Annex F (Interface Specifications)

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

Version: 1.1

Date: 1 February 2024

TABLE OF CONTENTS

1 DOCUMENT HISTORY 3
2 DEFINITIONS AND ABBREVIATIONS 3
3 INTRODUCTION 4
4 DATA EXCHANGE PROCESS 6
5 COMMON ASPECTS FOR ALL INTERFACES 18
6 INDIVIDUAL INTERFACE DESCRIPTIONS 24
7 ENVIRONMENTS & DATA EXPECTATION 48

1 Document history

Date of first appearance of this entry into the register	1 February 2024
Last update	1 February 2024
Next review	Fourth quarter 2024

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)
EDU	Education environment, a training sandbox environment for testing APIs
EETS	European Electronic Tolling Service
EFC	Electronic Fee Collection
FQDN	Fully Qualified Domain Name
HTTPS	Hypertext Transfer Protocol Secure
ISO	International Organisation for Standardisation
JSON	JavaScript Object Notation
KmToll	Kilometer Tolling Scheme in Denmark
OBE	On Board Equipment
REST	Representational State Transfer
TC	Toll Charger
TSP	Toll Service Provider. Note this this is sometimes referred to as an EETS Provider
UAT	User Acceptance Test, this is either referred to the environment where the UAT is conducted or the actual test

3 Introduction

This Annex describes the interfaces between the EETS Provider[1] (TSP) and Toll Charger (TC) in the Danish Kilometer 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. 3.0 (OpenAPI Specification v3.0.0) using REST API. This will help reducing the maintenance of this documentation when APIs are going through updates (versioning).

The architecture of the KmToll Scheme is compliant 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:

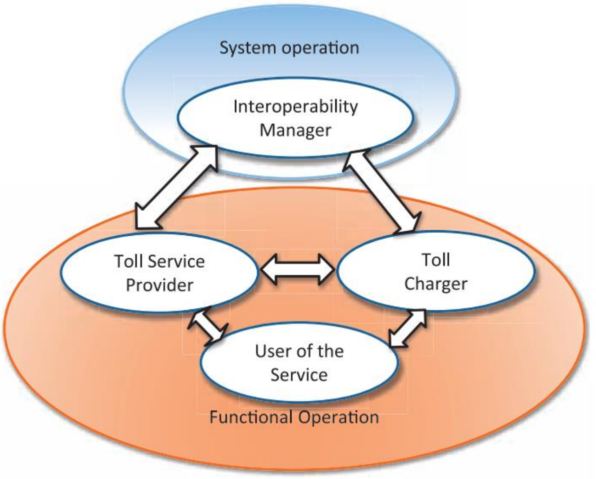


Figure 1. Roles in an EFC scheme as defined in DS ISO 17573-1

Sund & Bælt will be the Toll Charger(TC). Regular users will buy EETS from a TSP which are accredited onto the KmToll Scheme. Occasional users (or users who are unable or unwilling to enter into a service contract with a TSP) will be able to buy a toll ticket from a website operated by the Toll Charger. Occasional users fall outside the scope of this document. The Interoperability Manager is a divided role with responsibilities split between the TC, The Danish Road Directorate ("Vejdirektoratet") and the Danish Ministry of Taxation ("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 EN 16986, 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 a TSP wishing to connect to the Danish KmToll Scheme is familiar with these standards as they provide the underlying data and messaging formats to be used.

At the time of writing, the current ratified version of EN 16986 is EN 16986:2016. However, this version is outdated and does not adequately address the requirements of a Toll Charger dominant GNSS-based EETS compliant system, hence a new version of EN 16986, currently identified as preliminary version prEN 16986:2023 is in the process of ratification. It is expected that this ratification will be completed within the timescales of this project. However, it is noted that the updates

[1] In this document we will use the term TSP. A TSP are sometimes referred to as an EETS Provider (EP).

included in prEN 16986:2023 do not adequately meet the requirements of the KmToll Scheme, and the defined interfaces in this document introduce further changes to the standard. As this is no longer a fully compliant implementation of EN 16986, the *aidIdentifier* attribute on all TSP to TC the specifications will be changed in the future contain the value of 32, indicating a private version of the communications protocol, rather than the value of 17 which has been reserved for EN 16986:2023. At the present time, TSPs should continue to use the value of 17 for *aidIdentifier* until informed otherwise.

For the purposes of this Annex, all references to EN 16986 refer to prEN 16986:2023, unless specifically identified differently.

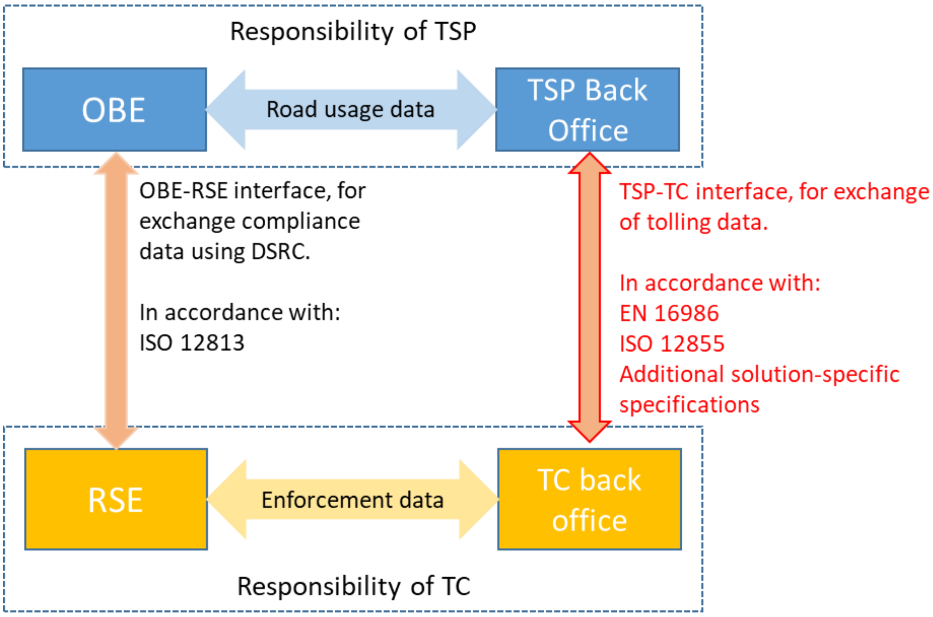


Figure 2. Interfaces in the Danish KmToll scheme

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[2]. The interface between the TC Back Office and the Roadside 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 EETS User's 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 6.14.

The interface between the TC and TSP Back Office (marked in red) is in scope. These interfaces 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 with a JSON payload supported by OpenAPI v3.0 documentation will be used

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

for communication between TC and the TSP. 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, TC intend to allow the TSP 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 2 cannot 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, therefore additional solution-specific specifications are defined in this specification.

4 Data exchange process

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

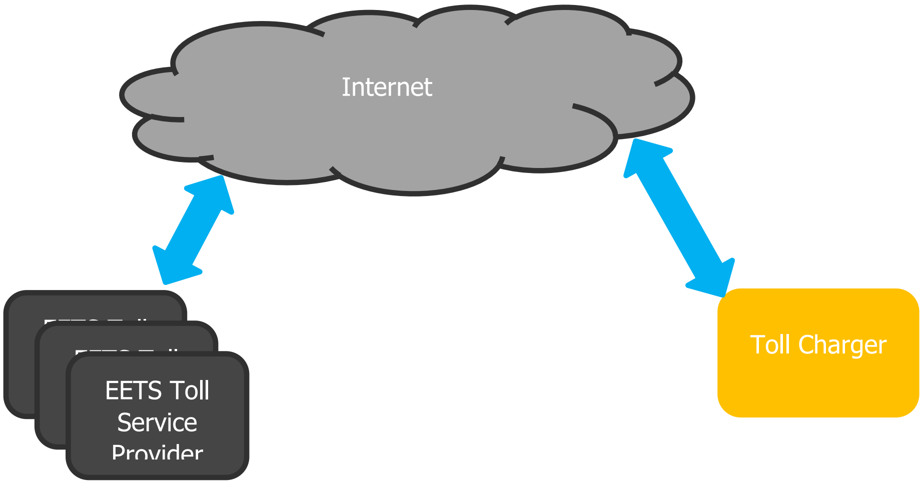


Figure 3. Connections between TSP and TC

As previously mentioned, data exchange will follow the requirements of standard EN 16986, 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 exchanges other than Trust Object and Actor Table exchanges will make use of RESTful web services. In a deviation from EN 16986, the exchange of data between the TSP and TC will use REST APIs and JSON payload 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[3] using the ADU structures per the standard, with additional data restrictions where required.

[3] It is recognised that previous versions of EN 16986 have required that data be coded in XM and defined in an XSD schema definition. For this project it has been decided that JSON is a more suitable coding for a RESTful Web interface, hence JSON coding is defined.

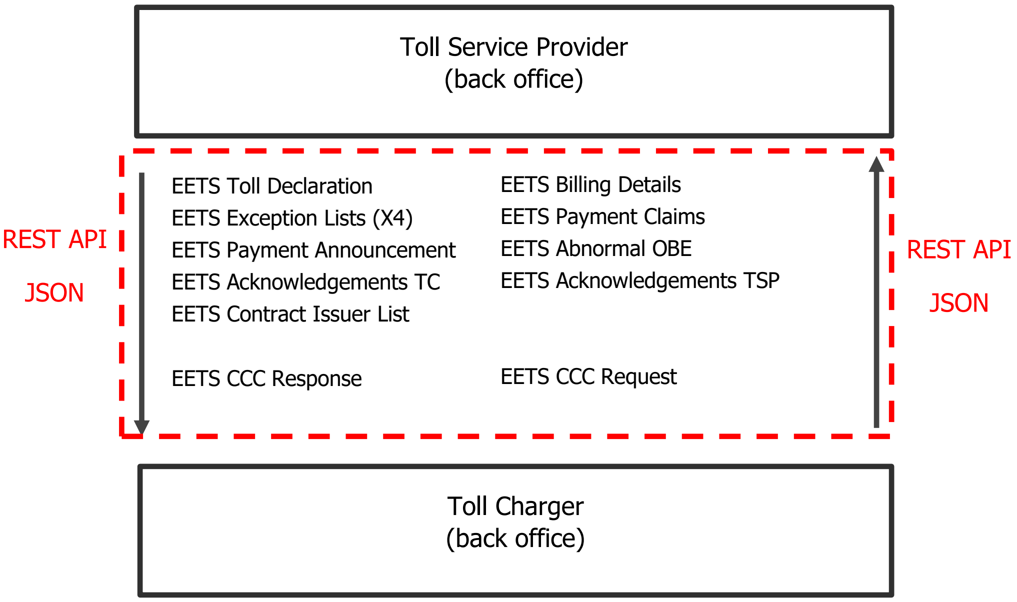


Figure 4. TSP to TC interfaces

The picture above illustrates, the HTTPS REST APIs towards the TC, using endpoints for exchanging data to and from the TC. 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 EN 16986, which in turn is based on the underlying ISO 12855:2022 base standard. All data exchanges other than Trust Object and Actor Table exchange are translated into REST APIs described in the OpenAPI 3.0 for sending data, and likewise for receiving data. Both are described in the JSON format.

To help app developers to integrate towards the TC's APIs, a Developer Portal will be deployed shown below (Figure 5). The managed Developer Portal is a feature in Azure API Management that allows internal or external developers to see specifications and try APIs that are published though a product (group of API's) on the Developer Portal. The TC will use the API Management to also expose any of the TSP's internal APIs as backend APIs, this ensures that all communication and authentication are handled in one place. Therefore, the TSP will need to develop their own APIs on the OpenAPI 3.0 specification and JSON payload defined by the TC. 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 and authorised by the TC to see the exposed API products available to the user. The Developer Portal will handle the full life cycle of the TC's APIs, being updates, new APIs or deprecating an API. Likewise, any updates to the TSP's APIs will need to be updated and re-configured within the TC API Management, as this will act as a client when calling the TSP's own APIs.

A document will be distributed to the TSP as part of the Accreditation Procedure 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 6.

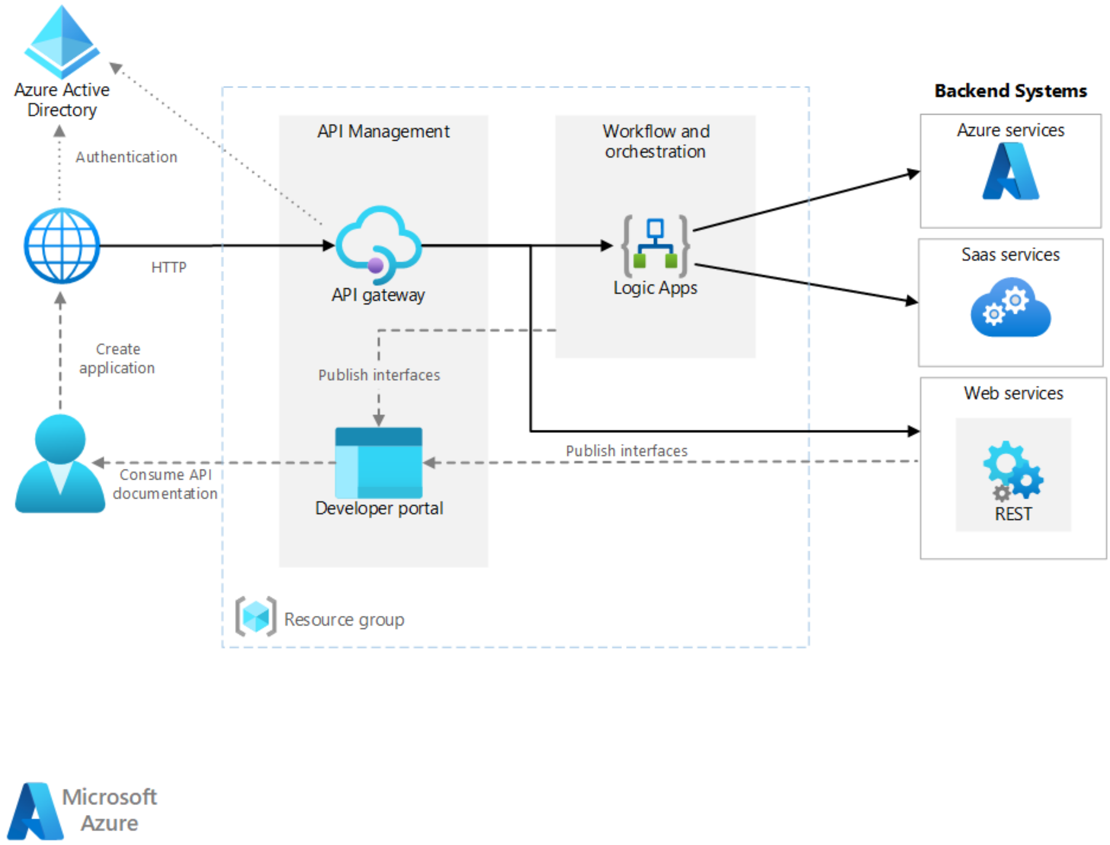


***Figure 5**. API Management Developer Portal*

Workflow:

To describe the workflow between the TC and TSP, please see Figure 6. The TC will have a sandbox for reading OpenAPI documentation for each API in the Developer Portal, which only will be available in the two environments (education environment and UAT). Please see section 7 for further information about the environments. The flow below only describes the actual API call and how it is routed and handled with the TC in the production environment.

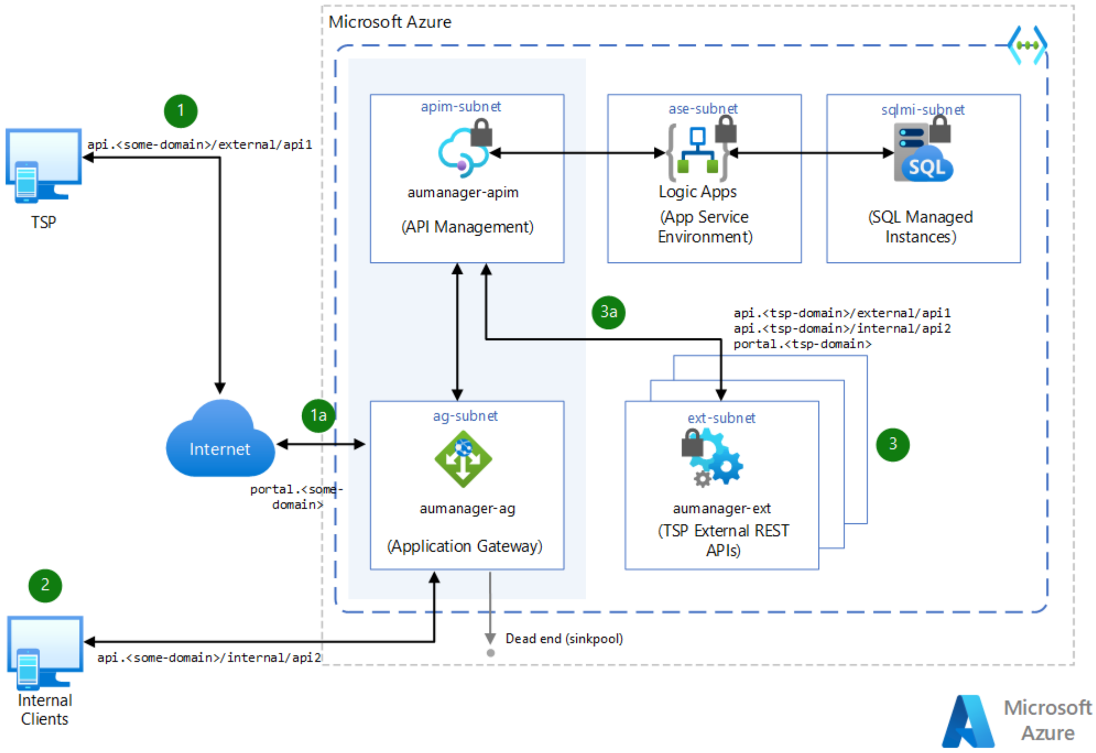


Figure 6. External APIs from TSP in APIM

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. Example of the TSP APIs, which are exposed as "backend APIs", this is used for calling back to the TSP. This ensures scalability for internal TC components, and a single secure integration point between the TSP and TC systems. API Management would play a role as client when calling the TSP APIs.

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 TSP are only exposed in the API Management as pointers with authentication configured (no. 3 +3a in (Figure 6)). These are showed in the figure as `api.<tsp-domain>/external/api`, but will replaced with the actual endpoints given by TSPs for a given environment..

The TC endpoints for each environment are defined in section 7.

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 all interfaces when communicating data towards TC and TSP.

Example:

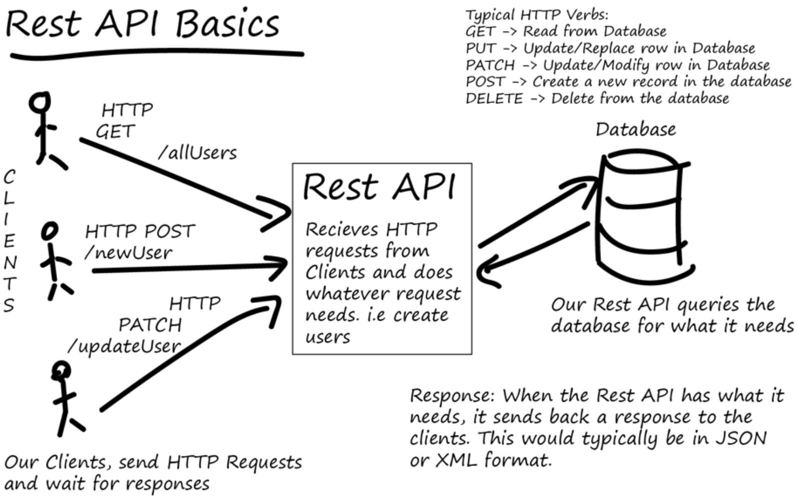

Rest API Basics
Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database
CLIENTS
HTTP GET /allUsers
HTTP POST /newUser
HTTP PATCH /updateUser
Rest API
Recieves HTTP requests from Clients and does whatever request needs. i.e create users
Database
Our Rest API queries the database for what it needs
Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.
Our Clients, send HTTP Requests and wait for responses

Figure 7. REST API basics

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:

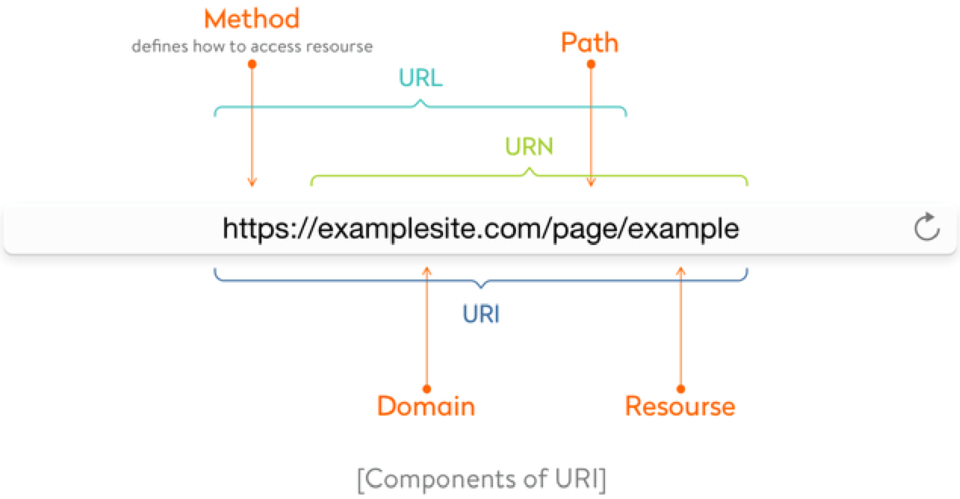
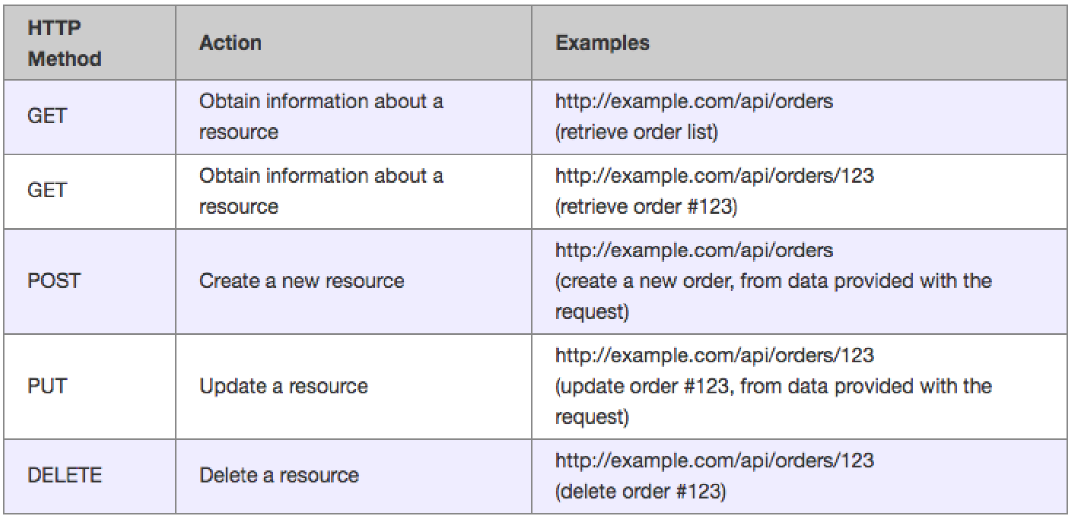

Method
defines how to access resourse
Path
URL
URN
https://examplesite.com/page/example
URI
Domain
Resourse
[Components of URI]

Figure 8. Components of a URI

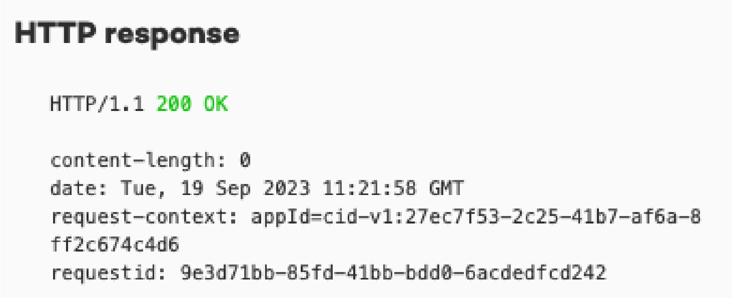


HTTP Method	Action	Examples
GET	Obtain information about a resource	http://example.com/api/orders (retrieve order list)
GET	Obtain information about a resource	http://example.com/api/orders/123 (retrieve order #123)
POST	Create a new resource	http://example.com/api/orders (create a new order, from data 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)

Figure *9.* HTTPS Methods

Example of response:

The response is a JSON file with an embedded list.



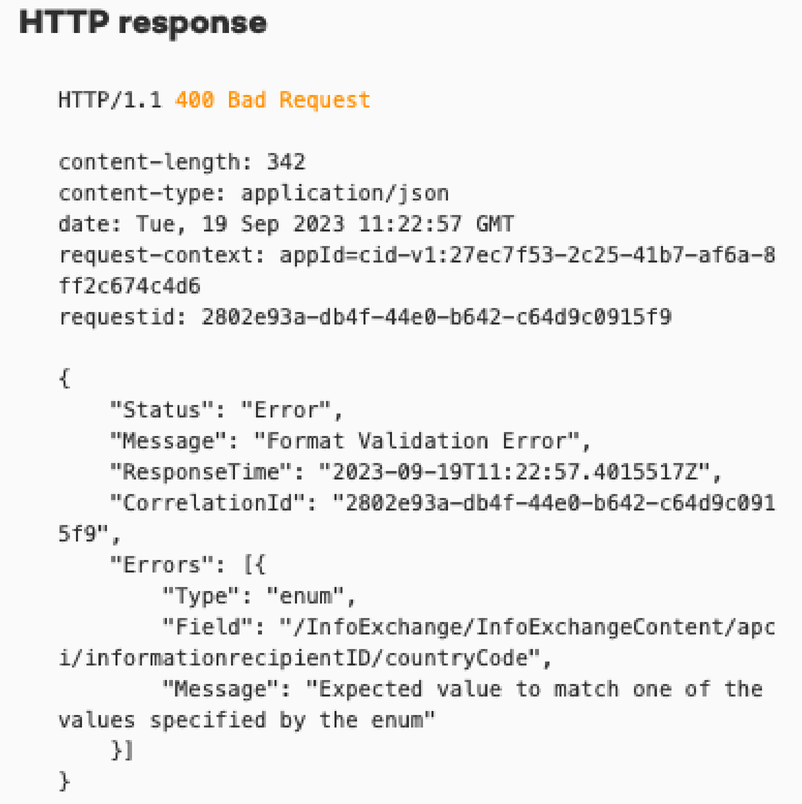
HTTP response

```
HTTP/1.1 200 OK

content-length: 0
date: Tue, 19 Sep 2023 11:21:58 GMT
request-context: appId=cid-v1:27ec7f53-2c25-41b7-af6a-8
ff2c674c4d6
requestid: 9e3d71bb-85fd-41bb-bdd0-6acdedfcd242
```

Figure *10.* HTTPS Response

Example on error on a specific parameter validation:



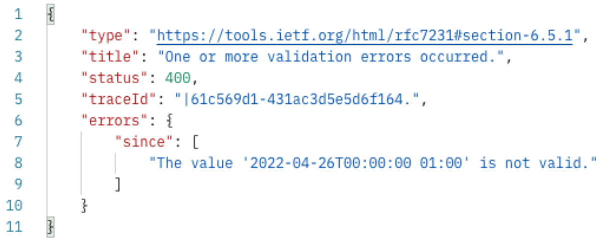
HTTP response

```
HTTP/1.1 400 Bad Request

content-length: 342
content-type: application/json
date: Tue, 19 Sep 2023 11:22:57 GMT
request-context: appId=cid-v1:27ec7f53-2c25-41b7-af6a-8
ff2c674c4d6
requestid: 2802e93a-db4f-44e0-b642-c64d9c0915f9

{
    "Status": "Error",
    "Message": "Format Validation Error",
    "ResponseTime": "2023-09-19T11:22:57.4015517Z",
    "CorrelationId": "2802e93a-db4f-44e0-b642-c64d9c091
5f9",
    "Errors": [{
        "Type": "enum",
        "Field": "/InfoExchange/InfoExchangeContent/apc
i/informationrecipientID/countryCode",
        "Message": "Expected value to match one of the
values specified by the enum"
    }]
}
```

Figure *11***.** HTTPS Error response



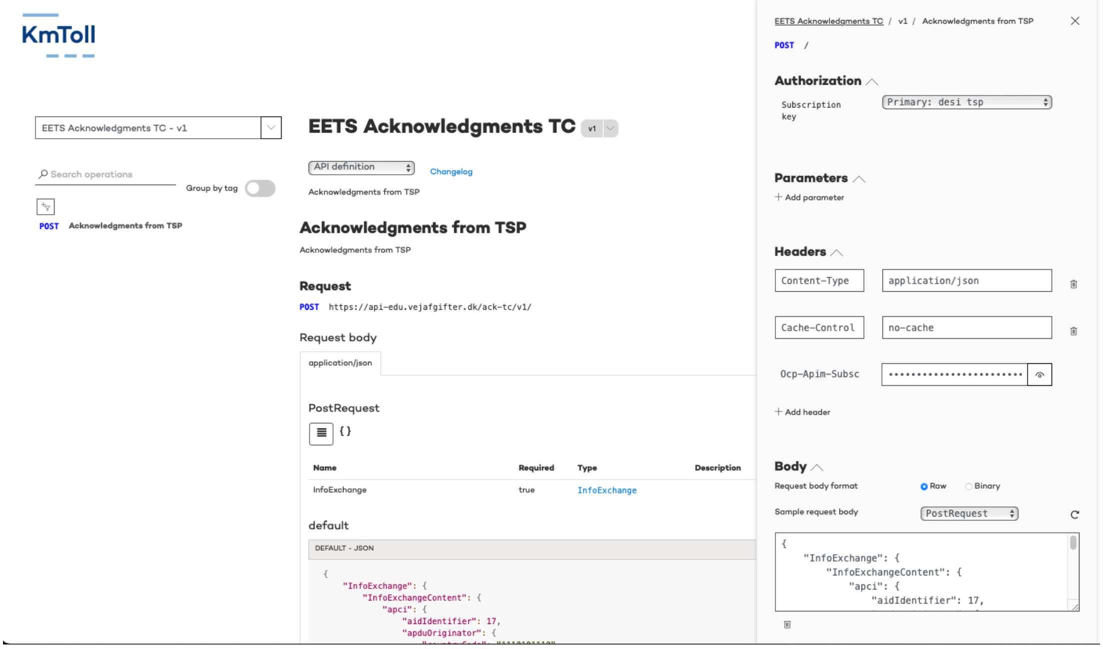
```
{
    "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
    "title": "One or more validation errors occurred.",
    "status": 400,
    "traceId": "|61c569d1-431ac3d5e5d6f164.",
    "errors": {
        "since": [
            "The value '2022-04-26T00:00:00 01:00' is not valid."
       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

OpenAPI Specification:

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

https://swagger.io/specification/

Swagger UI example in APIM:



Acknowledgments from TSP

Acknowledgments from TSP

Request

POST https://api-edu.vejafgifter.dk/ack-tc/v1/

Request body

application/json

PostRequest

Name	Required	Type	Description
InfoExchange	true	InfoExchange	

default

DEFAULT - JSON

```
{
    "InfoExchange": {
        "InfoExchangeContent": {
            "apci": {
                "aidIdentifier": 17,
                "apduOriginator": {
```

POST /

Subscription key: Primary: desi tsp

Content-Type	application/json
Cache-Control	no-cache
Ocp-Apim-Subsc	••••••••••••••••••••••••

```
{
    "InfoExchange": {
        "InfoExchangeContent": {
            "apci": {
                "aidIdentifier": 17,
```

Figure *12*. Swagger UI example in APIM

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 TSP and the TC are called Transactions in EN 16986. The following EN 16986 transactions must be supported:

Table *1*. EN16986 Transactions

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 lists) Ad-hoc (incremental lists)
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	Contractually agreed
PAYMENTANNOUNCEMENT	Allows a TSP to inform the TC that a payment has been made	Contractually agreed
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.

In addition to the transactions defined in EN 16986, a transaction has been defined which allows the TC to request compliance check data (CCC) from a TSP. This is to be used in the case where compliance data cannot be directly obtained from the roadside over the DSRC compliance check interface. This has been created to support OBE which does not have a DSRC capability, for example where an OBE is implemented as an App running on a smartphone platform.

A transaction to allow TSPs and TCs to exchange company-specific information (for example addressed, contacts, bank details etc) is also required. This will not be an API-based interface, but will be based on email exchanges (to be confirmed).

The transactions are described below:

Table 2. Additional transactions not defined in EN 16986

Transaction	Description	Typical frequency
CCCDATAEXCHANGE	Allows a TC to request CCC information for a specific user from a TSP. The TSP must respond with a package of CCC data which is similar in content to that supplied over the DSRC CCC interface for a DSRC-capable OBE. Only required if the TSP supports OBE Type 2.	Ad-hoc
ACTORTABLEEXCHANGE	Allows the TC or TSP to send updated information regarding addresses, bank details etc securely. This transaction is only expected to be required infrequently, so an API will not be defined for it. Instead Actor Tables will be exchanged as attachments to ITSM cases.	Ad-hoc

A Transaction usually consists of either one, two or three 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.

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 3. Interfaces to be implemented as Web APIs

Name	Implemented on	Description	Used in EN 16986 Transactions:
EETS Acknowledgements 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
EETS Acknowledgments TSP	TSP	Generic acknowledgement from TC A single interface will be implemented at the TSP to receive Acks	TOLLDECLARATIONS_SECT

		from TCs. The Acks will be used for multiple transactions.	EXCEPTIONLIST* PAYMENTANNOUNCEMENT CONTRACTISSUERLIST
EETS Toll Declaration	TC	Allows TSP to send road usage data in the form of Toll Declarations to the TC	TOLLDECLARATIONS_SECT
EETS Billing Details	TSP	Allows the TC to send Billing Details to the TSP	BILLINGDETAILS_TC
EETS Full WhiteList	TC	Allows the TSP to send full whitelists to the TC	EXCEPTIONLIST
EETS Incremental WhiteList	TC	Allows the TSP to send incremental whitelists to the TC	EXCEPTIONLIST
EETS Full BlackList	TC	Allows the TSP to send full blacklists to the TC	EXCEPTIONLIST
EETS Incremental BlackList	TC	Allows the TSP to send incremental blacklists to the TC	EXCEPTIONLIST
EETS Payment Claim	TSP	Allows the TC to send a Payment Claim (invoice) to the TSP	PAYMENTCLAIM
EETS Payment Announcement	TC	Allows the TSP to announce to the TC that an invoice (payment claim) has been paid	PAYMENTANNOUNCEMENT
EETS 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
EETS Contract Issuer List	TC	Allows the TSP to send contractual information regarding OBE to the TC (if implemented)	CONTRACTISSUERLIST
EETS CCC_Data_Request	TSP	Allows the TC to request CCC data from the TSP	CCCDATAEXCHANGE (not EN 16986)
EETS CCC Data Response	TC	Allows the TSP to send a response to the CCC_Data_Request to the TC	CCCDATAEXCHANGE (not EN 16986)

* See detailed description of the EETS Exception Lists for more information regarding the acknowledgement of Exception Lists.

Each interface will be implemented as a REST web service. Support for Trust Object and Actor Table Exchange transactions will not use REST interfaces. See sections 6.12 and 6.13 for further details.

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

- EETS Ack TC
- EETS Billing Details
- EETS Payment Claim
- EETS Report Abnormal_OBE
- EETS CCC Data Request (only if TSP supports OBE Type 2)

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

- EETS ACK TSP
- EETS Toll Declaration
- EETS Full White List
- EETS Incremental White List
- EETS Full Black List
- EETS Incremental Black List
- EETS Payment Announcement
- EETS CCC Data Response
- EETS 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 5.5 below. Messages rejected at the API 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 5.5 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.

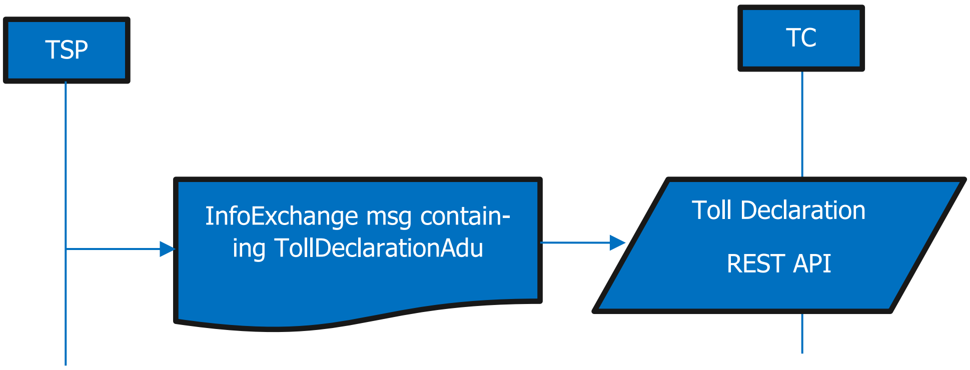


***Figure 13**. InfoExchange message example,* T*oll* D*eclaration*

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 message, 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.

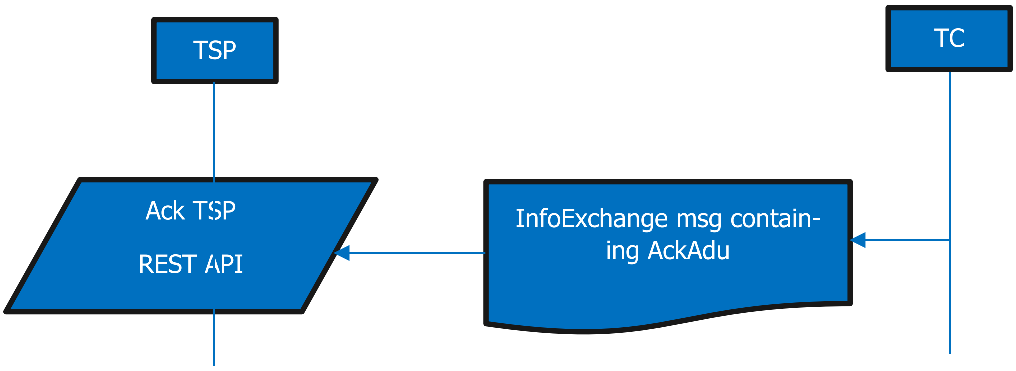


***Figure** 14. InfoExchange acknowledgement example*

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.

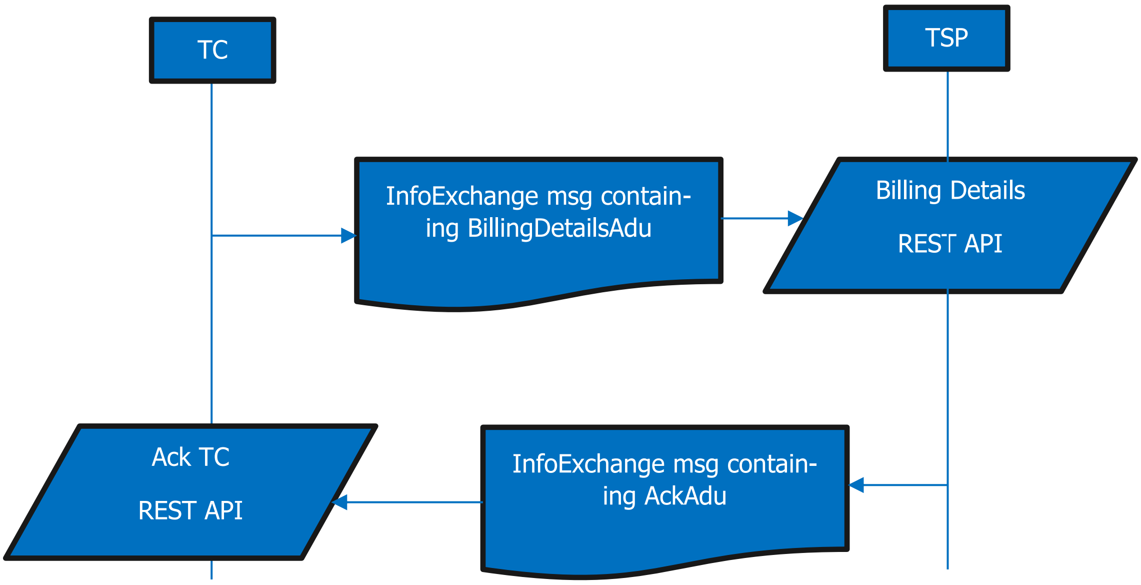

TC
TSP
InfoExchange msg containing BillingDetailsAdu
Billing Details
REST API
Ack TC
REST API
InfoExchange msg containing AckAdu

***Figure** 15. InfoExchange full flow example*

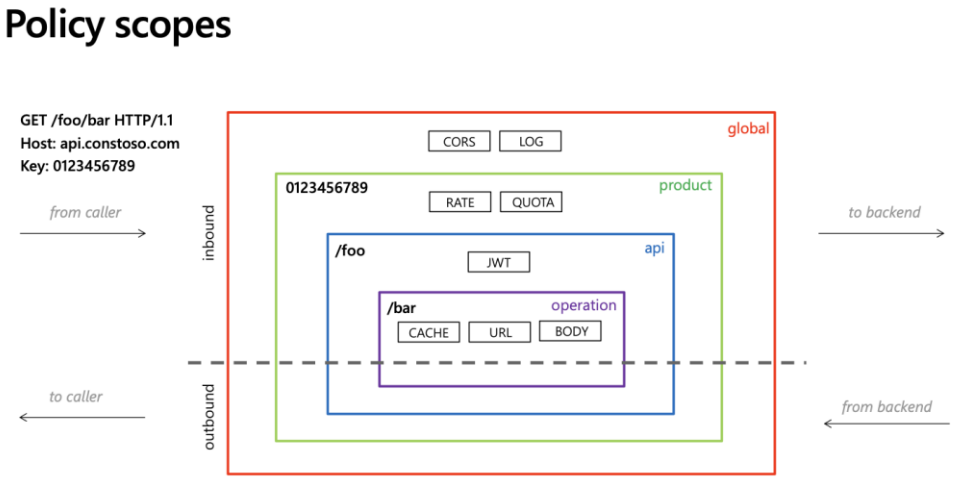
It can be seen that in all cases, the message receiver acts as the server – messages are pushed from sender to receiver.

5 Common aspects for all interfaces

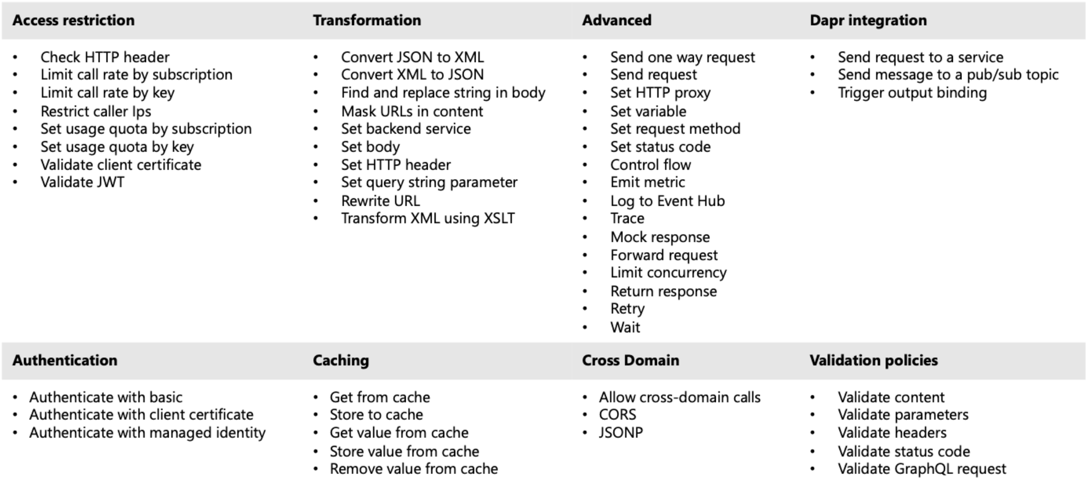
This chapter contains the specifications which are common to all the interfaces.

5.1 Control access

Each API will have so called "policies" applied, the scope of these policies is divided into different scopes: global, product and api, and ensures that each API will confine to the governance set by the TC. An example could be limiting 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.

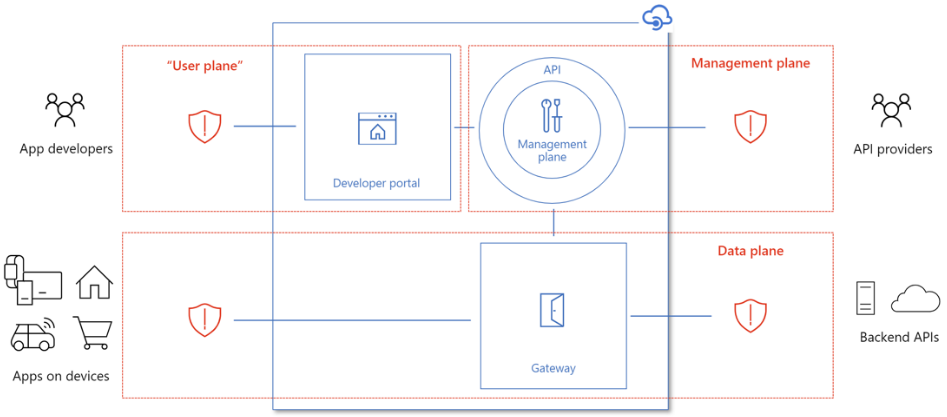


***Figure** 16. API Management*

When a TSP wishes to gain access to an API product, an API Management subscription key is required. The subscription key is generated by the TC and distributed securely to the TSP.

The API key type "Ocp-Apim-Subscription-Key" must be included in the HTTP header to the request, passing the value of a valid subscription key.

When API Management receives an API request from a client with a subscription key, it handles the request according to these rules:

- Check if it is a valid key associated with an active subscription.
- The subscription is scoped to an API product that is assigned to the TSP.
- If a valid key for an active subscription at an appropriate scope is provided, access is allowed. If a subscription key is incorrect or has expired, the error message "401 Access denied error" will be transmitted.

Subscription keys will have a lifetime of two years.

Each TSP will be provided with two subscription keys, a primary and a secondary. For the first year of operation, the TSP must use the primary key. At the end of each year, the TSP should switch to using the other key, and the first key will be replaced with a new key issued by the TC. This is shown diagrammatically in **Table** 4 below:

Table *4.* Subscription Key Management

	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6
Subscription key 1	🟩	🟧	🟧	🟧	🟧	🟧
Subscription key 2	🟨	🟩	🟧	🟧	🟧	🟧
Subscription key 3		🟨	🟩	🟧	🟧	🟧
Subscription key 4			🟨	🟩	🟧	🟧
Subscription key 5				🟨	🟩	🟧
Subscription key 6					🟨	🟩

🟩	Active Subscription key
🟨	Reserve subscription key
🟧	Subscription key deactivated

In year 1, two keys are issued, subscription key 1 (Primary key) and subscription key 2 (Secondary key). Subscription key 1 is the active key, and subscription key 2 is the reserve subscription key. At the end of each year, the previous reserve key (key 2) becomes the active key. Subscription key 1 is revoked, and a new subscription key (key 3) is issued and becomes the new reserve key. This is repeated at the end of each year. In this way, keys have a lifespan of 2 years, and the TSP always has 2 keys available for use. The first Primary key will be updated after one year and therefor only have a life span of one year.

5.2.1 Data plane access

To be able to use the TC provided APIs, API Management has the responsibility to do the initial authentication to prevent unauthorized access. This is done 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.

Figure 17 shows the process flow for using the OAuth 2.0 flow for accessing the APIs provided by TC:

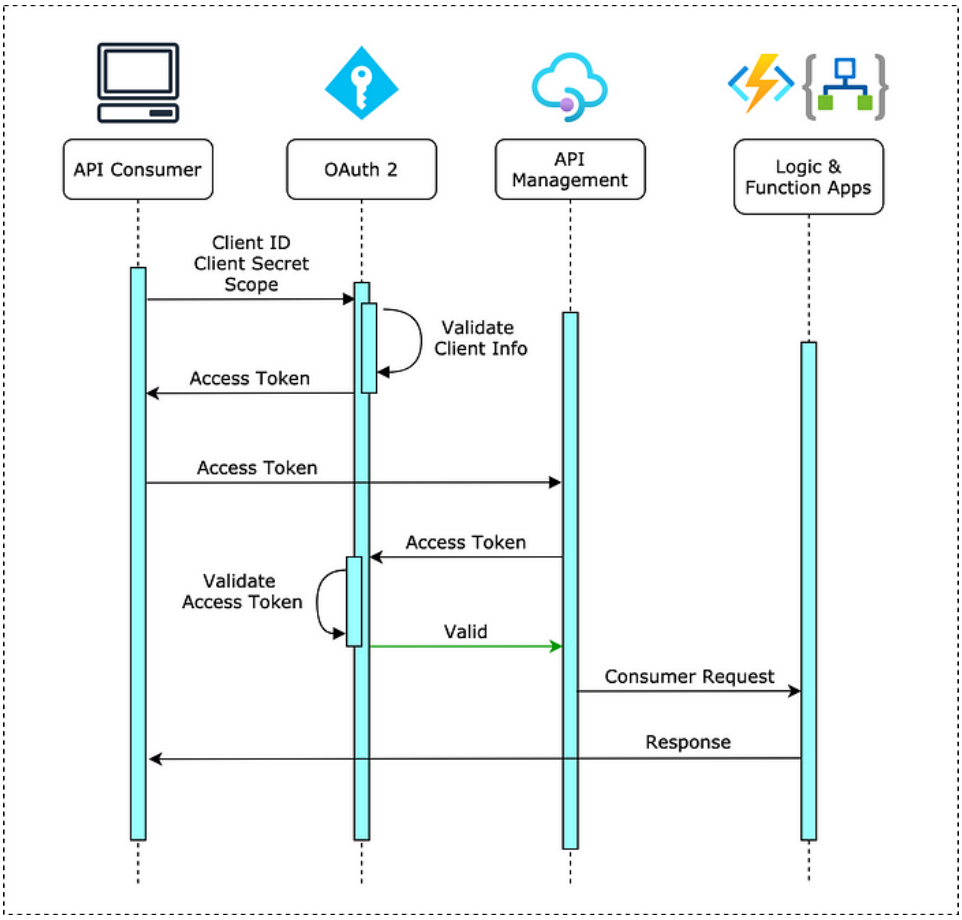
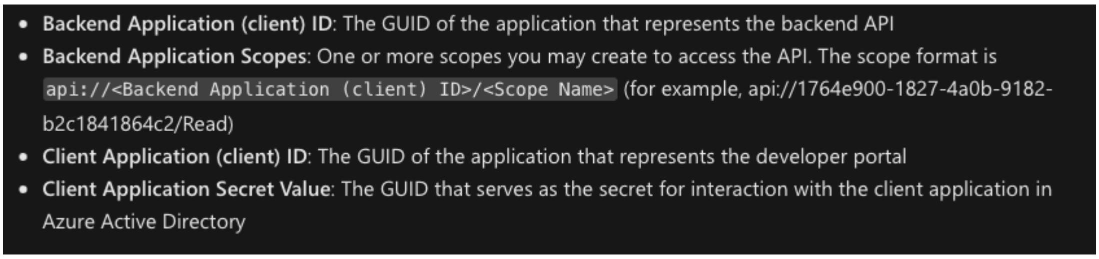


Figure *17*. Standardised OAuth 2.0 flow

It is a requirement that the TSP as a minimum secures its APIs with the same OAuth flow as shown in **Figure** 17 and that the TSP register the TC's API Management as client in their OAuth 2.0 Identity provider. TC will keep the keys of the TSP in the TC's Azure Key Vault and will only be used when calling the "backend API" of the TSP.

TSP is required to use Oauth 2.0 "Client Credential flow = machine to machine(M2M). See Oauth 2.0 get-started guide here if needed.

For the purpose of secure exchange, the TSP and TC shall share the following information ('secrets') with each other. The information is to be shared via secure exchange, using a method chosen by the TC:



- **Backend Application (client) ID**: The GUID of the application that represents the backend API
- **Backend Application Scopes**: One or more scopes you may create to access the API. The scope format is `api://<Backend Application (client) ID>/<Scope Name>` (for example, api://1764e900-1827-4a0b-9182-b2c1841864c2/Read)
- **Client Application (client) ID**: The GUID of the application that represents the developer portal
- **Client Application Secret Value**: The GUID that serves as the secret for interaction with the client application in Azure Active Directory

5.2.2 User plane login and subscription key – Developer portal (EDU and UAT environments only)

The TSP is required to provide the TC with an email address to be used for creating an account and providing login access to the Developer Portal.

Only one company-specific email address is allowed per TSP. It is a requirement that the email address is not a 'dead' email, and that the TSP can access and read replies since important information may be received on the email.

The TC is responsible for generating a username and create a user for the TSP. Once a user is created the TSP will receive an email with a link to active the user. The TSP should use the link to generate a password for the TC's organisation.

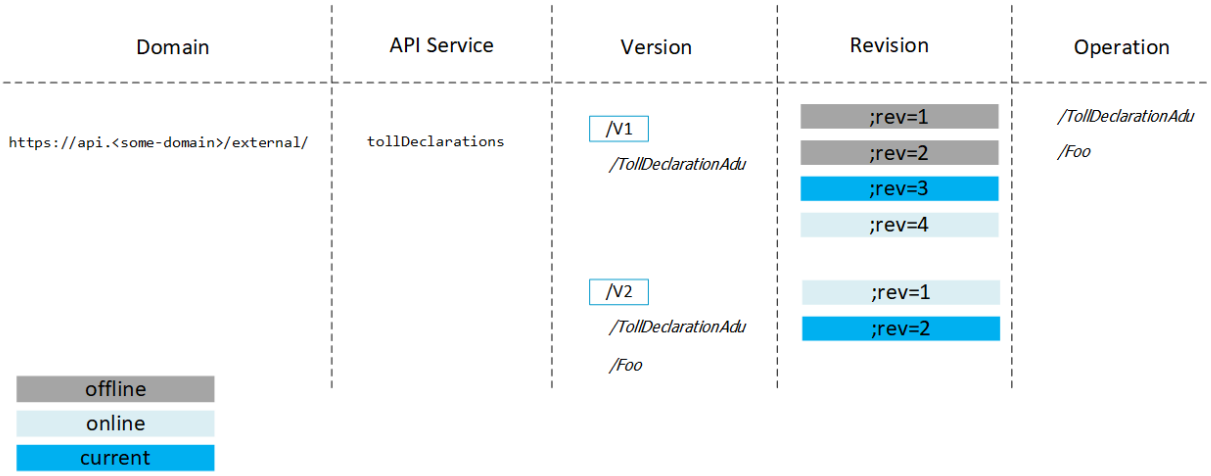
Once access is granted the TC will ensure that the TSP are given access to the relevant product(s) (group of APIs). Then the TSP will have access to the products in the Developer Portal from where they can get the relevant subscription key for the APIs.

It is a requirement that the relevant subscription key is filled out if the TSP wishes to submit request using the 'Try it' function in the Developer Portal.

5.3 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.

Table 5. Versioning



Domain	API Service	Version	Revision	Operation
https://api.<some-domain>/external/	tollDeclarations	/V1 /TollDeclarationAdu	;rev=1 ;rev=2 ;rev=3 ;rev=4	/TollDeclarationAdu /Foo
		/V2 /TollDeclarationAdu /Foo	;rev=1 ;rev=2	

The length of time for which an old version will remain online before being declared offline will depend on the criticality of the version upgrade. Three levels of criticality for incoming interfaces at the TC are defined:

- Critical – a new version is required to correct a defect which directly impacts the correct operation of the system. Critical updates can occur at any time, and the previous version will remain online if possible, for the shortest feasible time, typically one week.
- Urgent – a new version is required to correct a defect which has an impact on the smooth running of the system. Urgent updates can occur at any time, and the previous version will remain online for at least two weeks after release of a new version.

- Routine – an upgrade has been issued with updated or improved functionality. Typical reasons will include implementing changes required by law, changes in scheme rules and IT security etc. Routine upgrades should be expected once per year. The previous version will remain online for at least two months after release of a new version.

5.4 API products

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

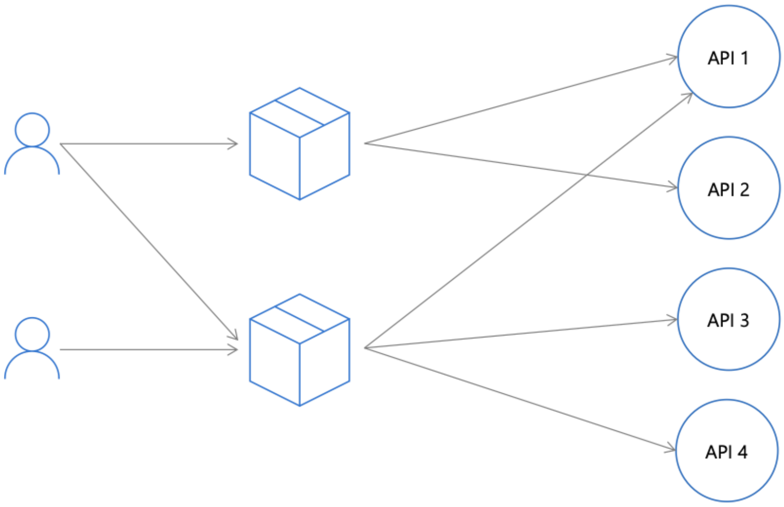


Figure 18. Logical products

5.5 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 Developer Portal for the implemented HTTP response codes, used by the different APIs.

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

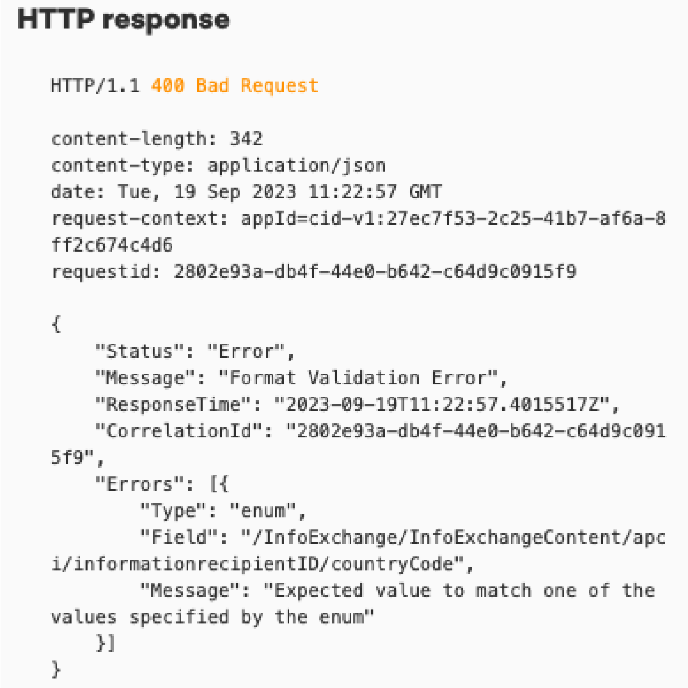
The body could look like this:

```
{
     "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.

Example of a code 400 error is shown in the code snippet below, where the value expected is not the same as required in the defined 'enum’



HTTP response

```
HTTP/1.1 400 Bad Request

content-length: 342
content-type: application/json
date: Tue, 19 Sep 2023 11:22:57 GMT
request-context: appId=cid-v1:27ec7f53-2c25-41b7-af6a-8
ff2c674c4d6
requestid: 2802e93a-db4f-44e0-b642-c64d9c0915f9

{
    "Status": "Error",
    "Message": "Format Validation Error",
    "ResponseTime": "2023-09-19T11:22:57.4015517Z",
    "CorrelationId": "2802e93a-db4f-44e0-b642-c64d9c091
5f9",
    "Errors": [{
        "Type": "enum",
        "Field": "/InfoExchange/InfoExchangeContent/apc
i/informationrecipientID/countryCode",
        "Message": "Expected value to match one of the
values specified by the enum"
    }]
}
```

6 Individual interface descriptions

The interfaces between the TC and TSP are defined in this section.

In general, a separate interface is defined for each ADU defined in EN 16986, though there are exceptions to this, for example a separate interface is defined for each of the four supported Exception Lists.

Note that an interface only allows data transfer in one direction. For example, the AckAdu described in section 6.5 of EN 16986 requires two interfaces, one for each direction, so the *EETS Acknowledgements TSP* interface allows the TC to send acknowledgements to the TSP, while the *EETS Acknowledgements TC* interface allows the TSP to send acknowledgements to the TC.

The data formats used in the interfaces described below are defined in EN 16986, which in turn uses the underlying standard ISO 12855:2022. While the standards define the data formats used in the interfaces, they may contain optional elements which are further defined in this section. In addition, some interfaces have additional fields to those defined in the standards. These are also defined in this section.

In addition to the interfaces defined in prEN16986:2023, two additional interfaces are defined, namely EETS CCC Data Request and EETS CCC Data Response. These are required to support app-based OBE which does not have a DSRC interface for compliance checking.

Note that the data formats accessed through and defined in OpenAPI specifications are the definitive definitions. If the data formats and restrictions defined in this document differs from that in OpenAPI specification, the OpenAPI specification takes precedence.

It is important to note that ISO 12855, and hence EN 16986, require that data is encoded using XML, and EN 16986 provides example XML Schema files for each of the profiles defined. The Danish KM Toll project has decided that data will be encoded in JSON (JavaScript Object Notation) and defined in appropriate JSON schemas. The JSON schemas for each interface are available in the developer portals after agreement with TC for accreditation and development purposes.

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 EETS Acknowledgments TSP Interface

6.1.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS Acknowledgments TSP – vn*", where "*vn*" is the version number currently in use.

6.1.2 Purpose of the Interface

The EETS Acknowledgments 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 EETS Acknowledgments TSP interface may only be used as part of the following transactions:

- TOLLDECLARATIONS_SECT
- PAYMENTANNOUNCEMENT
- CONTRACTISSUERLIST
- EXCEPTIONSLISTS

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

6.1.3 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.
2. Acknowledgments may be synchronous or asynchronous, depending on the transaction. See the descriptions for Toll Declarations, Payment Announcement, Contract Issuer List and Exception List interfaces for more details.

6.1.4 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 EN 16986.

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 EN 16986.

The ApduReasonCodes defined in Table 9 of ISO 12855:2022 must be supported.

The following AduReasonCodes defined in Table 11 of ISO 12855:2022 must be supported:

- Values 0 – 2 (generic ADU errors)
- Values 600 – 612 (toll declaration related errors)
- Value 3000 (semantic error)
- Value 3010 (action code not supported error)

The following user defined values must also be supported - this list may be further expanded in future versions of this specification:

- Value 10300 (Contract issuer list adu not accepted)
- Value 10900 (payment announcement adu not accepted)

Ranges marked as "reserved" will only be used when defined in future versions of the standards, or their use has been defined in bilateral agreements between the TC and the TSP.

Note further that it is not foreseen that the value 4000 (acceptedWithWarning) will be used, but it may be used in a future version of this interface.

6.1.5 Error Handling

API errors will be handled as described in section 5.5.

6.2 EETS Billing Details interface

6.2.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS Billing Details – vn*", where "*vn*" is the version number currently in use.

6.2.2 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 EETS Users of the TSP during the billing period covered by the Billing Details.

6.2.3 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 following additional specifications apply to this interface – Figure 14 from prEN16986:2023 is reproduced below for reference:

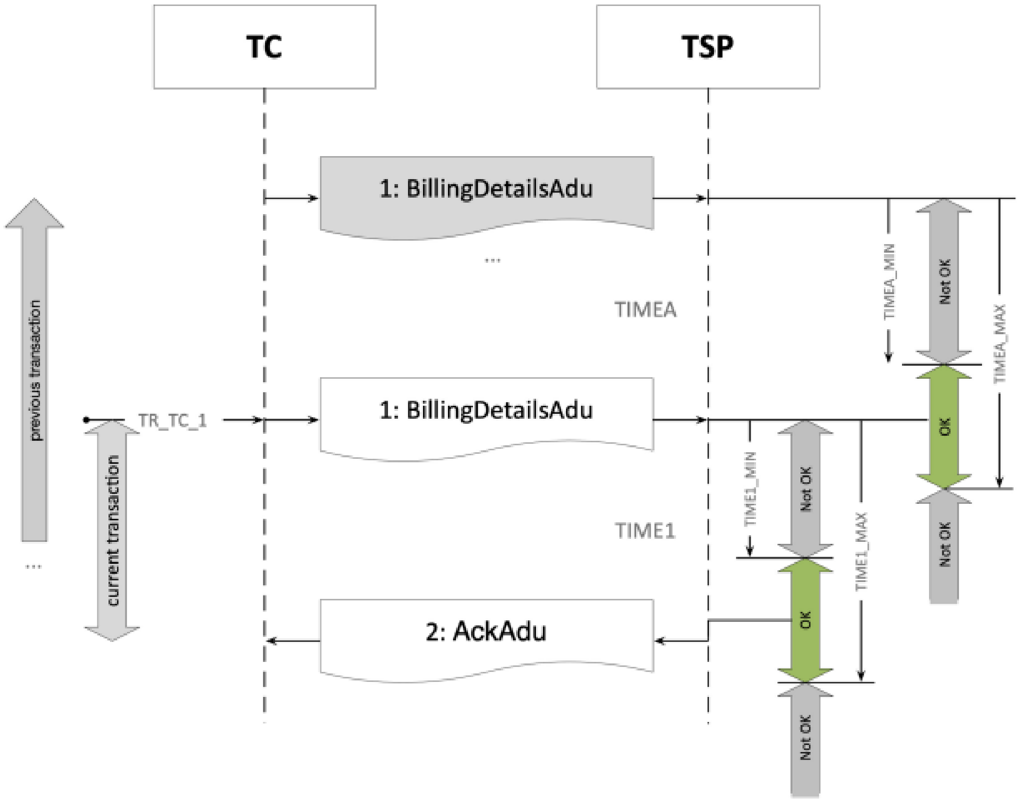


Figure *19*. Copy of Figure 14 from EN 16986

1. The TC will issue Billing Details once per billing period, currently assumed to be once every 24 hours. This time is contractually defined.
2. Only one *billingDetailsAdu* is allowed per transaction. This means that a separate Billing Details message will be sent for each end user.
3. Each Billing Details message must be acknowledged on the EETS Acknowledgements TC interface.

4. Billing details for different users can be sent in parallel. Therefore TIMEA_MIN (prEN16986:2023 Table 57 and Figure 14, is defined as 0s (for different users).
5. TIME1_MIN (prEN16986:2023 Table 57) is defined as 0.
6. TIME1_MAX (prEN16986:2023 Table 57) is set to 24 hours.
7. If TIME1_MAX is reached without an acknowledgement, the billing details may be re-sent. Re-sends will be sent a maximum of twice (i.e., a total of three times) before escalation to the TSP.

6.2.4 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 EN 16986.

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 EN 16986.

Identification of OBE type

The KmToll Scheme supports both OBE using dedicated hardware and OBE implemented as software on a nomadic platform (e.g., a smartphone), called Type 1 and Type 2 respectively. In order to identify the OBE type, a new data field called "*obeType"* is introduced as part of the "*obeId"* field within the adu. *obeType* is defined below:

Data Element	Data type	Description
obeType	Integer	Identified the OBE type in use. It can take the values: "1" – OBE is of type 1 "2" – OBE is of type 2 Values between "3" and "127" are reserved for future use.

Use of the actionCode field

When sending Billing Details to the TSP, the field *actionCode* will normally contain the value *send*, which indicates that this is a new Billing Details which needs to be processed by the TSP. However, it could happen that a previously sent Billing Details needs to be updated, for example following a successful complaint by an end user. This will be achieved by re-sending the Billing Details which are to be changed in a new *billingDetails* message. The re-sent Billing Details will contain the same *aduIdentifier* as the Billing Details being changed, but with the *actionCode* value *revoke,* which indicates to the TSP that the Billing Details should be cancelled. A new updated Billing Details will be sent in a new *billingDetails* message.

6.2.5 Error Handling

API errors will be handled as described in section 5.5.

6.3 **EETS Payment Claim interface**

6.3.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS Payment Claim – vn*", where "*vn*" is the version number currently in use.

6.3.2 Purpose of the Interface

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

6.3.3 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 – Figure 16 from prEN16986:2023 is reproduced below for reference:

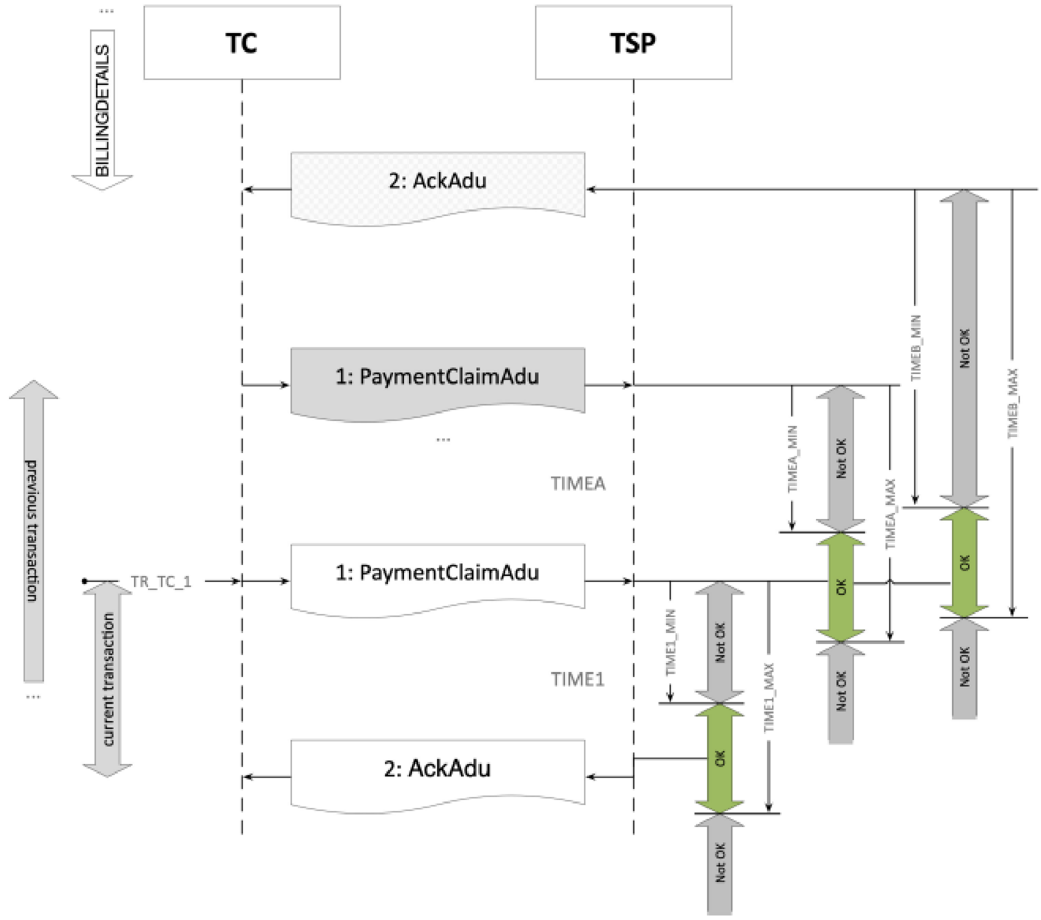


Figure *20*. Copy of Figure 16 from EN 16986

1) Payment Claims will be pushed to the TSP at contractually agreed intervals. This is currently expected to be once per calendar month.
2) Payment claims will be synchronously acknowledged, i.e., a Payment Claim must be acknowledged before another can be sent, though timeouts will apply.
3) It is possible that Payment Claims will be too large to be transferred in a single message, in which case they must be sent as multiple chunks (see description of the chunking process in "Message Formats" below).
4) If no chunking is required, each Payment Claim list message will be acknowledged with an *Acknowledge TC* Adu.
5) In the case where chunking is required, assume a message has "n" chunks:

a) Chunks 1 to (n-1) are acknowledged at the API level only (i.e., no *Acknowledge TC* Adu is sent)

b) Once all n chunks have been received, the entire list will be acknowledged with an *Acknowledge TC Adu*. Each chunk will be explicitly acknowledged within the *Acknowledge TC Adu* with a separate *AckTcAdu* data field.

6) TIMEA_MIN and TIMEA_MAX (prEN16986:2023 Table 71 and Figure 16) are undefined

7) TIMEB_MIN (prEN16986:2023 Table 71) is defined as 0

8) TIMEB_MAX (prEN16986:2023 Table 71) is undefined

9) TIME1_MIN (prEN16986:2023 Table 71) is defined as 0

10) TIME1_MAX (prEN16986:2023 Table 71) is defined as 24 hours

6.3.4 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 EN 16986. The *referenceDetailsList* data element will be used, containing the *billingDetailsList* choice.

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 EN 16986.

The *referenceDetailsList* contains a list of all the Billing Details for which payment is being claimed. As this may be a very large list (one entry per customer per day, so for a large TSP this could amount to millions of entries), the interface includes a mechanism for dividing the message into multiple smaller parts or "chunks". Each chunk will contain a maximum of 10,000 Billing Details entries. Four additional data elements are included to supporting the chunking:

- *totalPaymentClaimReferenceListEntries* – Total number of entries for this Payment Claim
- *paymentClaimReferenceListEntriesFromRange* – The value either "1" (first payload) or next number after previous *paymentClaimReferenceListEntriesToRange* (2nd and subsequent payloads)
- *paymentClaimReferenceListEntriesToRange* – *paymentClaimReferenceListEntriesFromRange* + *countOfPaymentClaimReferenceListEntries* – 1
- *countOfPaymentClaimReferenceListEntries* – Count of referenceDetailList entries in this payload

In cases where the chunking mechanism is required, each *apduIdentifier* must be unique as required by the standards. However, the different chunks of the same payment claim must have the same *aduIdentifier*.

Example 1: Payment Claim references 1 Billing Details (no chunking required):

Data Element	Value
apduIdentifier	12345
aduIdentifier	6789
totalPaymentClaimReferenceListEntries	1
paymentClaimReferenceListEntriesFromRange	1
paymentClaimReferenceListEntriesToRange	1

countOfPaymentClaimReferenceListEntries	1

Example 2: Payment Claim references 1000 billing details (no chunking required):

Data Element	Value
apduIdentifier	12345
aduIdentifier	6789
totalPaymentClaimReferenceListEntries	1000
paymentClaimReferenceListEntriesFromRange	1
paymentClaimReferenceListEntriesToRange	1000
countOfPaymentClaimReferenceListEntries	1000

Example 3: Payment Claim references 10,900 billing details, so two chunks required:

First message containing chunk 1

Data Element	Value
apduIdentifier	12345
aduIdentifier	6789
totalPaymentClaimReferenceListEntries	10900
paymentClaimReferenceListEntriesFromRange	1
paymentClaimReferenceListEntriesToRange	10000
countOfPaymentClaimReferenceListEntries	10000

Second message containing chunk 2

Data Element	Value
apduIdentifier	23456
aduIdentifier	6789
totalPaymentClaimReferenceListEntries	10900
paymentClaimReferenceListEntriesFromRange	10001
paymentClaimReferenceListEntriesToRange	10900
countOfPaymentClaimReferenceListEntries	900

6.3.5 Error Handling

API errors will be handled as described in section 5.5.

6.4 EETS Report Abnormal OBE interface

6.4.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS Report Abnormal OBE – vn*", where "*vn*" is the version number currently in use.

6.4.2 Purpose of the Interface

The interface allows the TC to report that the OBE in one of the TSP's EETS User's vehicle is behaving abnormally.

6.4.3 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 *ReportAbnormalObeAdu* will be acknowledged at the API level only, i.e., the *AckAdu* will not be used to acknowledge the *ReportAbnormalObeAdu.*

The following additional specifications apply to this interface:

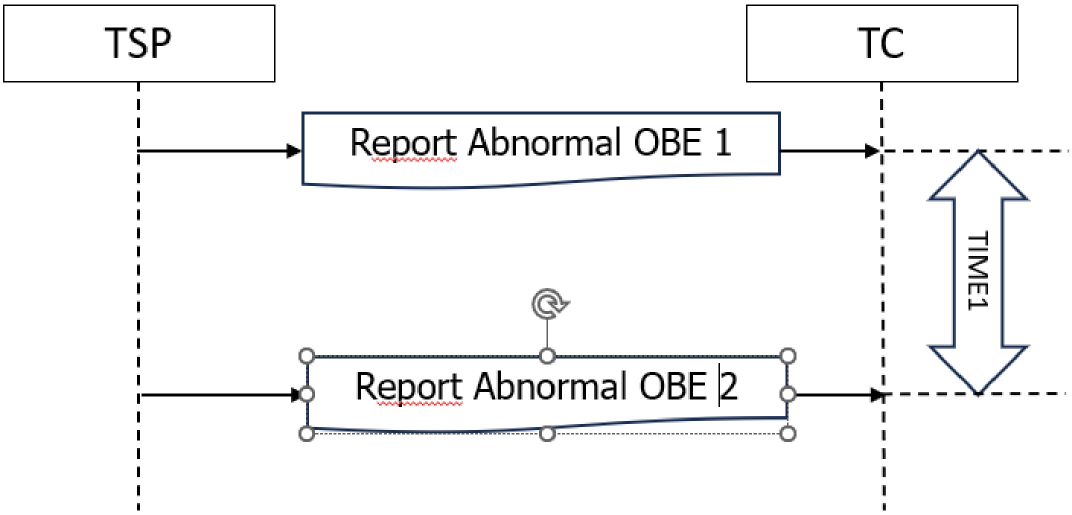


Figure 21. Report Abnormal OBE message flows

1. Each *reportAbnormalObe* message need be acknowledged at the API (HTTPS) level only.
2. Multiple *reportAbnormalObe* messages can be sent in parallel.
3. TIME1_MIN is defined as 0

6.4.4 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 EN 16986.

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 EN 16986.

6.4.5 Error Handling

API errors will be handled as described in section 5.5.

6.5 **EETS CCC Data Request interface**

6.5.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS* CCC Data Request – *vn*", where "*vn*" is the version number currently in use.

6.5.2 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 a TSP which intend to offer an OBE Type 2 to the EETS User will be required to implement this interface.

6.5.3 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 EN 16986, 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.*

The communications sequence is shown below.

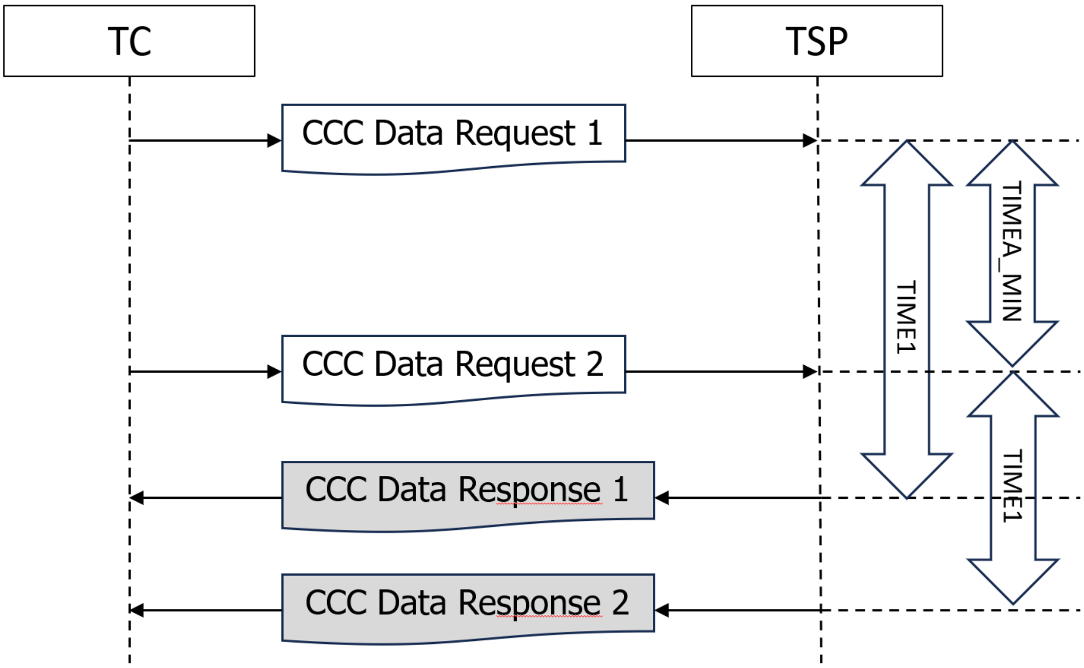


Figure 22. CCC Data Request flow

The TSP is required to respond to the CCC Data Request with a CCC Data Response.

TIMEA_MIN, the minimum time between successive CCC Data Requests is 0. Values for TIME1 are given in the description of the CCC Data Response interface, see section 6.14.

6.5.4 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 Section-Autonomous profile (with TC dominance) of specification EN 16986.

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

Within the RequestAdu, the *requestedADUType* element will contain the user defined value of 128 to indicate that this is a CCC_Data_Request.

6.5.5 Error Handling

API errors will be handled as described in section 5.5.

6.6 **EETS Acknowledgments TC Interface**

6.6.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS* Acknowledgments TC – *vn*", where "*vn*" is the version number currently in use.

6.6.2 Purpose of the Interface

The EETS Acknowledgments TC 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 EETS Acknowledgments 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.

6.6.3 Communications Processes

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

Acknowledgments may be synchronous or asynchronous, depending on the transaction. See the descriptions for Billing Details, Payment Claim, and Report Abnormal OBE interfaces for more details.

6.6.4 Message Formats

The EETS Acknowledgments 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 EN 16986.

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 EN 16986.

The AduReasonCodes defined in Table 9 of ISO 12855:2022 must be supported.

The following AduReasonCodes defined in Table 11 of ISO 12855:2022 must be supported:

- Values 0 – 2 (generic ADU errors)
- Values 500 – 502 (abnormal OBE report errors)
- Values 700 – 716 (billing details related errors)
- Values 800 – 809 (payment claim related errors)
- Value 3000 (semantic error)
- Value 3010 (action code not supported error)

Ranges marked as "reserved" will only be used when defined in future versions of the standards, or their use has been defined in bilateral agreements between the TC and the TSP.

Note further that it is not foreseen that the value 4000 (acceptedWithWarning) will be used, but it may be used in a future version of this interface.

6.6.5 Error Handling

API errors will be handled as described in section 5.5.

6.7 **EETS Toll Declaration interface**

6.7.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS Toll Declaration – vn*", where "*vn*" is the version number currently in use.

6.7.2 Purpose of the Interface

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

The EETS Toll Declaration contains the GNSS position data generated by the On-Board Equipment (OBE) of the TSP, which is required by the TC for the calculation of tolls. The report also contains toll-relevant data about the vehicle, for example maximum permissible train '6.2, vehicle class, emissions class etc.

Note that a Toll Declaration may only contain data for a single OBE

Note 2 Examine the restrictions in the next section carefully. It is important to understand that acknowledgement restrictions for Toll Declarations from the same OBE are different to acknowledgement restrictions for Toll Declarations from different OBE.

6.7.3 Communications Processes

The Toll Declaration 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 – Figure 21 from prEN16986:2023 is reproduced below for reference:

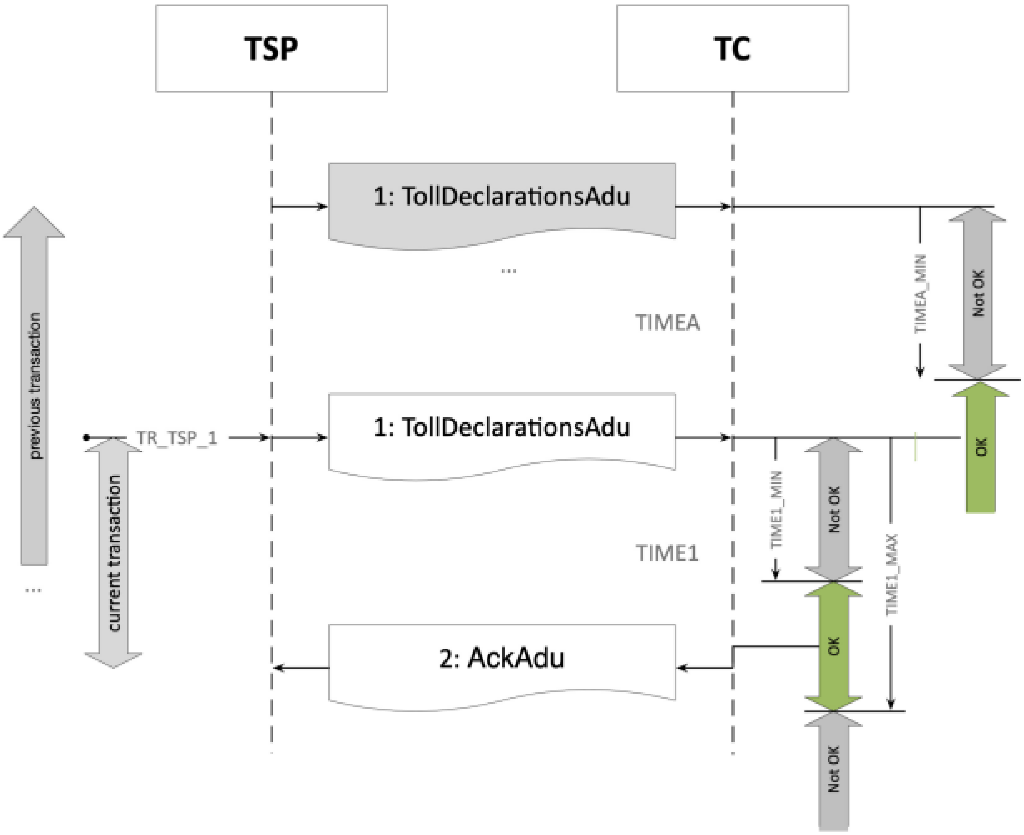


Figure 23. Copy of Figure 21 from EN 16986

1. Multiple Toll Declarations from different OBE can be started in parallel.
2. Toll Declarations from the same OBE MUST be sent in time sequence, as determined by the timestamps of the raw GNSS data points. Out of sequence Toll Declarations will not be processed.
3. Each Toll Declaration from the same OBE must be synchronously acknowledged, i.e., once a Toll Declaration for a specific OBE has been sent, a subsequent Toll Declaration for that OBE can only be sent after the initial one has been acknowledged.
4. If, after an agreed timeout period (see TIME1_MAX below) no acknowledgement is received, the toll declaration should be re-sent.
5. TIMEA_MIN (prEN16986:2023 Table 132 and Figure 21) is defined as 0.
6. TIME1_MIN (prEN16986:2023 Table 132) is defined as 0.
7. TIME1_MAX (prEN16986:2023 Table 132) is defined as 5 minutes.

6.7.4 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 EN 16986.

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 EN 16986.

A toll declaration may only contain one *tollDeclarationAdu*, which in turn may only contain one *usageStatement*. Hence a toll declaration may only contain tolling information from one OBE.

Identification of OBE type

The KM-Toll system supports both OBE using dedicated hardware and OBE implemented as software on a nomadic platform (e.g., a smartphone), called Type 1 and Type 2 respectively. In order to identify the OBE type, a new data field called “*obeType”* is introduced as part of the “*obeId”* field. *obeType* is defined below:

Data Element	Data type	Description
obeType	Integer	Identified the OBE type in use. It can take the values: “1” – OBE is of type 1 “2” – OBE is of type 2 Values between “3” and “127” are reserved for future use.

Addition of Additional GNSS data fields.

In addition to the data fields defined in ISO 12855:2022 and EN 16986, a new data field called “*additionalGnssData*” is introduced as part of the “*measuredRawData*” field. This will contain the following data elements:

Data Elements	Data type	Description
additionalGnssData		Provides additional GNSS information as measured by the GNSS receiver.
Speed	Integer	Measured speed on the GNSS receiver, in cm/s. Minimum speed is “0”, maximum speed is 9999 cm/s. If this data is not available, the value “-1” should be entered. In the unlikely event that the measured speed is greater than 9999 cm/s, then the speed value must be recorded as “9999”
heading	Integer	Measured heading of the GNSS receiver, in tenths of a degree. Allowed values are “0” to “3599”, where “0” is due north and #3599” is a heading of 359.9 degrees. If this data is not available, the value “-1” should be entered.

	Integrity	string	Signal integrity as reported by the GNSS receiver. This is a single character, reported as a string of length 1, with the following allowed values: “A” – Autonomous mode “D” – Differential mode “E” – Dead reckoning for locates without subsequent editing “M” – Manual mode for locates supplemented subsequently “S” – Simulated mode with subsequently edited location information “N” – Non-valid location.
	noOfSatellites	Integer	Number of satellites used to obtain the geo-location. This must be a value between “0” and “99”. “0” satellites should be reported if this data is not available, for example for simulated locations, dead reckoning etc. In the unlikely event that the number of satellites used is greater than 99, the number must be reported as “99”
	hDOP	Integer	The Horizontal Dilution of Precision, as reported by the GNSS receiver. This must be a value between “0” and “999”. This value must be reported in tenths, i.e., if HDOP is 1, it should be reported as the value “10” If HDOP is unavailable, it should be reported as value “-1”
	hAcc	Integer	Horizontal Accuracy of the reported geo-location. This must be a value between “0” and “9999”. This is an estimate of the 1-sigma error radius for the geo-location, i.e. this is the radius of the likely uncertainty error for a 68% (1-sigma) error probability. hAcc must be reported in cm. If hAcc is unavailable, it should be reported as value “-1”

The data fields speed, heading, hDOP, hAcc allow the use of a “data not available” indicator. It is not acceptable for this to be a default value; it is only to be used in specific circumstances where the data is temporarily unavailable for technical reasons.

Vehicle environmental characteristics

In order to comply with EU directives, it has been necessary to add a new attribute to the field *environmentalCharacteristics*, describing the vehicle's CO2 emissions class. This attribute is called *euCO2EmissionClass*, and is an integer with a value between 1 and 5 inclusive.

Note that three different attributes are defined within the *VehicleWeightLimits* field. At this time, it is still to be finalised which will be used in the toll calculation.

6.7.5 Error Handling

API errors will be handled as described in section 5.5.

6.8 **EETS Exception Lists interfaces**

6.8.1 Identification of the Interfaces

These interfaces are identified on the relevant portal by the names “*EETS Full Black List – vn*”, “*EETS Full White List – vn*”, “*EETS Incremental Black List – vn*”, and “*EETS Incremental White List – vn*”, where “*vn*” is the version number of the interface currently in use.

6.8.2 Purpose of the Interface

The interfaces are used by the TSP to transfer exception lists to the TC. Exception lists can be of four types: Full Whitelist, Incremental Whitelist, Full Blacklist, and Incremental Blacklist

Although each exception list type is transferred using a separate interface, these are all described in this section as they are extremely similar. The four interfaces described in this section are:

- EETS Full Whitelist
- EETS Incremental Whitelist
- EETS Full Backlist
- EETS Incremental Backlist

6.8.3 Communications Processes

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

The *AckAdu* is used to acknowledge the EETS Exception Lists, but with special considerations for full lists – see the Message Formats section below.

The following additional specifications apply to this interface – Figure 9 from prEN16986:2023 is reproduced below for reference:

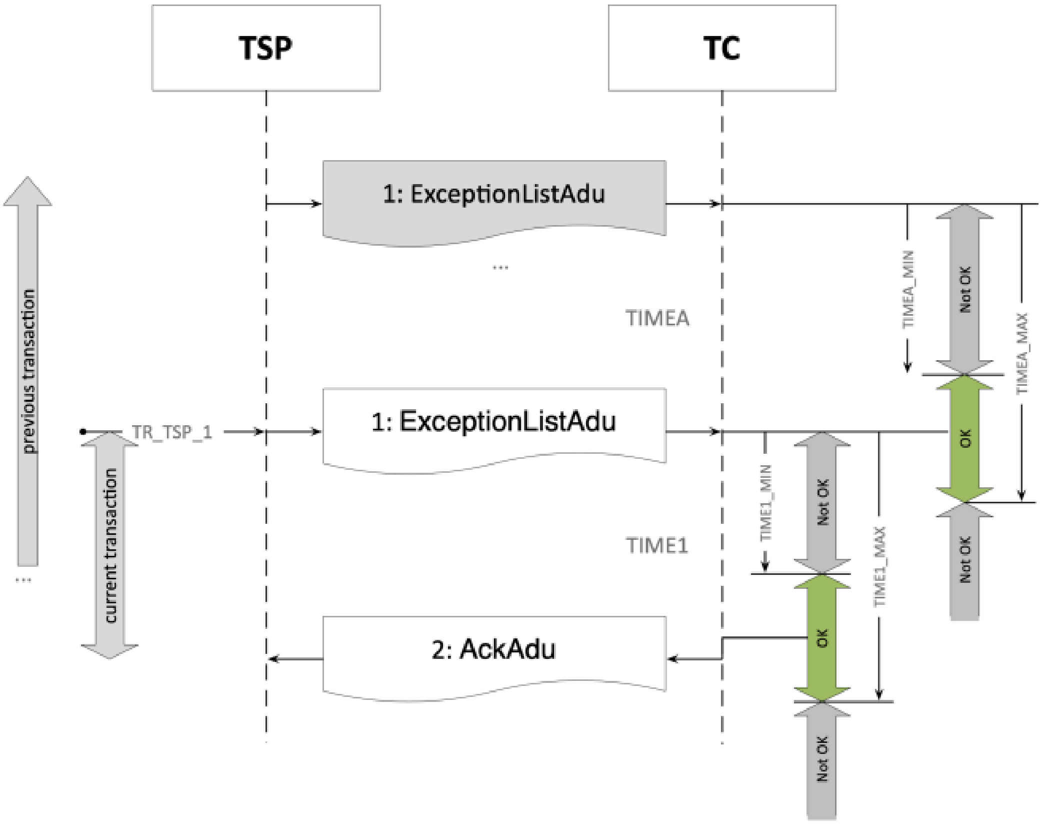


Figure 24. Copy of Figure 9 from EN 16986

1) Full Exception Lists must be transferred every 24 hours:
2) Incremental Exception Lists may be transferred as required.

3) It is possible that full Exception Lists will be too large to be transferred in a single message, in which case they must be sent as multiple chunks (see description of the chunking process in "Message Formats" below).
4) If no chunking is required, each Exception List message will be acknowledged with an *Acknowledge TSP Adu*.
5) In the case where chunking is required, assume a message has "n" chunks:
   a) Chunks 1 to (n-1) are acknowledged at the API level only (i.e., no *Acknowledge TSP Adu* is sent)
   b) Once all n chunks have been received, the entire list will be acknowledged with an *Acknowledge TSP Adu*. Each chunk will be explicitly acknowledged within the *Acknowledge TSP Adu* with a separate *AckTspAdu* data field.
6) Incremental Exception Lists will be acknowledged with an *Acknowledge TSP Adu*.
7) TIMEA_MIN (prEN16986:2023 Table 33 and Figure 9) is defined as 20 hours (TBC) for full exception lists
8) TIMEA_MAX (prEN16986:2023 Table 33) is defined as 28 hours (TBC) for full exception lists
9) TIMEA_MIN (prEN16986:2023 Table 33) is defined as 0 for incremental exception lists
10) TIMEA_MAX (prEN16986:2023 Table 33) is undefined for incremental exception lists
11) TIME1 values (prEN16986:2023 Table 33) are undefined as exception lists will be acknowledged at the API level

6.8.4 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 EN 16986.

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 EN 16986.

In the full White List and Black List messages, the data element *exceptionListEntries* contains a list of all the users identified on the Exception List. As this may be a very large list, the interface includes a mechanism for dividing the message into multiple smaller parts or "chunks". Each chunk will contain a maximum of 6,000 exception list entries. Four additional data elements are included to supporting the chunking:

- *totalExceptionListEntries* – Total number of entries for this Payment Claim
- *exceptionListEntriesFromRange* – The value either "1" (first payload) or next number after previous *exceptionListEntriesToRange* (2nd and subsequent payloads)
- *exceptionListEntriesToRange* – *exceptionListEntriesFromRange* + *countOfExceptionListEntries* – 1
- *countOfExceptionListEntries*- Count of exception list entries in this payload

Note that these data elements are not required in the incremental White Lists and Black Lists, and hence are not included in these messages.

In cases where the chunking mechanism is required, each *apduIdentifier* must be unique as required by the standards. However, the different chunks of the same Exception List must have the same *aduIdentifier*1, which identifies them as being part of the same Exception List.

The use of the chunking mechanism is described in the three examples below.

Example 1: Exception List having one entry (no chunking required):

Data Element	Value
apduIdentifier	12345
aduIdentifier	6789
totalExceptionListEntries	1
exceptionListEntriesFromRange	1
exceptionListEntriesToRange	1
countOfExceptionListEntries	1

Example 2: Exception List having 1000 entries (no chunking required):

Data Element	Value
apduIdentifier	12345
aduIdentifier	6789
totalExceptionListEntries	1000
exceptionListEntriesFromRange	1
exceptionListEntriesToRange	1000
countOfExceptionListEntries	1000

Example 3: Exception List having 10900 entries, so two chunks required:

First message containing chunk 1

Data Element	Value
apduIdentifier	12345
aduIdentifier	6789
totalExceptionListEntries	10900
exceptionListEntriesFromRange	1
exceptionListEntriesToRange	6000
countOfExceptionListEntries	6000

Second message containing chunk 2

Data Element	Value
apduIdentifier	23456
aduIdentifier	6789

totalExceptionListEntries	10900
exceptionListEntriesFromRange	6001
exceptionListEntriesToRange	10900
countOfExceptionListEntries	4900

The actions supported for incremental lists are limited as follows:

- An entry on an incremental whitelist will be added to the White List only. Removing users from a White List can only be achieved at the next full White List update.
- An entry on an incremental Black List will only be removed from the current Black List. Adding users to the Black List can only be achieved at the next full Black List update.

Identification of OBE type

The KmToll Scheme supports both OBE using dedicated hardware and OBE implemented as software on a nomadic platform (e.g., a smartphone), called Type 1 and Type 2 respectively. In order to identify the OBE type, a new data field called "*obeType"* is introduced as part of the "*obeId"* field. *obeType* is defined below:

Data Element	Data type	Description
obeType	Integer	Identified the OBE type in use. It can take the values: "1" – OBE is of type 1 "2" – OBE is of type 2 Values between "3" and "127" are reserved for future use.

6.8.5 Error Handling

API errors will be handled as described in section 5.5.

6.9 EETS Payment Announcement interface

6.9.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS* Payment Announcement – *vn*", where "*vn*" is the version number currently in use.

6.9.2 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).

6.9.3 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:

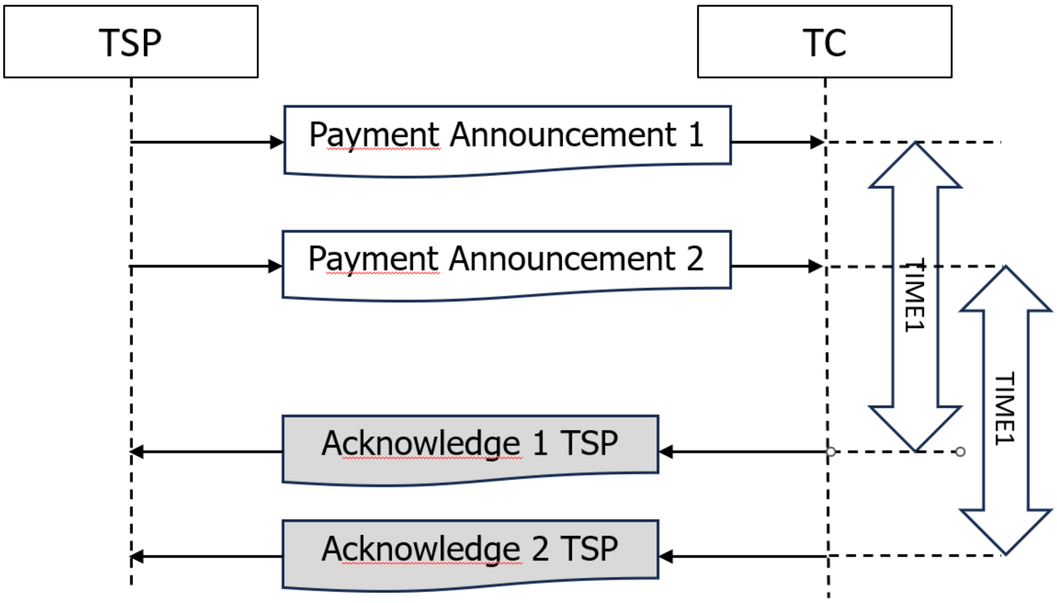


Figure 25. Payment Announcement flows

1. The TSP shall issue a Payment Announcement within an agreed period after making a payment to the TC. This time is contractually defined.
2. The TC will acknowledge the Payment Announcement within TIME1. This is currently defined as 24 hours.
3. The Payment Announcement will be asynchronously acknowledged, i.e., multiple Payment Announcements may be sent in parallel.

6.9.4 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 EN 16986.

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 EN 16986.

Data fields removed from or added to those defined in EN 16986:

1. As ISO 12855:2022 has no pre-defined AduReasonCodes to indicate rejection of the payment announcement, a user defined code of value 10900 has been defined for this purpose. If a future version of ISO 12855 defines specific codes within the pre-defined range of 900 – 999, these may be used in a future version of this specification.

6.9.5 Error Handling

API errors will be handled as described in section 5.5.

6.10 **EETS Contract Issuer List interface**

6.10.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS* Contract Issuer List – *vn*", where "*vn*" is the version number currently in use.

6.10.2 Purpose of the Interface

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

6.10.3 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 – Figure 22 from prEN16986:2023 is reproduced below for reference:

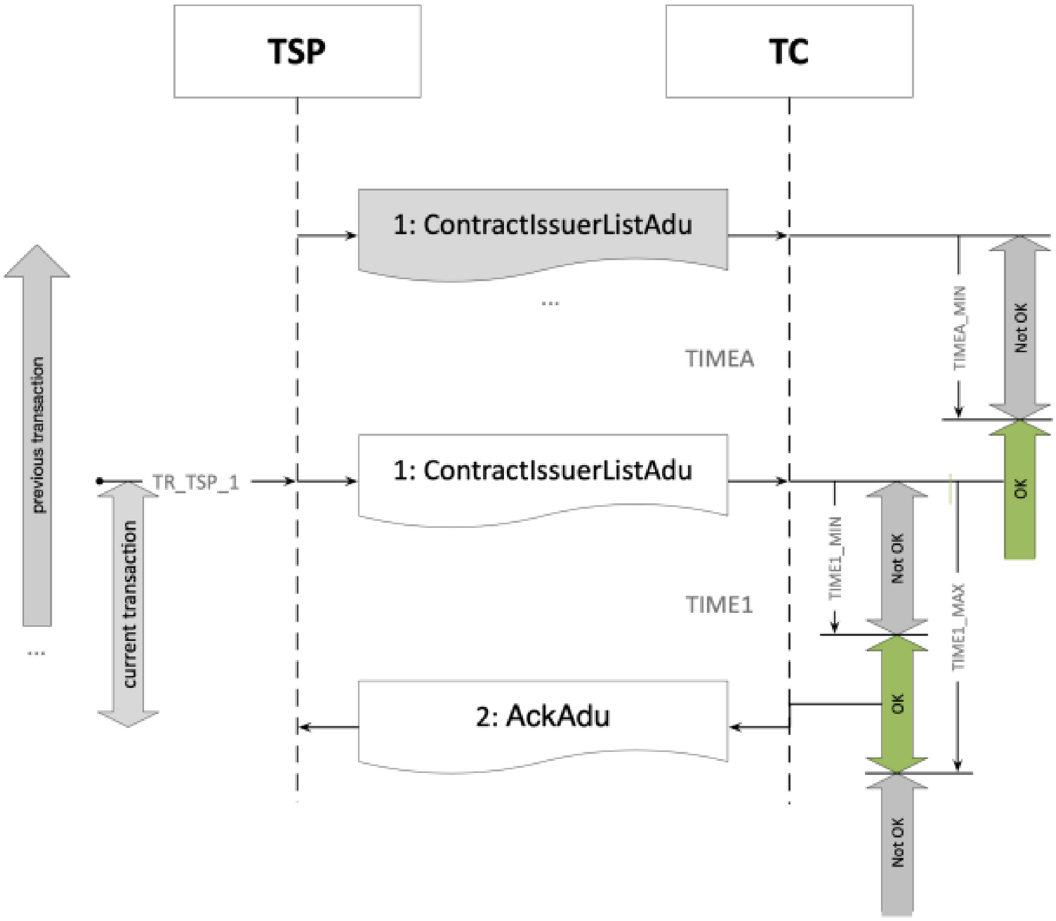


Figure 26. Copy of Figure 12 from EN 16986

1. TIMEA_MIN (prEN16986:2023 Table 49 and Figure 12) is defined as 24 hours (TBC)
2. TIME1_MIN (prEN16986:2023 Table 49) is defined as 0 (TBC)
3. TIME1_MAX (prEN16986:2023 Table 49) is defined as 24 hours (TBC)

6.10.4 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 EN 16986.

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 EN 16986.

Data fields removed from or added to those defined in EN 16986:

1. As ISO 12855:2022 has no pre-defined AduReasonCodes to indicate rejection of the contractIssuerList adu, a user defined code of value 10300 has been defined for this purpose. If a future version of ISO 12855 defines specific codes within the pre-defined range of 300 – 399, these may be used in a future version of this specification.

6.10.5 Error Handling

API errors will be handled as described in section 5.5.

6.11 **EETS CCC Data Response interface**

6.11.1 Identification of the Interface

This interface is identified on the relevant portal by the name "*EETS* CCC Data Response – *vn*", where "*vn*" is the version number currently in use.

6.11.2 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.

6.11.3 Communications Processes

The CCC_Data_Response will be transferred using a REST Web-Service interface as defined in section 4. Acknowledgement will only take place at the API level, i.e., it will not be explicitly acknowledged with an AckAdu.

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

The communications sequence is shown below.

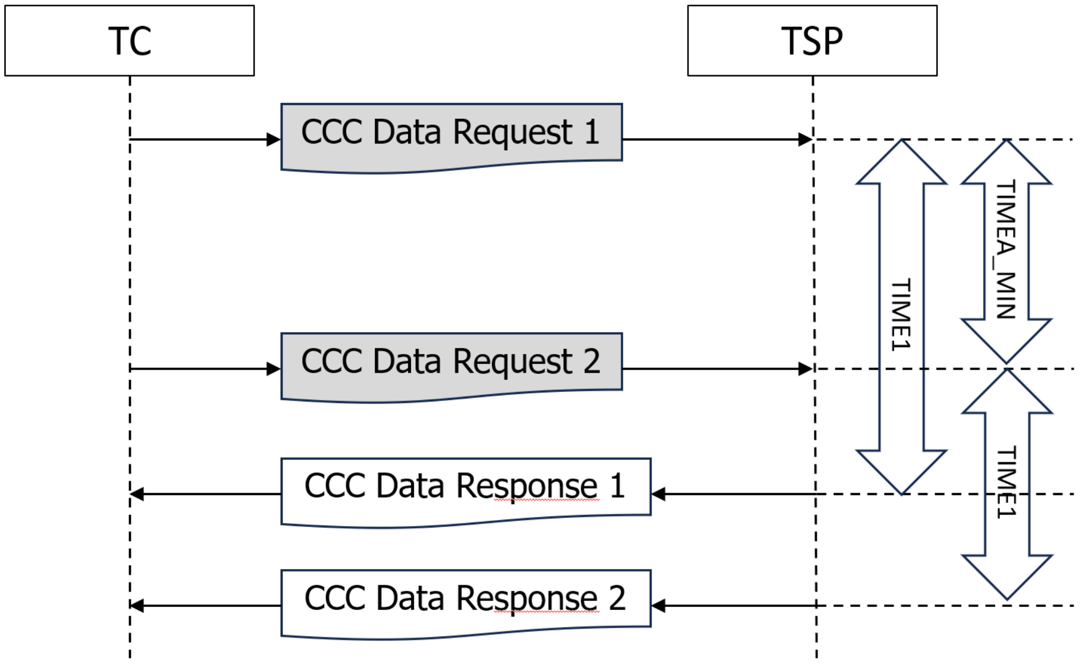


Figure 27. Sequence diagram for CCC Data Request transaction

The TSP is required to respond to the CCC Data Request with a CCC Data Response.

TIME1_MIN, the minimum time to respond to a CCC Data Request is 0.

TIME1_MAX, the maximum time to respond to a CCC Data Request is 15 minutes. If after 15 minutes the TSP does not have the status information available (for example, there has been no connectivity to the OBE), the TSP must respond with a *CCC_Data_Response* with the *cccStatusUnavailable* flag set to 1.

In the case where the TSP has responded with the *cccStatusUnavailable* flag set to 1, the TC will re-request the data up to 24 hours later.

6.11.4 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 EN 16986.

The CCC_Data_Response will contain data regarding the status of the OBE. The following data will be reported:

Data Element	**Description**
timeOfCccStatus	Time in UTC at which the status was recorded
locationOfCccStatus	Location of OBE at time of status, in lat/long
cccContext	Context information recorded in OBE at the time of status, consisting of provider information, contract type and context version.
vehicleLpn	Detailed vehicle license plate information
vehicleClass	Vehicle class as coded in EN 15509:2023 Table A2, attribute 17
vehicleWeightLimits	Vehicle vehicleMaxLadenWeight and vehicleTrainMaximumWeight

vehicleSpecificCharacteristics	Additional vehicle characteristics, consisting of environmental characteristics (euroValue, copValue, euCO2EmissionsClass), engineCharacteristics (ISO/TS 17573-03:2021, table 13) and futureCharacteristics (ISO/TS 17573-03:2021, table 3)
equipmentObuId	As specified in ISO/TS 17573-3
obeStatusHistory	Consisting of: • statusIndicator, a value between "0" and "5" as defined in ISO 12813:2019, Table 5. • timeWhenChanged, Date and time at which the CCC status was last changed to the current status. • timeWhenActivated, Last time the OBE/App started to evaluate current time, place and other parameters • timeWhenObePowered, Exact usage to be defined.

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

6.11.5 Error Handling

API errors will be handled as described in section 5.5.

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 transfers are expected to be infrequent, a manual exchange mechanism (e.g., upload via secure web interface) is envisaged.

6.13 **Actor Table (EXCHANGEACTORTABLE transaction)**

Actor Tables will not be transmitted over a REST interface. The transfer mechanism will be specified by the TC. As these transfers are expected to be infrequent, a manual exchange mechanism (e.g. secure email, secure file share etc.) is envisaged.

6.14 **Compliance Check Communication (CCC) transaction (DSRC)**

6.14.1 Purpose of the Interface

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

6.14.2 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:

6.14.3 Message Formats

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

6.14.4 Error Handling

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

7 Environments & Data Expectation

The TC has different environments for different purpose. The following process diagram below shows the normal flow and the order of environments that will be opened gradually when the process of going from a sandbox environment to full production-ready APIs.

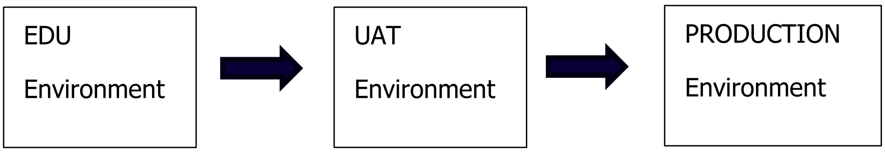


Figure 28. Environments used for development, test and production

To link between the TC and TSP environments, it is crucial that TSP at least has at least a test and a production environment. In the test environment, or in the TC's EDU and UAT environments, no live production data (or GDPR-relevant data) are sent.

EDU Environment (KMToll-EDU):

Purpose:

This environment is used for the accreditation purpose and used as a sandbox to play with the APIs provided by TC. The Developer Portal also has APIs defined in OpenAPI format that the TSP should use to develop their own APIs.

The endpoints can be tested for format validation and connectivity test. The initial release for v1 of the APIs will not contain the Oauth 2.0 requirements; this will be added in v2. The idea of this environment is "connectivity test" and a format validation check against payload to/from TC to TSPs. This environment is a downscaled environment, and not scaled for any kind of performance test.

Data:

No real production data should be sent to these endpoints.

URLs:

Developer Portal: https://portal-edu.vejafgifter.dk

API Example: https://api-edu.vejafgifter.dk/ack-tc/v1/

UAT Environment (KMToll-UAT):

Purpose:

This environment is used for the integration flow test. The idea of this environment is "integration test" and is connected to multiple systems. This environment has similarity environment as production and is scaled for performance test. The test performed on this environment must be coordinated tests which needs approval from TC.

Data:

No real production data should be sent to these endpoints, and only coordinated tests are allowed.

URLs:

Developer Portal: https://portal-uat.vejafgifter.dk

API Example: https://api-uat.vejafgifter.dk/ack-tc/v1/

PRODUCTION Environment (KMToll-PROD):

Purpose:

This environment is the actual production environment and has integration towards all of the TC's sub-suppliers and 3rd parties.

Data:

Only real production data may be sent to these endpoints.

URLs:

Developer Portal: https://portal.vejafgifter.dk

API Example: https://api.vejafgifter.dk/ack-tc/v1/