outside the scope of this document.

The DSRC interface between the RSE and the users' OBE must comply with standard ISO 12813:2019. As this constitutes an interface between the TSP and the TC, this interface is elaborated in this document in section 6.13.

The interface between the TC and TSP back offices (marked in red) is in scope. This interface must be compliant with the Profile standard EN 16986, which in turn uses the toolbox interface standard ISO 12855 as its base standard. While EN 16986 defines the data elements to be used, the standard is insufficiently detailed, so this specification will further elaborate on the data elements to be used, as well as any restrictions to these data elements.

While the TSP to TC interface will follow EN 16986 as closely as possible, a different transfer mechanism will be used from those described in Section 6.6.2 of the standard. The options described (web services using SOAP, or file transfer using FTP/FTPS) are no longer appropriate. Instead, a REST API supported by OpenAPI v3.1 documentation will be used for communication towards toll charger. The data exchange process is described in more detail in the following section.

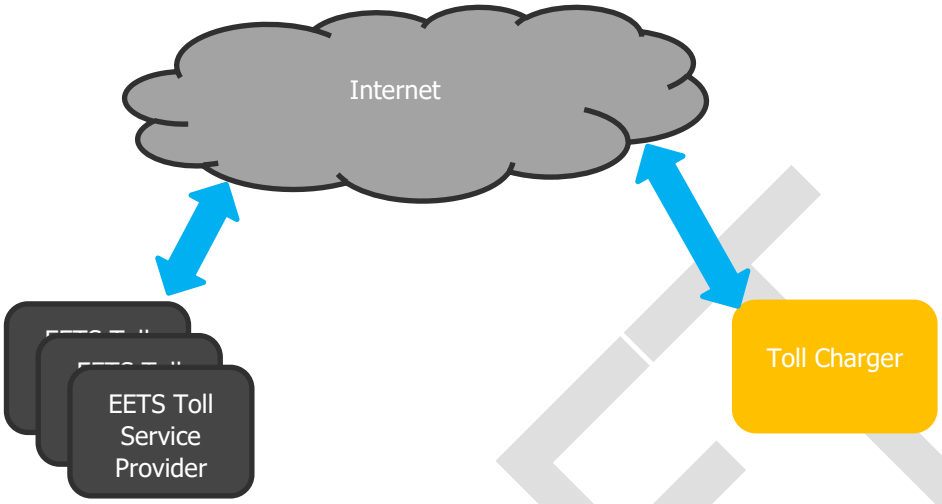
In addition to tolling services using dedicated On-Board Equipment (OBE) as envisaged by EN 16986, S&B intend to allow TSPs to offer a tolling service using non-dedicated nomadic devices as OBE, for example smartphones. For simplicity, we will refer to dedicated OBE as OBE Type 1, and nomadic devices as OBE Type 2.

As OBE Type 2cannot be assumed to include DSRC capability, additional interfaces between the TSP and TC back offices are specified to facilitate the Compliance Check Communication (CCC) functionality normally implemented on the DSRC interface. These CCC interfaces are not defined in EN 16986, and therefore additional solution-specific specifications.

[3] Note that ISO 12855 requires that this interface must be compliant with ISO 17575. However, compliance with this is the responsibility of the TCP.

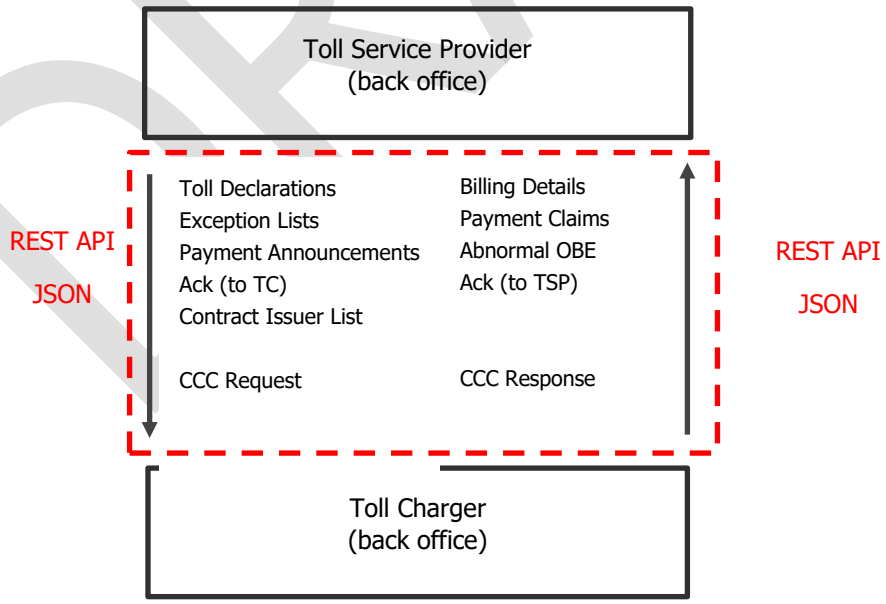
4 DATA EXCHANGE PROCESS

The transfer of operational data between TSPs and the TC will use Web Services as shown below.



As previously mentioned, data exchange will follow the requirements of standard prEN 16986:2023, which in turn is based on the underlying ISO 12855:2022 toolbox standard. These standards define the data formats for exchange between the various actors in a tolling system.

All data exchange other than Trust Object exchange will make use of RESTful web services. In a deviation from EN 16986:2022, the exchange of data between the TSP and TC will use REST APIs and JSON as shown below. Note that this deviation from the standard is limited to the Transfer Mechanism described in section 6.6 of the standard. The contents of the messages transferred over the API will be coded in JSON[4] using the ADU structures per the standard.



[4] It is recognised that previous versions of EN 16986 have required that data be coded in XML, and defined in an XSD schema definition. The current assumption in this project is that JSON is a more suitable coding for a Web interface, hence JSON coding is defined. Note however that this decision is still subject to final ratification.

The picture above illustrates, the HTTP REST APIs towards toll charger, using endpoints for exchanging data to and from Toll charger. As noted before, CCC Data Request and CCC Data Response are new message types which are not defined in EN 16986.

Data exchange will follow the requirements of standard prEN 16986:2023, which in turn is based on the underlying ISO 12855:2022 base standard. All data exchange other than Trust Object exchange are translated into REST APIs described in the OpenAPI 3.1 for sending data, and likewise for receiving data. Both are described in the JSON format.

To help app developers to integrate towards the toll charger APIs, a developer portal would be deployed shown below (Figure 3). The managed developer portal is an Azure API Management feature that allows internal or external developers and other interested parties to discover and use APIs that are published through API Management. This API management is also used to expose any of the Toll Service Providers (TSP) internal APIs as backend APIs, this ensures that all communication and authentication are handled one place. This document only describes the initial view on the APIs, and future updates and change handling of the APIs are to be found in the developer portal.

Developers need to get authenticated to see the exposed APIs available to use. This portal will handle the full life cycle of the APIs, being updates, new APIs or deprecating and API. The developer portal is using Microsoft Azure as the cloud platform.

The onboarding process document, will be send out later to disclose the full URL for the developer portal as well the processes for exposing the required APIs from the TSP described in point 3 from the Figure 4.

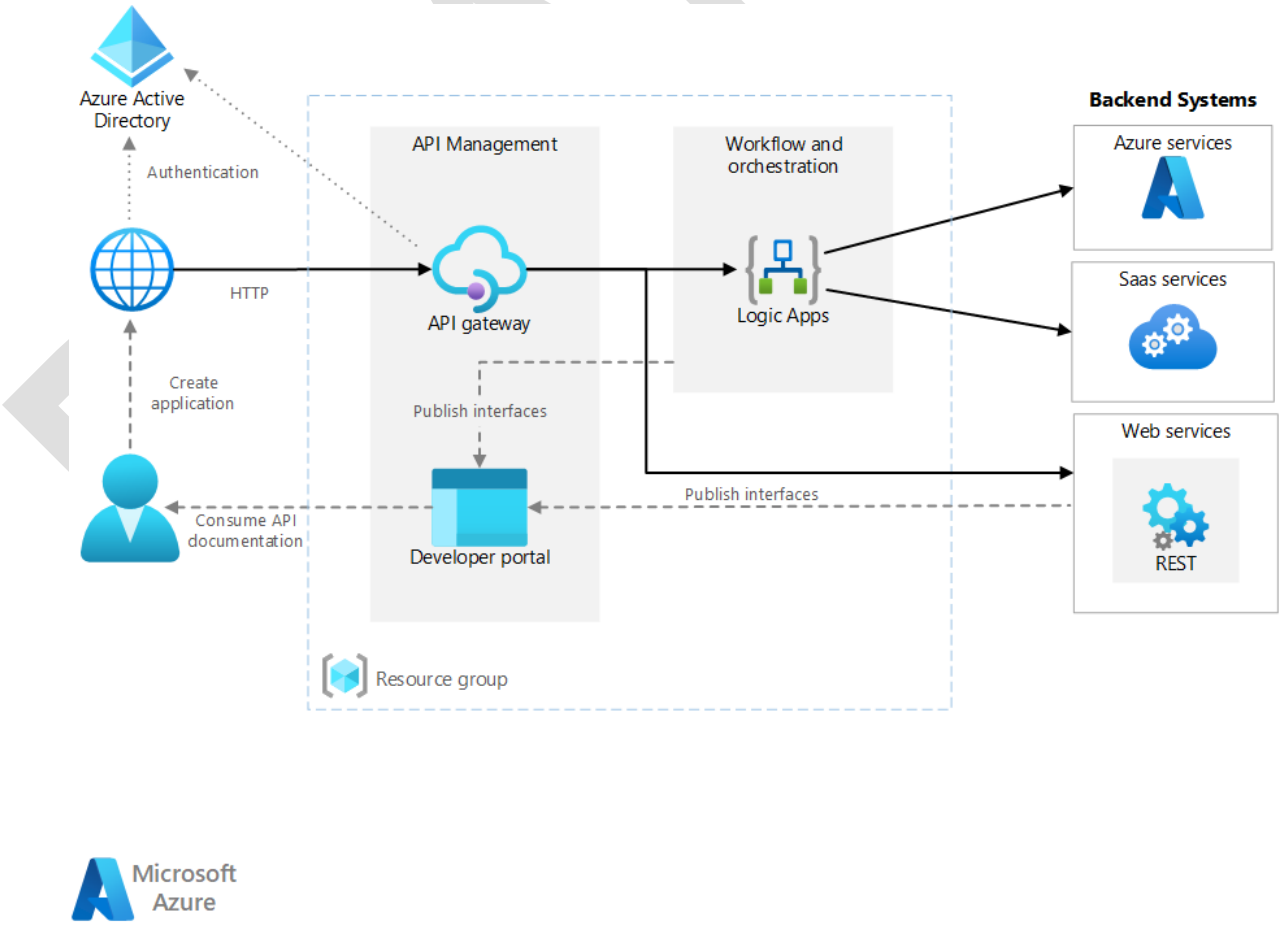


Figure 3. API Management Developer Portal

Workflow:

To describe the workflow between the TC and TSP, please see the figure (Figure 4).

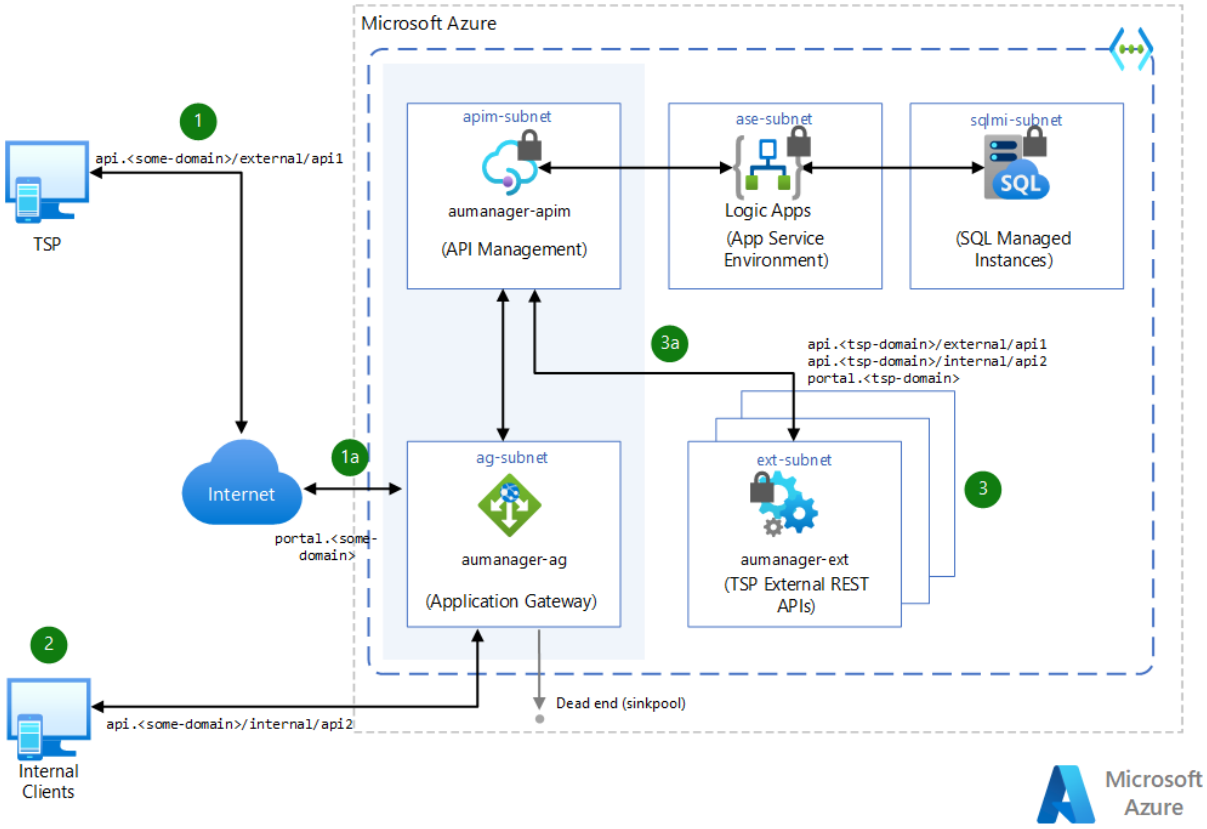


Figure 4. External APIs from TSP in APIM

1. Calls from the TSP to TC happens through the Application Gateway
2. Internal private calls within TC workloads in Azure, these are not public available
3. TSP APIs are exposed as "backend APIs", which is used for calling back to each TSPs. This ensures scalability for internal TC components, and a single point secure integration for TSPs to implement between TC and TSPs.

The API management is protected with application gateway in front, which setup the URL redirection mechanism that sends the request to the proper backend pool, depending on the URL format of the API call:

URLs formatted like `api.<some-domain>/external/*` can reach the back end to interact with the requested APIs. These are calls meant to be from TSP towards TC.

The APIs that are hosted at the TSPs are only exposed in the API management as pointers with authentication configured (no. 3 +3a in (Figure 4)). These are showed in the figure as `api.<tsp-domain>/external/api`.

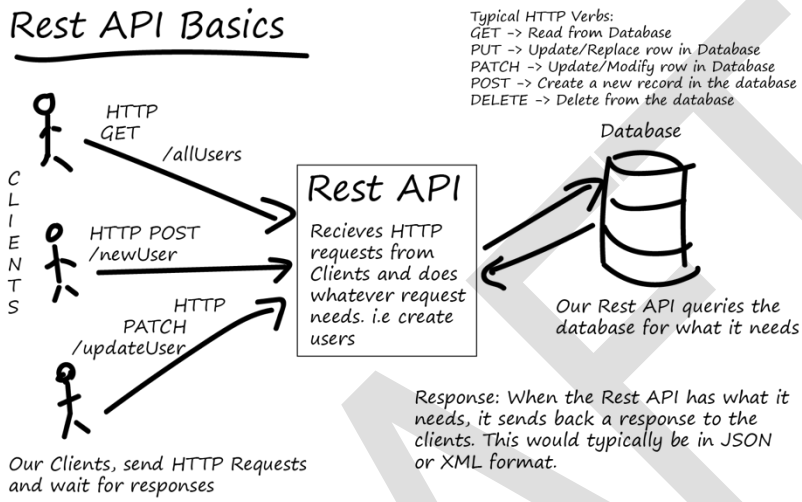
API Management also accepts and properly maps internal calls, which come from resources in the same Azure virtual network, under `api.<some-domain>/internal/*`. These calls is only intended to be used internally within TC.

Calls formatted as `api.<some-domain>/*` go to a dead end, which is a back-end pool with no target.

4.1 Message Level Protocol: REST

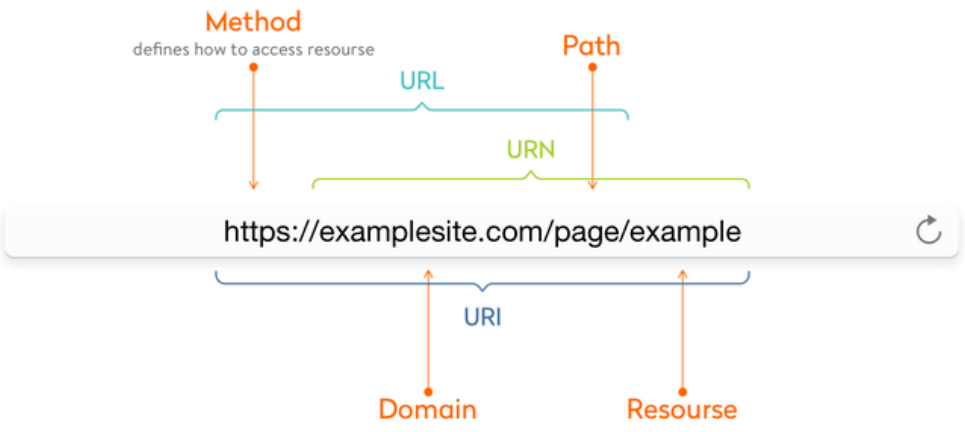
RESTful APIS have become the main standard for enabling communication between the server part of a product and its client, and it will be used for the all interfaces when communicating data towards TC and TSP.

Example:



The key principle of REST is to divide the API into logical resources. A Uniform Resource Identified, or URI, is a sequence of symbols that identifies a resource and often allows developers to access representations of that resource. The current and syntax of URIs is defined by the RFC 3986 standard.

Example of the HTTPS methods and URI:



[Components of URI]

HTTP Method	Action	Examples
GET	Obtain information about a resource	http://example.com/api/orders (retrieve order list)
GET	Obtain information about a resource	http://example.com/api/orders/123 (retrieve order #123)
POST	Create a new resource	http://example.com/api/orders (create a new order, from data provided with the request)
PUT	Update a resource	http://example.com/api/orders/123 (update order #123, from data provided with the request)
DELETE	Delete a resource	http://example.com/api/orders/123 (delete order #123)

Example of response:

The response is a JSON file and could be with an embedded list.

```
HTTP/1.1 200 OK
{
  "project_id": int,
  "name": string,
  "description": string,
  "budget": float,
  "deadline": timestamp,
  "item_list": [
      id1: int,
      id2: int,
      id3: int
    ]
}
```

Example on error on a specific parameter validation:

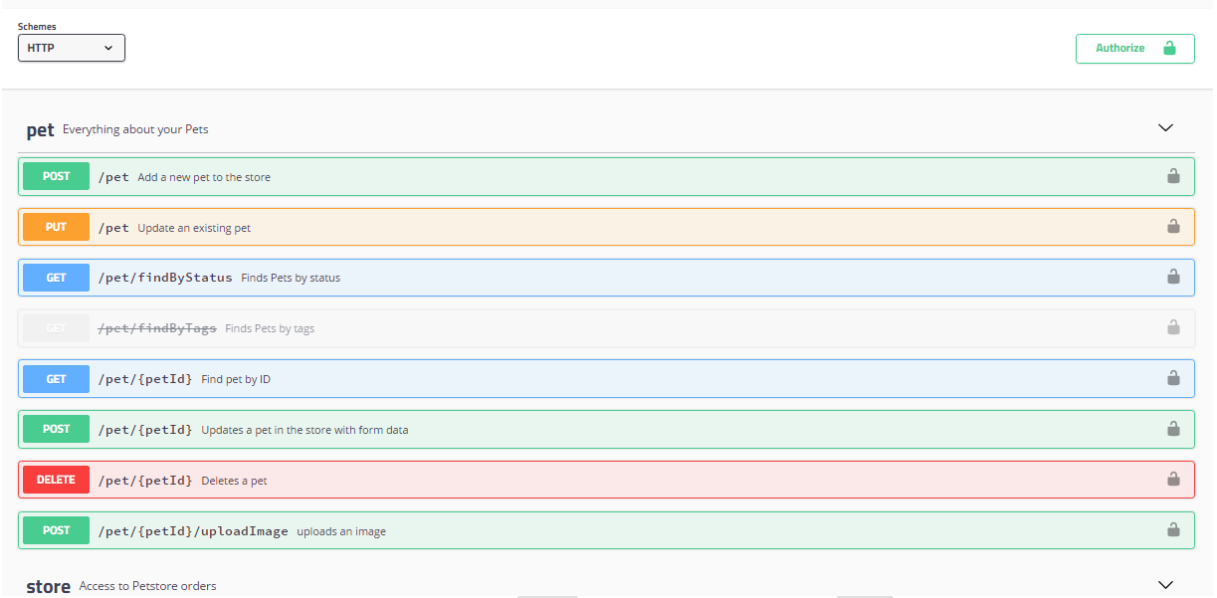
```
{
    "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
    "title": "One or more validation errors occurred.",
    "status": 400,
    "traceId": "|61c569d1-431ac3d5e5d6f164.",
    "errors": {
        "since": [
            "The value '2022-04-26T00:00:00 01:00' is not valid."
        ]
    }
}
```

OpenAPI Specification:

Each APIs in APIM will have an OpenAPI documentation as well a swagger UI in the developer portal, so the developers are able to see and test the APIs directly in the portal. Please refer to the following URL for more information:

https://swagger.io/specification/

Swagger UI example:



4.2 Message types and interfaces

Data to be exchanged will be contained in Application Protocol Data Units (APDUs) which will be defined by the ASN.1 type InfoExchange, as defined in Section 6.2 of ISO 12855:2022.

Data exchanges between the TSPs and the TC are called Transactions in EN 16986. The following EN 16986 transactions must be supported:

Transaction	Description	Typical frequency
EXCEPTIONLISTS	Allows a TSP to send exception lists to the TC. White and black lists are supported, which can be full or incremental	Daily (full list) Ad-hoc (incremental list)
TOLLDECLARATIONS_SECT	Allows a TSP to transfer toll declarations to TC in a section-based, autonomous toll scheme	Every 5 minutes
BILLINGDETAILS_TC	Allows a TC to transfer billing details to the TSP in a TC-Dominant system	Daily
PAYMENTCLAIM	Allows a TC to invoice a TSP	Monthly (TBC)
PAYMENTANNOUNCEMENT	Allows a TSP to inform the TC that a payment has been made	Monthly (TBC)
REPORTABNORMALOBE	Allows a TC to report to a TSP that it has detected abnormal behaviour in an OBE registered to the TSP	Ad-hoc
CONTRACTISSUERLIST	Allows a TSP to send contractual information regarding OBE to the TC	Ad-hoc
TRUSTOBJECTS*	Allows the exchange of trust objects between the TSP and the TC	Ad-hoc

* Note that the TRUSTOBJECTS transaction is expected to be implemented manually, and hence will not require a web interface.

A Transaction consists of at least two messages, typically a message containing data, and a message acknowledging the data. Each message consists of an InfoExchange Application Protocol Data Unit (APDU). An APDU is a self-contained message transferred between TSP and TC. An APDU in turn consists of Application Protocol Control Information (ACPI), one or more Application Data Units (ADUs) which contains the operational data to be transferred, and optionally authentication data. See ISO 12855:2022 for more details. The KmToll scheme will not use authentication as defined in

the InfoExchange definition as authentication will be provided by the underlying communications services.

The arrows pointing towards TC (from TSP) in the below picture are interfaces that TC needs to develop as REST APIs using the APDUs, and give access to these APIs to TSP, please refer to column "Implemented On" column in Table 1.

Messages between the TSP and TC will make use of a number of predefined interfaces. These interfaces, which are defined in this document, are:

Table 1. Interfaces to be implemented

Name	Implemented on	Description	Used in ISO 16986 Transactions:
Ack_TC	TC	Generic acknowledgement from TSP. This allows third party suppliers to validate incoming data which has already been validated at the API level. A single interface will be implemented at the TC to receive Acks from TSPs. The Acks will be used for multiple transactions.	BILLINGDETAILS_TC PAYMENTCLAIM REPORTABNORMALOBE
Ack_TSP	TSP	Generic acknowledgement from TC A single interface will be implemented at the TSP to receive Acks from TCs. The Acks will be used for multiple transactions.	TOLLDECLARATIONS_SECT EXCEPTIONLIST PAYMENTANNOUNCEMENT
Toll_Declarations	TC	Allows TSP to send road usage data in the form of Toll Declarations to the TC	TOLLDECLARATIONS_SECT
Billing_Details	TSP	Allows the TC to send Billing Details to the TSP	BILLINGDETAILS_TC
Exception_Lists	TC	Allows the TSP to send white and black lists to the TC	EXCEPTIONLIST
Payment_Claim	TSP	Allows the TC to send a Payment Claim (invoice) to the TSP	PAYMENTCLAIM
Payment_Announcement	TC	Allows the TSP to announce to the TC that an invoice (payment claim) has been paid	PAYMENTANNOUNCEMENT
Report_Abnormal_OBE	TSP	Allows the TC to report to the TSP that the OBE in one of their client's vehicles is exhibiting abnormal behaviour	REPORTABNORMALOBE
Contract_Issuer_List	TC	Allows the TSP to send contractual information regarding OBE to the TC (if implemented)	CONTRACTISSUERLIST

CCC_Data_Request	TSP	Allows the TC to request CCC data from the TSP	N/A
CCC_Data_Response	TC	Allows the TSP to send a response to the CCC_Data_Request to the TC	N/A

Each interface (apart from the Trust Objects) will be implemented as a REST web service.

To allow the TC to send messages to the TSP, the TSP must therefore implement the following REST interfaces:

- Ack_TC
- Billing_Details
- Payment_Claim
- Report_Abnormal_OBE
- CCC_Data_Request

The TC will implement the following interfaces to allow the TSPs to send data to the TS:

- ACK_TSP
- Toll_Declarations
- Exception_Lists
- Payment_Announcement
- CCC_Data_Responce
- Contract_Issuer_List

4.3 Interfaces, Transactions and ADUs

It is important to understand the relationship between transaction, ADUs and interfaces. To gain a fuller understanding of these relationships, a complete understanding of the relevant standards (ISO 12855 and EN 16986) is essential. The reader is referred to these standards for further details.

4.4 Validation

All data exchanges will be subject to a two-step validation process.

At the API level, all messages are checked for conformance to the message formats defined in the relevant schema definitions. Any messages failing this validation check will be rejected at the API level. It is the responsibility of the sender to ensure the appropriate action to be taken, based on the reason for the rejection. More details on the rejection at the API level are given in section 5.7 below. Messages rejected at the APIU level will NOT be further processed, so will not result in an ACK_TC or ACK_TSP being sent.

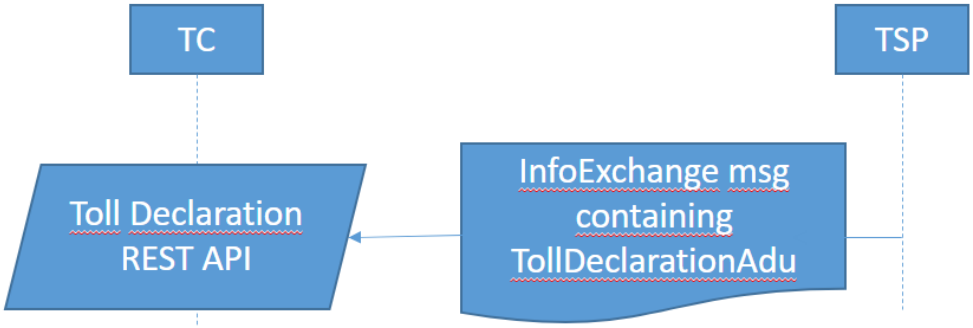
Messages validated at the API level (i.e. those resulting in a Successful Response as described in section 5.7 below) will be passed on to the back-end processes, where the contents of the message will be subject to further validation. These validation checks are more comprehensive than those at the API level. The results of these validation checks will be communicated to the sender in an ACK_TC or ACK_TSP message as appropriate. It is the responsibility of the Sender to interpret the contents of the ACK_xx message and to take the appropriate action.

4.5 Example of a TSP to TC transaction using REST

A TSP wishes to send a Toll Declaration to the TC. For this, they will use the TOLLDECLARATIONS_SECT transaction.

The data to be transmitted is coded into a TollDeclarationADU. The TollDeclarationAdu is packaged with the appropriate APCI into an InfoExchange message.

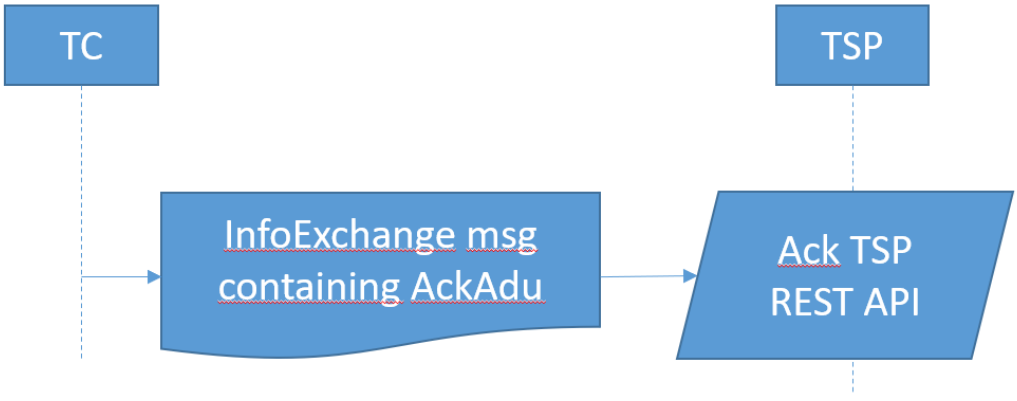
To send the InfoExchange message, the TSP connects to the REST API service interface for Toll Declarations, provided by the TC, and posts the message to the API, as shown below.



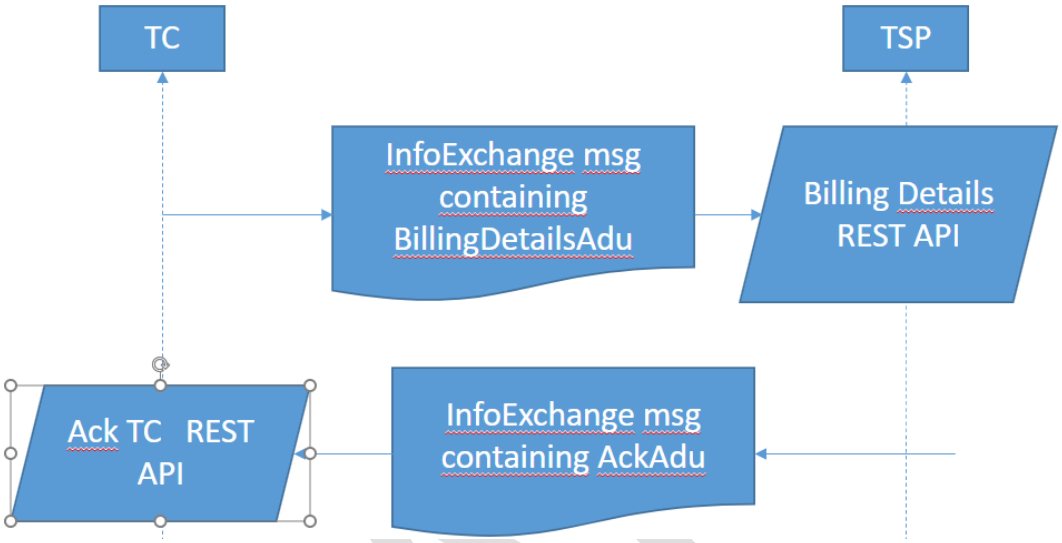
If the message is received correctly, the API will respond with the relevant acknowledgement at the API level. As described above in 4.4, this is only acknowledging that a correctly formatted message has been received; a separate Acknowledgement ADU is used to inform the TSP that the Toll Declaration, containing valid tolling data, has been received by the TC. This is described below.

The TC will process the incoming Toll Declaration, and once it is satisfied that the ADU contains valid Tolling data, it will then initiate an Acknowledge transaction with the relevant TSP.

To send the Ack, the data to be transmitted is coded into an AckAdu. The AckAdu is packaged with the appropriate APCI into an InfoExchange message. TC will make a backend REST API call to the the API service provided by TSP for TC-initiated Ack messages. The TSP will retrieve the Ack message from the TCAck API service, as shown below. Note that the AckAdu may not be required in all cases. If the validation undertaken by the REST API fulfils all validation requirements, the AckAdu will not be required. This will be further elaborated in a later version of this document.



For cases where the TC wishes to send data to the TSP, for example Billing Details, the above sequence is used, but somewhat in reverse; the TC creates the relevant Billing Details InfoExchange message to the Billing Details REST API service implemented by the TSP. Once the message has been processed, the TSP will push an Ack message to the relevant Ack_TSP REST API service, as shown below.



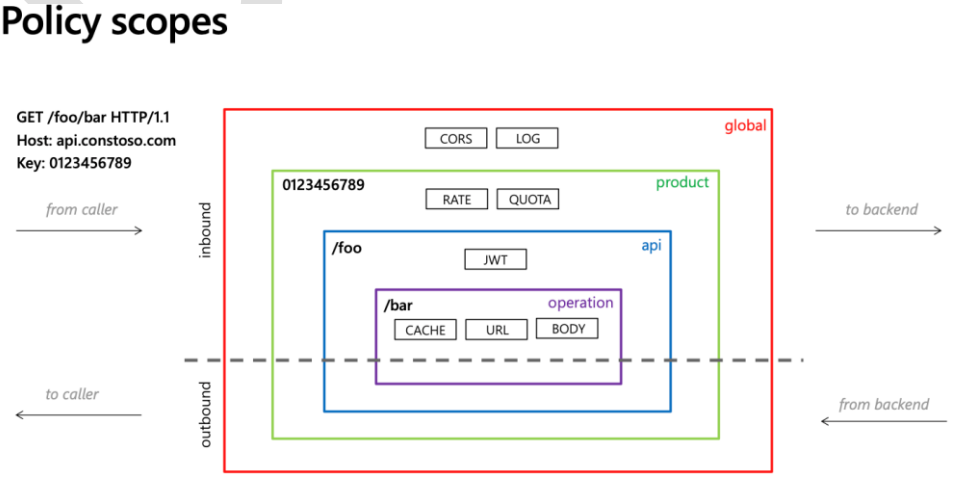
It can be seen that in all cases, the message receiver acts as the server – messages are pushed from sender to receiver.

5 COMMON ASPECTS FOR ALL INTERFACES

This chapter contains the specifications which are common to all the interfaces.

5.1 Control access

Each API will have so called "policies" applied, the scope of these policies are divided into different scopes: global, product and api, and ensures that each API will confine to the governance set by the toll charger. An example could be limitation to how many times a TSP can call each API within a specified time (throttling), and rerouting policies, as well the logging mechanism.



Some of the common polices are listed below and more details on what policies are enforced will be provided on request and if needed:

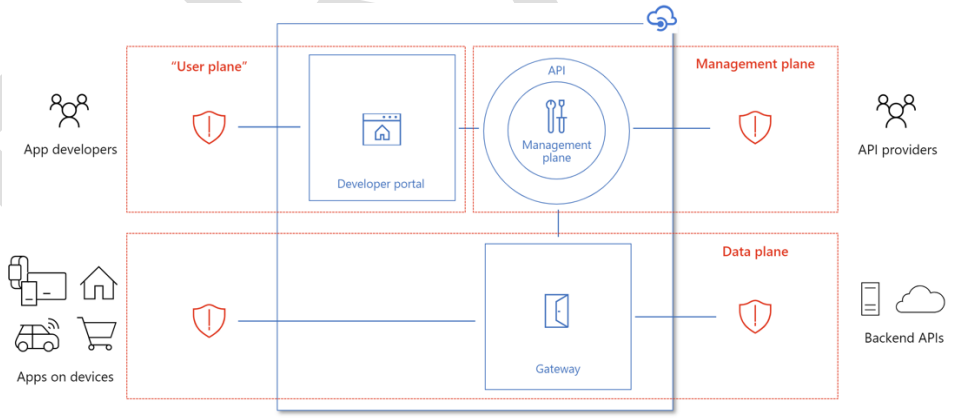
Access restriction	Transformation	Advanced	Dapr integration
• Check HTTP header • Limit call rate by subscription • Limit call rate by key • Restrict caller Ips • Set usage quota by subscription • Set usage quota by key • Validate client certificate • Validate JWT	• Convert JSON to XML • Convert XML to JSON • Find and replace string in body • Mask URLs in content • Set backend service • Set body • Set HTTP header • Set query string parameter • Rewrite URL • Transform XML using XSLT	• Send one way request • Send request • Set HTTP proxy • Set variable • Set request method • Set status code • Control flow • Emit metric • Log to Event Hub • Trace • Mock response • Forward request • Limit concurrency • Return response • Retry • Wait	• Send request to a service • Send message to a pub/sub topic • Trigger output binding
Authentication	**Caching**	**Cross Domain**	**Validation policies**
• Authenticate with basic • Authenticate with client certificate • Authenticate with managed identity	• Get from cache • Store to cache • Get value from cache • Store value from cache • Remove value from cache	• Allow cross-domain calls • CORS • JSONP	• Validate content • Validate parameters • Validate headers • Validate status code • Validate GraphQL request

5.2 Security & Authentication for REST APIs

The API endpoints will be secured using HTTPS, this will protect authentication credentials like API keys, passwords and JSON Web Tokens (JWT). The HTTP will also include a timestamp in the request, so the server can compare the request timestamp and accept the request only within a certain timeframe, for instance one or two minutes. This eliminates cases of replay attacks from hackers trying to brute force the system.

The following diagram is a conceptual view of Azure API Management, showing the management plane (Azure control plane), API gateway (data plane), and developer portal (user plane), each with at least one option to secure interaction.

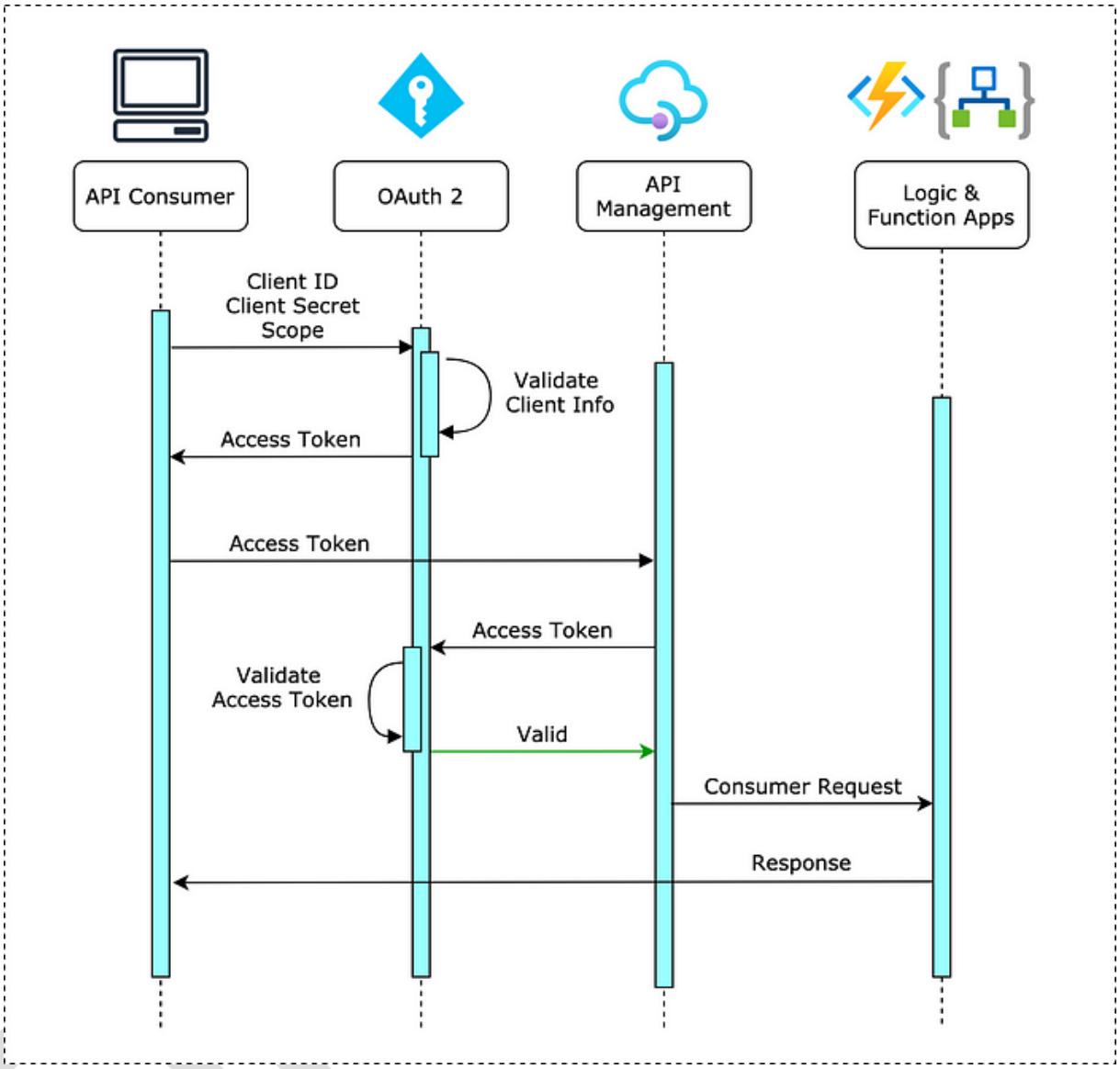
The scope of this document is to cover the "User plane" accessing the Developer portal and the "Data plane" calling the APIs through the gateway as TSP provider.



5.3 Data plane access

To be able to use the TC provided APIs, APIM is having the responsibility to do the initial authentication to prevent unauthorized access. This is done using the using the standardized OAuth 2.0 flow, API Management can retrieve and refresh access tokens to be used inside of API management or sent back to a client. OAUTH 2.0 is the open standard for access delegation which provides client a secure delegated access to the resources on behalf of the resource owner.

Below shows the process flow for using the OAuth 2.0 flow for accessing the APIs provided by TC:



Securing TC provided APIs as "Backend APIs":

To let the APIM expose the TSP provided APIs as "backend APIs", its recommended that TSP is minimum securing the API with the same OAuth flow as above, where with ClientID and Client Secret are shared to TC to call the APIs. These keys will be kept in the TCs Azure Key Vault and will only be used when calling the "backend API".

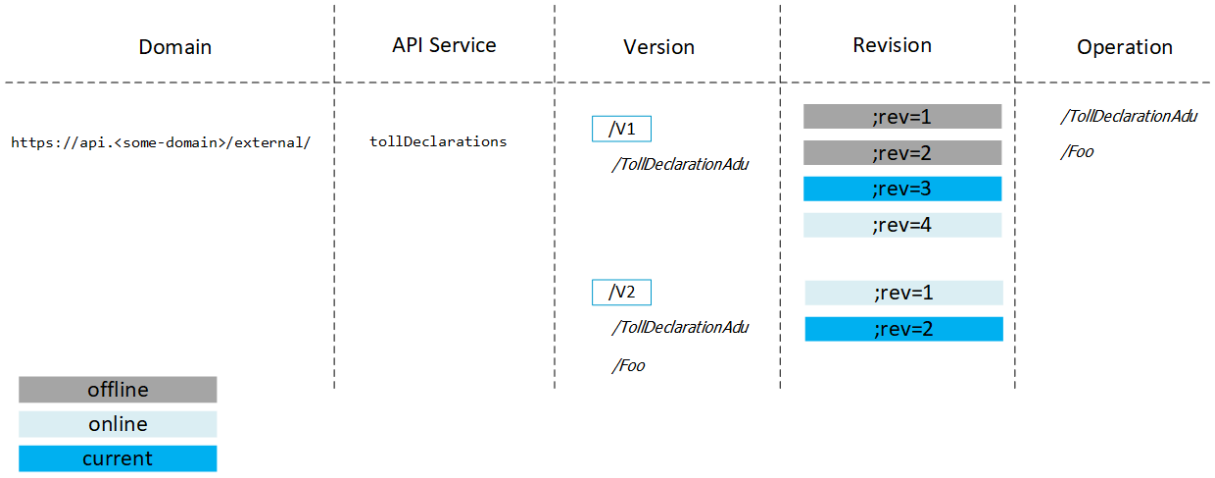
5.4 User plane access – Developer portal

The URL to the developer portal together with the authentication information and onboarding process will be provided as part of the onboarding.

5.5 Versioning

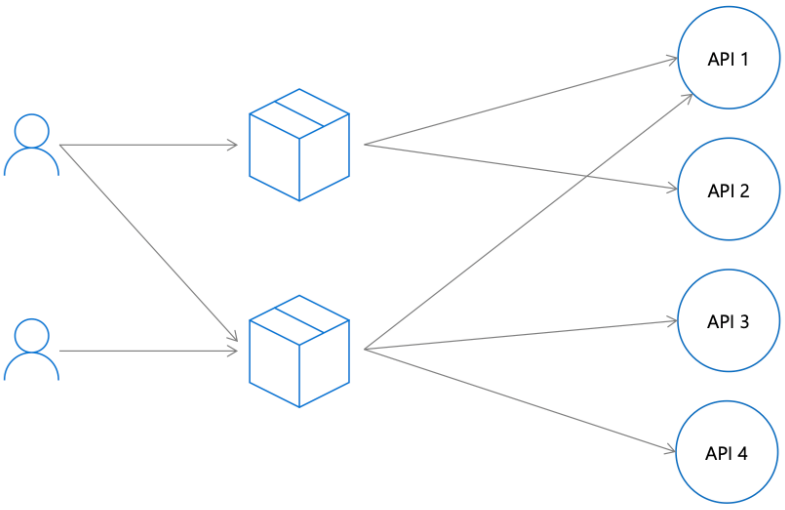
Each APIs would go through different versions as a natural way to keep them relevant and up-to-date when modification to existing API or new exchange data between TC and TSP are identified. Below table is showcasing an example how each API URI is represented when changing versions,

revisions. The table is also showing that some apis and revision can be sunset (offline) when they are no longer maintained.



5.6 API products

When logging into the developer portal, the APIs would be divided into logical products. These products is only an abstraction layer and a TSP developer will only see the products that are relevant. APIM is used for API discovery.



5.7 Error/validation handling

Basic HTTP response status codes will indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes and returned for as response for a specific API call on the message level, while a specific cause for any error or validation are to be found in the body of the returned call (please see the links):

Informational responses (100 – 199) – Server acknowledges a request

Successful responses (200 – 299) - Server completed the request as expected

Redirection messages (300 – 399)- Client needs to perform further actions to complete the request

Client error responses (400 – 499) - Client sent an invalid request

Server error responses (500 – 599) - Server failed to fulfill a valid request due to an error with server

The REST API error and validation handling, uses **the IETF devised** RFC 7807, **which creates a generalized error-handling schema**.

This schema is composed of five parts:

1. *type* – a URI identifier that categorizes the error
2. *title* – a brief, human-readable message about the error
3. *status* – the HTTP response code (optional)
4. *detail* – a human-readable explanation of the error
5. *instance* – a URI that identifies the specific occurrence of the error

The body could look like this:

```
{
    "type": "/errors/incorrect-user-pass",
    "title": "Incorrect username or password.",
    "status": 401,
    "detail": "Authentication failed due to incorrect username or password.",
    "instance": "/login/log/abc123"
}
```

Note that the *type* field categorizes the type of error, while *instance* identifies a specific occurrence of the error in a similar fashion to classes and objects, respectively.

These fields provide a client or developer with information to help troubleshoot the problem and also constitute a few of the fields that make up standard error handling mechanisms.

5.8 Service levels

This will be described in a later version of this document (if appropriate)

6 INDIVIDUAL INTERFACE DESCRIPTIONS

Each separate interface is defined below. These are called Transactions in the 16986 standard.

The interfaces described below are defined in prEN 16986:2022, which in turn uses the underlying standard ISO 12855:2022. While the standards define the interfaces, they contain optional elements which need to be defined, and that is the purpose of this interface specification.

Interfaces described in sections 6.1 to 6.5 are to be implemented at the TSP.

Interfaces described in 6.6 to 6.11 will be implemented at the TC.

6.1 Ack_TSP Interface

6.1.1 Purpose of the Interface

The Ack_TSP interface is implemented by the TSP and is used by the TC to acknowledge the correct receipt of message from the TSP. It is never used in isolation, and should be considered as a part of the messaging protocol.

Acknowledgements can be synchronous or asynchronous, depending on the message type being acknowledged.

In this version of the Specification, the Ack_TSP interface may only be used as part of the following transactions:

- TOLLDECLARATIONS_SECT
- EXCEPTIONLIST
- PAYMENTANNOUNCEMENT

In future versions, it may be used in other transactions.

6.1.2 Communications Processes

The Acknowledgement will be transferred using a REST Web-Service interface as defined in section 4.

The following additional specifications apply to this interface:

1. The Acknowledgement is used to Acknowledge (positively or negatively) the messages containing other ADUs. It is never used in isolation.

6.1.3 Message Formats

The Ack_TC data is transmitted in an *InfoExchange message* with an *AckAdu* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.1.4 Error Handling

API errors will be handled as described in section 5.7.

6.2 Billing_Details interface

6.2.1 Purpose of the Interface

The interface is used by the TC to issue regular Billing Details to the TSP. The Billing Details provides the TSP with details of all Tolls incurred by the TSP's customer during the billing period covered by the Billing Details.

6.2.2 Communications Processes

The Billing Details will be transferred using a REST Web-Service interface as defined in section 4, and will be Acknowledged (see 6.6).

The *AckAdu* is used to acknowledge the *BillingDetailsADU.*

The following additional specifications apply to this interface:

1. The TC will issue Billing Details once per billing period, currently assumed to be once every 24 hours. This time is contractually defined.

6.2.3 Message Formats

The Billing Details is transmitted in an *InfoExchange message* with a *BillingDetailsADU* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.2.4 Error Handling

API errors will be handled as described in section 5.7.

6.3 Payment_Claim interface

6.3.1 Purpose of the Interface

The interface will be used by the TC to issue Payment Claims to a TSP.

6.3.2 Communications Processes

The Payment Claim will be transferred using a REST Web-Service interface as defined in section 4, and will be Acknowledged (see 6.6).

The *AckAdu* is used to acknowledge the *PaymentClaimADU.*

The following additional specifications apply to this interface:

1. Payment Claims will be pushed to the TSP at contractually agreed intervals.

6.3.3 Message Formats

The Payment Claim is transmitted in an *InfoExchange message* with a *PaymentClaimADU* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.3.4 Error Handling

API errors will be handled as described in section 5.7.

6.4 Report_Abnormal_OBE interface

6.4.1 Purpose of the Interface

The interface allows the TC to report that the OBE in one of the TSP's customer vehicles is behaving abnormally.

6.4.2 Communications Processes

The Report Abnormal OBE will be transferred using a REST Web-Service interface as defined in section 4, and will be Acknowledged (see 6.6).

The *AckAdu* (see 6.6) is used to acknowledge the *ReportAbnormalObeAdu.*

The following additional specifications apply to this interface:

1. None identified.

6.4.3 Message Formats

The Report Abnormal OBE data is transmitted in an *InfoExchange message* with a *ReportAbnormalObeAdu* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.4.4 Error Handling

API errors will be handled as described in section 5.7.

6.5 CCC_Data_Request interface

6.5.1 Purpose of the Interface

The interface will be used by the TC to request Compliance Check data from the TSP. The TSP will be expected to reply with a message containing Compliance Check data to the CCC_Data_Response interface described below. The Compliance Check data will be similar to the CCC data transmitted from the RSE to the OBE in the conventional DSRC-based CCC transaction.

Note that only TSPs which intend to offer an OBE Type 2 to their customers will be required to implement this interface.

6.5.2 Communications Processes

The CCC_Data_Request will be transferred using a REST Web-Service interface as defined in section 4. It will not be explicitly acknowledged as the response from the TSP will constitute an implicit Acknowledgement.

Note that the *CCCDataRequestADU* is not defined in prEN 16986:2023, but has been specifically created for this application. It is based on the *RequestAdu* structure defined in ISO 12855:2022

The *CCCDataResponseADU* is used to implicitly acknowledge the *CCCDataRequestADU.*

6.5.3 Message Formats

The Payment Claim is transmitted in an *InfoExchange message* with a *CCCDataRequestADU* as defined in this document and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal.

6.5.4 Error Handling

API errors will be handled as described in section 5.7.

6.6 Ack_TC Interface

6.6.1 Purpose of the Interface

The Ack_TSP interface is implemented by the TC and is used by the TSP to acknowledge the correct receipt of message by the TSP. It is never used in isolation, and should be considered as a part of the messaging protocol.

Acknowledgements can be synchronous or asynchronous, depending on the message type being acknowledged.

In this version of the Specification, the Ack_TSP interface may only be used as part of the following transactions:

- BILLINGDETAILS_TC
- PAYMENTCLAIM
- REPORTABNORMALOBE

In future versions, it may be used in other transactions.

6.6.2 Communications Processes

The Acknowledgement will be transferred using a REST Web-Service interface as defined in section 4.

6.6.3 Message Formats

The Ack_TSP data is transmitted in an *InfoExchange message* with an *AckAdu* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.6.4 Error Handling

API errors will be handled as described in section 5.7.

6.7 Toll_Declarations interface

6.7.1 Purpose of the Interface

The Toll Declarations interface is to be used by the EETS Provider for the transmission of Toll Declarations generated by EETS on-board equipment from the EETS Provider's central system to the Toll Charger.

The Toll Declaration contains the GNSS position data generated by the On-Board Equipment (OBE) of the EETS Provider, which is required by the Toll Charger for the calculation of tolls. The report also contains toll-relevant data about the vehicle, for example maximum permissible train weight, vehicle class, emissions class etc.

6.7.2 Communications Processes

The Road Usage Report will be transferred using a REST Web-Service interface as defined in section 4, and will be Acknowledged (see 6.6).

Each InfoExchange message may only contain exactly one *TollDeclarationADU* and thus exactly one *ChargeReport.*

The *AckAdu* (see 6.6) is used to acknowledge the *TollDeclarationADU.*

The following additional specifications apply to this interface:

1. None identified.

6.7.3 Message Formats

The Road Usage data is transmitted in an *InfoExchange message* with a *TollDeclarationAdu* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.7.4 Error Handling

API errors will be handled as described in section 5.7.

6.8 Exception_Lists interface

6.8.1 Purpose of the Interface

The interface I used by the TSP to transfer exception lists to the TC. Exception lists can be of four types:

- Full White List
- Incremental White List
- Full Black List
- Incremental Black List

6.8.2 Communications Processes

The Exception Lists will be transferred using a REST Web-Service interface as defined in section 4, and will be Acknowledged (see 6.6).

The *AckAdu* is used to acknowledge the *ExceptionListAdu.*

The following additional specifications apply to this interface:

1. Full Exception Lists will be transferred at regular intervals, as contractually defined.
2. Incremental Exception Lists will be transferred as required

6.8.3 Message Formats

The Exception Lists are transmitted in an *InfoExchange message* with a *ExceptionListAdu* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.8.4 Error Handling

API errors will be handled as described in section 5.7.

6.9 Payment_Announcement interface

6.9.1 Purpose of the Interface

The interface is used by the TSP to inform the TC that a payment has been made to the TC. This payment will normally be made in response to a Payment Claim (see 6.3)

6.9.2 Communications Processes

The Payment Announcement will be transferred using a REST Web-Service interface as defined in section 4, and will be Acknowledged (see 6.6).

The *AckAdu* is used to acknowledge the *PaymentAnnouncementAdu.*

The following additional specifications apply to this interface:

1. The TSP shall issue a Payment Announcement within an agreed period after making a payment to the TC. This time is contractually defined.

6.9.3 Message Formats

The Payment Announcement data is transmitted in an *InfoExchange message* with a *PaymentAnnouncementAdu* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.9.4 Error Handling

API errors will be handled as described in section 5.7.

6.10 Contract_Issuer_List interface

6.10.1 Purpose of the Interface

The interface is used by a TSP to provide a TC with contractual information about their issued OBE.

6.10.2 Communications Processes

The Contract Issuer List will be transferred using a REST Web-Service interface as defined in section 4, and will be Acknowledged (see 6.6).

The following additional specifications apply to this interface:

None defined at this stage

6.10.3 Message Formats

The Contract Issuer List data is transmitted in an *InfoExchange message* with a *ContractIssuerListAdu* in accordance with standard ISO/DIS 12855:2022 and this must in principle satisfy the restrictions described in the SectionAutonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal. This will also contain any restrictions or changes to the formal message format definitions in prEN 16986:2023.

6.10.4 Error Handling

API errors will be handled as described in section 5.7.

6.11 CCC_Data_Response interface

6.11.1 Purpose of the Interface

The interface will be used by the TSP to respond to a Compliance Check data request from the TSP. The Compliance Check data will be similar to the CCC data transmitted from the RSE to the OBE in the conventional DSRC-based CCC transaction.

Note that only TSPs which intend to offer an OBE Type 2 to their customers will be required to support this interface.

6.11.2 Communications Processes

The CCC_Data_Response will be transferred using a REST Web-Service interface as defined in section 4. It will not be explicitly acknowledged.

Note that the *CCCDataResponseADU* is not defined in prEN 16986:2023, but has been specifically created for this application. It is based on the *ReportCccEventAdu* structure defined in ISO 12855:2022

6.11.3 Message Formats

The Payment Claim is transmitted in an *InfoExchange message* with a *CCCDataResponseADU* as defined in this document and this must in principle satisfy the restrictions described in the Section-Autonomous profile (with TC dominance) of specification prEN 16986:2023.

Formal message and data formats can be downloaded from the API Management Developer Portal.

6.11.4 Error Handling

API errors will be handled as described in section 5.7.

6.12 Trust Objects (TRUSTOBJECT transaction)

Trust objects will not be transmitted over a REST interface. The transfer mechanism will be bilaterally agreed between the TC and each TS. As these transfer are expected to be infrequent, a manual exchange mechanism (e.g. email) is envisaged.

6.13 Compliance Check Communication (CCC) transaction (DSRC)

6.13.1 Purpose of the Interface

The interface is used to enable compliance checking of the OBE of passing vehicles passing a road-side enforcement (RSE) point by allowing the RSE to directly interrogate the OBE using DSRC.

6.13.2 Communications Processes

The communications between the RSE and the OBE will use Dedicated Short Range Communications (DSRC) according to the ISO 12813:2019 and associated standard.

The following additional specifications apply to this interface:

1. None identified.

6.13.3 Message Formats

Message formats will be as defined in ISO 12813:2019.

6.13.4 Error Handling

As defined in ISO 12813:2019 and normative references therein.

7 PRODUCTION VS. EDU ENVIRONMENTS

This will be described in a later version of this document