INTERNATIONAL STANDARD

ISO 17575-2

First edition
2016-01-15

Electronic fee collection — Application interface definition for autonomous systems —

Part 2: **Communication and connection to the lower layers**

Perception du télépéage — Définition de l'interface d'application pour les systèmes autonomes —

Partie 2: Communications et connexions aux couches basses

Reference number
ISO 17575-2:2016(E)

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/TC 204, *Intelligent transport systems*.

This edition of ISO 17575-2 cancels and replaces ISO/TS 17575-2:2010, which has been technically revised. The following changes have been made:

— conversion from a Technical Specification to an International Standard;

— editorial and formal corrections as well as changes to improve readability.

ISO 17575 consists of the following parts, under the general title *Electronic fee collection — Application interface definition for autonomous systems*:

— *Part 1: Charging*

— *Part 2: Communication and connection to the lower layers*

— *Part 3: Context data*

In this edition of the ISO 17575-series the contents of ISO/TS 17575-4:2011 were incorporated into ISO 17575-3:2016. ISO/TS 17575-4:2011 will be withdrawn once ISO 17575-3 has been published.

Introduction

0.1 Autonomous systems

ISO 17575 is a series of standards defining the information exchange between the Front End and the Back End in electronic fee collection (EFC) based on autonomous on-board equipment (OBE). EFC systems automatically collect charging data for the use of road infrastructure including motorway tolls, zone-based fees in urban areas, tolls for special infrastructure like bridges and tunnels, distance-based charging and parking fees.

Autonomous OBE operates without relying on dedicated road-side infrastructure by employing wide-area technologies such as Global Navigation Satellite Systems (GNSS) and Cellular Networks (CN). These EFC systems are referred to by a variety of names. Besides the terms autonomous systems and GNSS/CN systems, the terms GPS/GSM systems and wide-area charging systems are also in use.

Autonomous systems use satellite positioning, often combined with additional sensor technologies such as gyroscopes, odometers and accelerometers, to localize the vehicle and to find its position on a map containing the charged geographic objects, such as charged roads or charged areas. From the charged objects, the vehicle characteristics, the time of day and other data that are relevant for describing road use, the tariff and ultimately the road usage fee are determined.

Two strengths of the autonomous approach to electronic fee collection are its flexibility, allowing the implementation of almost all conceivable charging principles, and its independence from local infrastructure, thereby predisposing this technology towards interoperability across charging systems and countries. Interoperability can only be achieved with clearly defined interfaces, which is the aim and justification of ISO 17575.

0.2 Business architecture

This part of ISO 17575 complies with the business architecture defined in ISO 17573. According to this architecture, the toll charger is the provider of the road infrastructure and, hence, the recipient of the road usage charges. The toll charger is the actor associated with the toll charging role (see Figure 1).

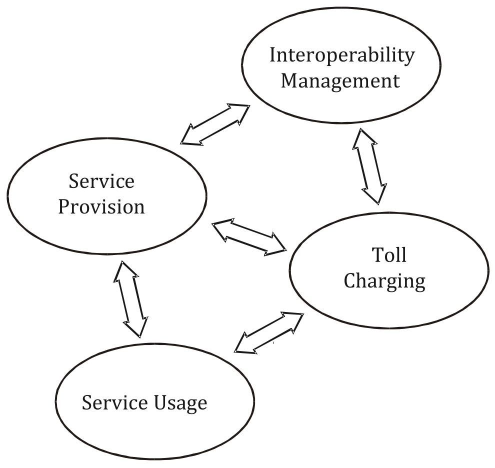

Figure 1 — The role-based model underlying ISO 17575

Service providers issue OBE to the users of the road infrastructure. Service providers are responsible for operating OBE that will record the amount of road usage in all toll charging systems the vehicle passes through and for delivering the charging data to the individual toll chargers. In general, each service provider delivers charging data to several toll chargers and, in general, each toll charger receives charging data from more than one service provider. Interoperability management, as shown in Figure 1, comprises all specifications and activities that define and maintain a set of rules that govern the overall toll charging environment.

0.3 Technical architecture

The technical architecture of Figure 2 is independent of any particular practical realization. It reflects the fact that some processing functionalities can either be allocated to the OBE or to an associated off-board component (proxy). An example of processing functionality that can be realized either on- or off-board is map-matching, where the vehicle locations in terms of measured coordinates from GNSS are associated to geographic objects on a map that either reside on- or off-board. Also, the computation of tariffs can be done with OBE tariff tables and processing, or with an off-board component.

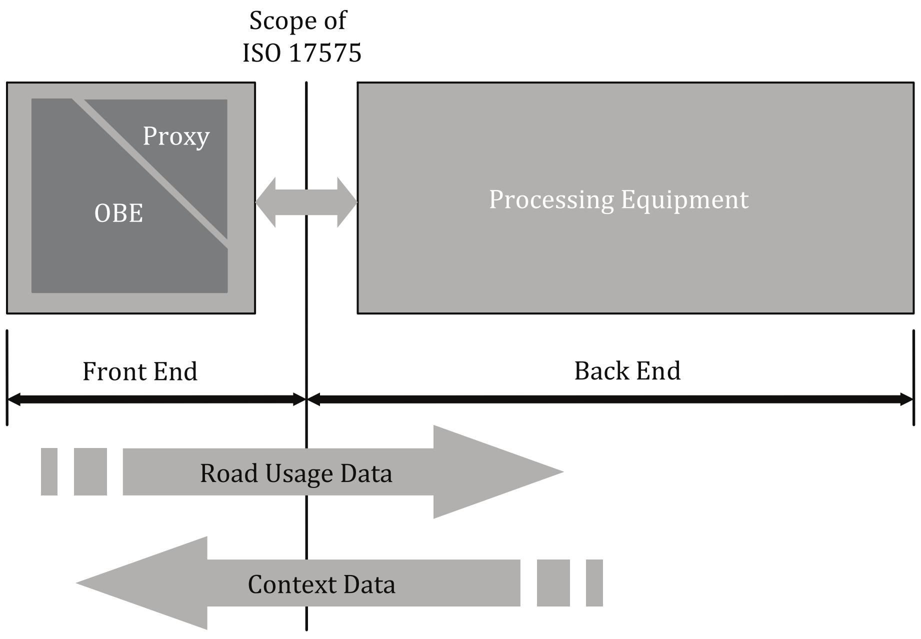

Figure 2 — Assumed technical architecture and interfaces

The combined functionality of OBE and proxy is denoted as Front End. A Front End implementation where processing is predominately on the OBE-side is known as a smart client (or intelligent client, fat client) or edge-heavy. A Front End where processing is mostly done off-board is denoted as thin-client or edge-light architecture. Many implementations between the “thin” and “thick” extremes are possible, as depicted by the gradual transition in the wedges in Figure 2. Both extremes of architectural choice have their merits and are one means where manufacturers compete with individual allocations of functionality between on-board and central resources.

Especially for thin client OBE, manufacturers might devise a wide variety of optimizations of the transfer of localization data between OBE and off-board components, where proprietary algorithms are used for data reduction and data compression. Standardization of this transfer is neither fully possible nor beneficial.

0.4 Location of the specification interface

In order to abstract from, and become independent of, these architectural implementation choices, the primary scope of ISO 17575 is the data exchange between Front End and Back End (see the corresponding vertical line in Figure 2). For every toll regime, the Back End will send context data, i.e. a description of the toll regime in terms of charged objects, charging rules and, if required, the tariff scheme to the Front End, and will receive usage data from the Front End.

It has to be noted also that the distribution of tasks and responsibilities between service provider and toll charger will vary individually. Depending on the local legal situation, toll chargers will require “thinner” or “thicker” data, and might or might not leave certain data processing tasks to service providers. Hence, the data definitions in ISO 17575 may be useful on several interfaces.

ISO 17575 also provides for basic media-independent communication services that may be used for communication between Front End and Back End, which might be line-based or an air-link, and can also be used for the air-link between OBE and central communication server.

0.5 The parts of ISO 17575

Part 1: Charging, defines the attributes for the transfer of usage data from the Front End to the Back End. The contents of charge reports might vary between toll regimes, hence, attributes for all requirements are offered, ranging from attributes for raw localization data, for map-matched geographic objects and for completely priced toll transactions. A toll regime comprises a set of rules for charging, including the charged network, the charging principles, the liable vehicles and a definition of the required contents of the charge report.

Part 2: Communication and connection to lower layers, defines basic communication services for data transfer over the OBE air-link or between Front End and Back End. The data defined in ISO 17575-1 and ISO 17575-3 can but need not be exchanged using the communication stack as defined in ISO 17575-2.

Part 3: Context data, defines the data to be used for a description of individual charging systems in terms of charged geographical objects and charging and reporting rules. For every toll charger's system, attributes as defined in ISO 17575-3 are used to transfer data to the Front End in order to instruct it on which data to collect and report.

0.6 Application needs covered by ISO 17575

The ISO 17575 series of standards

— is compliant with the architecture defined in ISO 17573:2010,

— supports charges for use of road sections (including bridges, tunnels, passes, etc.), passage of cordons (entry/exit) and use of infrastructure within an area (distance, time),

— supports fee collection based on units of distance or duration, and based on occurrence of events,

— supports modulation of fees by vehicle category, road category, time of usage and contract type (e.g. exempt vehicles, special tariff vehicles, etc.),

— supports limiting of fees by a defined maximum per period of usage,

— supports fees with different legal status (e.g. public tax, private toll),

— supports differing requirements of different toll chargers, especially in terms of

  — geographic domain and context descriptions,

  — contents and frequency of charge reports,

  — feedback to the driver (e.g. green or red light), and

  — provision of additional detailed data on request, e.g. for settling of disputes,

— supports overlapping geographic toll domains,

— supports adaptations to changes in

  — tolled infrastructure,

  — tariffs, and

  — participating regimes, and

— supports the provision of trust guarantees by the service provider to the toll charger for the data originated from the Front End.

Electronic fee collection — Application interface definition for autonomous systems —

Part 2: Communication and connection to the lower layers

1 Scope

This part of ISO 17575 defines how to convey all or parts of the data element structure defined in other parts of ISO 17575 over any communication stack and media suitable for this application. It is applicable only to mobile communication links (although wired links, i.e. back office connections, can use the same methodology).

To establish a link to a sequence of service calls initializing the communication channel, addressing the reception of the message and forwarding the payload are required. The definition provided in this part of ISO 17575 includes the required communication medium independent services, represented by an abstract application programming interface (API).

The communication interface is implemented as an API in the programming environment of choice for the Front End (FE) system. The specification of the Back End (BE) API is outside the scope of this part of ISO 17575.

The definition of this API in concrete terms is outside of the scope of this part of ISO 17575. This part of ISO 17575 specifies an abstract API that defines the semantics of the concrete API as illustrated in Figure 3 and its protocol implementation conformance statement (PICS) proforma (see Annex B). An example of a concrete API is presented in Annex C. Where no distinction is made between the abstract and concrete communications APIs, the term "communications API" or just "API" can be used.

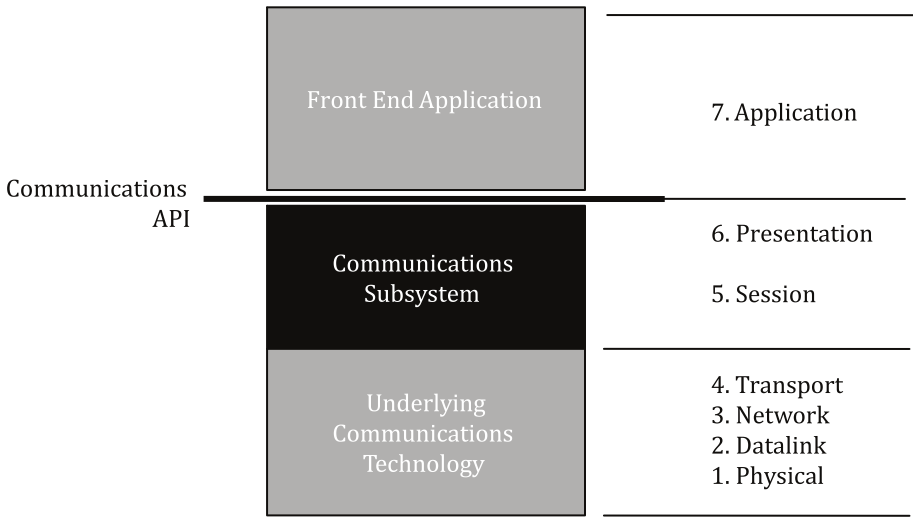

Figure 3 — Scope of this part of ISO 17575

This part of ISO 17575 also provides a detailed specification for the structure of associated API statements, an example on how to implement it and its role in a complex toll cluster such as the EETS (see Annex A to Annex E).

Media selection policies, certificate handling and encryption mechanisms are outside of the scope of this part of ISO 17575.

2 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

2.1
attribute
addressable package of data consisting of a single data element or structured sequences of data elements

[SOURCE: ISO 17575-1:2016, 3.2]

2.2
authenticator
data, possibly encrypted, that is used for authentication

[SOURCE: EN 15509:2014, 3.3]

2.3
Back End
BE
part of a back office system interfacing to one or more *Front Ends* (2.6)

[SOURCE: ISO 17575-1:2016, 3.4]

2.4
data element
coded information, which might itself consist of lower level information structures

[SOURCE: ISO 17575-1:2016, 3.9]

2.5
data integrity
property that data has not been altered or destroyed in an unauthorized manner

2.6
Front End
FE
part of a tolling system consisting of an *OBE* (2.9) and possibly a *proxy* (2.10) where road tolling information and usage data are collected and processed for delivery to the *Back End* (2.3)

[SOURCE: ISO/TS 19299:2015, 3.17]

Note 1 to entry: The Front End comprises the *on-board equipment* (2.9) and an optional *proxy* (2.9).

2.7
Front End application
part of the Front End above the API

2.8
interoperability
ability of systems to exchange information and to make mutual use of the information that has been exchanged

[SOURCE: ISO/IEC/TR 10000-1:1998, 3.2.1, modified.]

2.9
on-board equipment
OBE
all required equipment on-board a vehicle for performing required EFC functions and communication services

Note 1 to entry: Other sub-units should be considered optional.

2.10
proxy
optional part of a *Front End* (2.6) that communicates with external equipment and processes the data received into an agreed format to be delivered to the *Back End* (2.3)

[SOURCE: ISO 17575-1:2016, 3.13]

2.11
service primitive
elementary communication service provided by the application layer protocol to the application processes

Note 1 to entry: The invocation of a service primitive by an application process implicitly calls upon and uses services offered by the lower protocol layers.

[SOURCE: ISO 14906:2011, 3.18, modified — the subject has been deleted.]

2.12
toll service provider
entity providing toll services in one or more toll domains

[SOURCE: ISO 17573:2010, 3.23, modified — the definition has been condensed.]

2.13
toll
charge, tax or duty levied in connection with using a vehicle in a toll domain

[SOURCE: ISO/TS 19299:2015, 3.42, modified — "any" has been deleted from before "charge".]

Note 1 to entry: The definition is the generalization of the classic definition of a toll as a charge, a tax, or a duty for permission to pass a barrier or to proceed along a road, over a bridge, etc. The definition also includes fees regarded as an (administrative) obligation, e.g. a tax or a duty.

2.14
toll charger
entity which levies toll for the use of vehicles in a toll domain

[SOURCE: ISO 17573:2010, 3.16, modified — "legal" has been deleted from before "entity" and "the use of" has been added.]

3 Abbreviated terms

For the purpose of this document, the following abbreviated terms apply unless otherwise specified.

ADU	Application data unit (ISO 14906)
APDU	Application protocol data unit (ISO 14906)
AP	Application process (ISO 14906)
API	Application programming interface
ASN.1	Abstract Syntax Notation One (ISO/IEC 8824-1)

BE	Back End
CN	Cellular network
EID	Element identifier (ISO 14906)
FE	Front End
GNSS	Global Navigation Satellite System
OBE	On-board equipment (ISO 14906)
VAT	Value added tax

4 EFC Front End communication architecture

4.1 General

A communications subsystem is required to establish the communication link between a Front End (FE) and a Back End (BE) Application. It provides data transport for the tolling FE Application via the communications session that takes part across the line shown in Figure 4. In cases where a proxy is present in the FE system, the communications subsystem defines the communications between the BE and the proxy. The link between the proxy and the on-board equipment (OBE) is out of the scope of this part of ISO 17575. In cases where no proxy is present (the "smart client"), the communications subsystem defines the communications between the OBE and the BE.

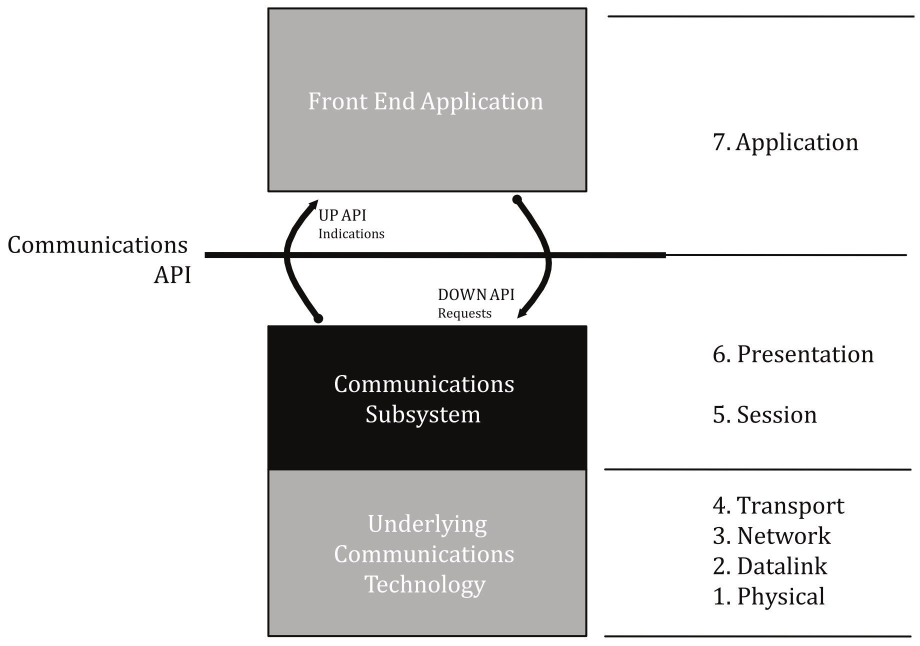

Figure 4 — Relationship between Application and Protocol Stack

The communications subsystem is further subdivided into two distinct components. The communications API itself offers communications functionality to the FE Application. Below this is the underlying communications technology, which provides the functionality that the API abstracts. Although the API is independent of the underlying technology, it does place a number of functional demands upon it. For this reason, the functional requirements on the underlying communications technology are listed in 6.2.

Some underlying technologies are more capable than others. In cases where a very capable technology is in use, the code interfacing the API to the underlying technology will serve little more function than a simple pass through. For more simplistic transport layer technologies the communications subsystem will have to do considerably more.

It is expected that these APIs will be “reflected” in the BE such that FEs and BEs can communicate over arbitrary bearer infrastructures. Details of the abstract API definitions are specified in Annex A. The specification of the BE API is outside the scope of this part of ISO 17575.

4.2 Relationship with the overall EFC architecture

The communications API provides the lower layers of the interface shown in Figure 5. The API has no semantic knowledge of the application data units (ADUs) it is carrying. It does differentiate between “standard specific” and “arbitrary” ADUs but it has no semantic knowledge about what these mean and simply carries them as transparent octet streams atop an arbitrary communication bearer that is selected at runtime.

5 EFC communication services (functions)

5.1 General concept

The API carries two “types” of message (ADU): structured elements relating directly to the definitions in other parts of ISO 17575, and unstructured elements, which are outside of the scope of this part of ISO 17575 and receive no further consideration within it.

It is outside the scope of this part of ISO 17575 to identify the data elements for transmission and the associated payload.

NOTE 1 The `protocolVersion` (part of `ChargeReport`) as defined in ISO 17575-1:2016 and `protocolVersion` and `tollContext` (both part of `aduHeader`) as defined in ISO 17575-3:2016 can be useful when implementing specific transaction(s).

The abstract API for the communications services can be implemented in any programming environment that defines the concept of event delivery, allowing the API to report information or to deliver results of operations to the FE Application. The general sequence of events is

— initialize and parameterize the communications interface,

— establish a communications session,

— transfer data in the context of the session,

— terminate the communications session, and

— de-initialize the communications interface.

In a normal case, the FE Application will initialize (a number of) communications interfaces when it first starts. An active session is then established either as a direct action by the FE Application or in response to an incoming request for a session from the BE. The flow of events through the lifetime are shown in 5.2 to 5.7 and relevant definitions are given in Annex A.

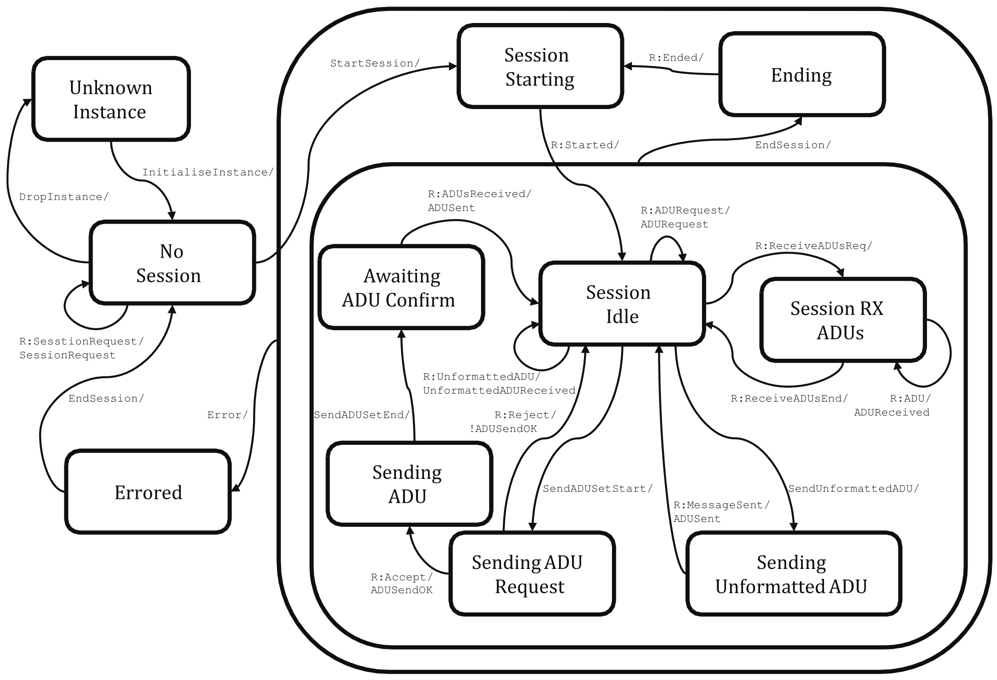

Figure 5 — Session state diagram

API calls down the communications stack fall into two classes: synchronous and asynchronous. Synchronous calls giving results immediately are limited to those API calls that can quickly return a result (`InitialiseInstance` and `GetParameter`). Other API calls are asynchronous and return their results via the event mechanism that is initialized during `InitialiseInstance`.

NOTE 2 This prevents threads from locking while waiting for information to be returned and so reduces the requirement for multithreaded programming, which is often inappropriate for embedded applications such as those typically seen in the field of application.

Figure 6 shows an example of the relationship and interactions between different states, which are visible across the API.

EXAMPLE Illustration of message flow for cases where the FE Application wishes to establish a session, exchange some information with the BE, and them terminate the session. The references to API Events refer to the definition in Annex A.

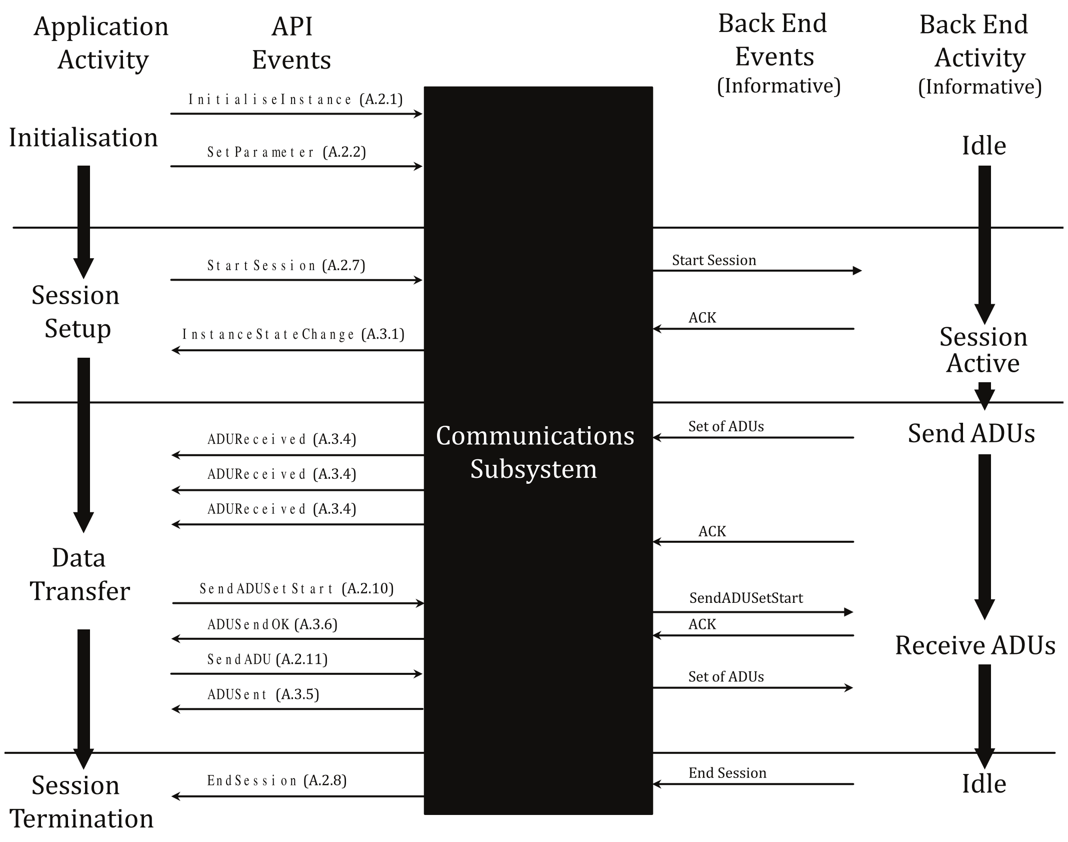

Figure 6 — Example message flow and session state chart

5.2 Initialization phase

5.2.1 General

The FE Application shall initialize the communications interfaces that it wishes to make use of by means of calls to `InitialiseInstance`. For each instance created, the FE Application defines which underlying communications stack is to be used. More than one interface may be used at the same time and the choice between interfaces is the decision of the FE Application. The FE Application also provides a set of event reception capabilities, which are used by the API to inform the FE Application of status changes.

Any additional parameterization of the instance that is required is performed by repeated calls to `SetParameter`. A set of parameters are recognised by the communications API itself and those which are not are passed through transparently to the underlying stack. Queries for the existing state of parameters may be made via calls to `GetParameter`.

Once this process has been completed, the FE Application now has access to a (set of) communication instances. State changes for events that occur on any of these interfaces will be delivered by means of an `InstanceStateChange` event notification.

A chart showing the relationship and interactions between each of these states is shown in the example given in Figure 6.

5.2.2 Incoming (BE to FE Application) session request

The BE may wish to establish communications with the FE. In this case, it performs whatever actions are appropriate for the particular communications technology concerned. This will result in the FE Application being informed of the request at some point in time by means of a `SessionRequested` event with a datum SessionHandle indicating the identifier that the FE Application should use for the session. From here, the process is the same as for an outgoing session establishment, as shown in 5.2.3.

NOTE The FE Application might defer a session request for an arbitrary length of time, subject to operational constraints.

5.2.3 Outgoing (FE Application to BE) session establishment

The FE Application, either asynchronously or via the process in 5.2.2, requests a session within the chosen communication context by means of StartSession, parameterized with information about the session to be established. This returns immediately while also starting the process of creating the session within the communications technology layers below.

Once the session is established, an InstanceStateChange event informs the FE Application. The session is now active and communications between the FE Application and the BE can take place within its context.

5.3 Point-to-point communication service primitives

5.3.1 General

Once the session is active, ADUs can be sent to the BE by the FE Application. Both structured (i.e. context aware) and unstructured (context unaware) communications capabilities are provided. All ADUs (structured and unstructured) are opaque to the communications layers and the distinction is only made to avoid the overhead of higher layer demultiplexing.

Within the context of the communication session the FE Application is considered the master and has control of the session.

5.3.2 Unstructured messages (ADUs)

A facility is provided to transit unformatted ADUs over the communications infrastructure. This facility is typically used for the purpose of software upgrades and various other manufacturer-specific information exchanges.

ADUs are sent via calls to SendUnformattedADU. The maximum size of ADU that can be sent via this mechanism is available via a parameter from the API. Once the message has been transmitted, an indication shall be provided (ADUSent) that it has been received successfully by the remote end and that further transactions are possible.

5.3.3 Structured messages (ADUs)

Structured ADUs shall be sent in sets. A set shall be constructed for transmission via a call to SendADUSetStart. This requests a permission to enter into structured ADU-sending mode from the far end. ADUs are added to the set to be sent via calls to SendADU, which shall return the amount of space remaining in the transmit buffer. The set shall be closed with a call to SendADUSetEnd. Once transmission has been competed the FE Application shall be informed via an ADUSent event.

This is an implementation optimization option as ADUs are transmitted while the set is still being assembled. This means that the available buffer space indicated by the return from SendADU should be trusted more than local calculations in the FE Application as more space can be made available when elements are transmitted.

5.4 Session end

When it is time for a session to end, the FE Application shall make a call to EndSession and provide a suitable reason code. This shall start the process of ending the session within the underlying communications technology. Session termination may not be immediate and transactions that are in process will be completed before closure occurs. FE Application implementers shall expect to have to handle activities from the open session until an `InstanceStateChange` to state `STNoSession` is received.

5.5 Session failure

A session may end through no fault of the BE or the FE Application by means of intervening communications infrastructure failure. In this instance, the FE Application shall be informed of the fact by means of an `InstanceStateChange` to `STErrored`. Any ADUs sent by the FE Application for which it has not received an `ADUSent` confirmation shall be assumed to have failed.

The communications technology shall not attempt to automatically re-establish the session; that is the responsibility of the FE Application. If the communications session was as a result of the BE request then the session shall be re-established at the first convenient opportunity. If the session was as a result of an FE Application event then the decision to re-establish is left to the FE Application.

Note that some communications technologies are, by their nature, intermittently available. To support these a `StackAvail` call is provided to allow the FE Application to make intelligent choices between available infrastructures. If a medium fails during a session this shall be indicated according to the processes noted above.

5.6 Security considerations

All communications should be encrypted. Certificate handling and encryption mechanisms are outside of the scope of this part of ISO 17575.

5.7 Media selection options

Media may be selected between means of having several instances instantiated with independent communication technologies. The capabilities of each medium can be determined via `GetParameter` calls, and the local availability of any specific medium in an active instance can be determined through calls to `StackAvail`.

6 The use of a communication stack

6.1 General

This part of ISO 17575 allows for multiple communications technologies to be used at the same time, and for the FE Application to be able to select amongst them the one most appropriate for the specific communication to take place. Implementations shall use this capability for "multi-mode" OBEs to allow for the use of different technologies depending upon the urgency of the communication, their capability and the extent of the available infrastructures.

There are, however, a number of underlying capabilities that any communication technology (or rather, stack that sits on top of the technology) needs to provide.

6.2 Requirements for an underlying communication technology

This part of ISO 17575 allows for a wide range of underlying communications technologies. However, the following list of properties for these stacks shall be provided:

— capability of supporting or emulating a "session";

— capability of reliably transferring ADUs, in sequence, bidirectionally across the link;

— capability of reporting the safe receipt of ADUs at the remote end of the link;

— capability of transporting data elements of arbitrary length between the FE and the BE;

- capability of establishing a session from the FE to the BE;
- some means of signalling a demand from the BE to the FE Application that the BE wishes a session to be established;
- capability of reliably detecting the loss of a session and of delivering that information to the communications API;
- delivery of ADUs across the link in a timely fashion;
- capability of carrying an appropriate amount of data.

6.3 Mobile terminated calls

The approach taken within this part of ISO 17575 never requires an incoming, in-band communication port to be open. If a BE wishes to make a connection to an FE Application for any purpose, it can use out-of-band signalling; these mechanisms are outside the scope of this part of ISO 17575. The range of out-of-band signalling options is considerable (e.g. SMS messages, push button on the unit, broadcast signal, etc.). A consequence of this approach, to use out-of-band signalling, is that there is never any requirement for an IP addressable end point for a mobile terminated call. This avoids security issues and also simplifies issues of network address translation and private subnets, since the FE Application will only ever need to make an outgoing connection.

NOTE A consequence of this approach is that there is never any requirement for an IP addressable end point for a mobile terminated call. This avoids the security issues discussed above but also simplifies issues of network address translation and private subnets, since the FE Application will only ever need to make an outgoing connection.

Annex A
(normative)

Abstract API definition

A.1 General

This annex defines the properties of commands and data flow at the API between the EFC specific software – the Front End (FE) Application – and the interface layer of the communications subsystem of the platform used to run the EFC application, see also Figure 4. Examples of how this interface can be realized are provided in Annex C and Annex D.

The communications API consists of a "down" API, from the FE Application to the communications stack, and an "up" API, from the communications stack to the FE Application. Each will be considered in turn.

A.2 Down API (FE Application to communications stack)

A.2.1 InitialiseInstance

Purpose:	Initialize the communications API for use. The interface is initialized in `STNoSession` condition.	
Takes:	`StackID`	Underlying communications stack to be employed
	`Callbacks`	Reference to the event handlers to be used for this instance, see 5.2
Precondition:	The interface must not have been previously initialized	
Returns:	A handle to the instance of the API. This handle is used for all communications	
Errors:	Returns an invalid instance if it was not possible to create the instance	

A.2.2 SetParameter

Purpose:	Set a parameter for this instance of the API. All parameters and values are expressed as strings.	
Takes:	`Instance`	The instance to which this parameter relates
	`Parameter`	The parameter to be set
	`value`	The value to give to the parameter
Precondition:	A valid instance is required	
Returns:	Error code	
Errors:	`ERNoError, ERNotSet`	

A.2.3 GetParameter

Purpose: Get a parameter value for this instance of the API

Takes:	`Instance`	The instance to which this parameter relates
	`Parameter`	The parameter to be set
Precondition:	A valid instance is required	
Returns:	The specified parameter value as a string	
Errors:	Invalid string if the parameter is not known	

A.2.4 DeleteParameter

Purpose:	Delete a parameter from this instance of the API	
Takes:	`Instance`	The instance to which this parameter relates
	`Parameter`	The parameter to be deleted
Precondition:	A valid instance is required	
Returns:	Error code	
Errors:	`ERNoError, ERNotSet`	

A.2.5 StackAvail

Purpose:	Indicate if the communications stack is currently available	
Takes:	`Instance`	The instance to which this check relates
Precondition:	A valid instance is required	
Returns:	Boolean indicating if the stack is currently available	
Errors:	Will return `FALSE` in the case of an error	

A.2.6 DropInstance

Purpose:	Delete an API interface	
Takes:	`Instance`	The instance to which this drop request relates
	`Severity`	Speed with which this interface should be removed (`SENormal`, `SEUrgent`, `SEUnconditional`)
Precondition:	A valid instance in the `STNoSession` state (for `SENormal` and `SEUrgent`) is required	
Returns:	Error code	
Errors:	`ERUnknownInstance, ERBadState, ERNoError`	

A.2.7 StartSession

Purpose: Start the process to establish a session in the context of this interface (i.e. using the established communications channel and parameters). Session establishment is defined by an event indicating a change to `STSessionIdle` when the session is in place. Parameterization of the session is via the `SetParameter`/`GetParameter` methods.

Takes:	`Instance`	The instance within which this session request should be met
	`Reason`	Reason for this session establishment
	`SessionHandle`	Any session handle information that is required

Precondition: Active instance and no active session already. Correct parameterization in place

Returns: Error code

Errors: `ERInSession, ERNoInstance, ERUnknownEndpoint, ERInSession, ERSessionFailed, ERNoError`

A.2.8 EndSession

Purpose: To start the process to end an active session in the context of this instance. Session end is confirmed via the state change event to state `STNoSession`. Under normal conditions any outstanding transactions in the context of the session will be completed before the close is performed.

Takes:	`Instance`	The instance within which this session is to be terminated
	`Reason`	The reason for this session termination

Precondition: Existence of an active session

Returns: Error code

Errors: `ERInSession, ERNoInstance, ERNoError`

A.2.9 SendUnformattedADU

Purpose: To send an unformatted ADU to the other end of the communication link. In the context of this document an unformatted ADU is one that has no special meaning to the communications API and is processed by vendor specific code. Confirmation of receipt at the far end is by means of an event indication.

Takes:	`Instance`	The instance within which this message is to be sent
	`MessageLen`	The length of the message, in octets, to be sent
	`Message`	The message itself

Precondition: Existence of an active session with no transaction in progress

Returns: Error code

Errors: `ERBadState, ERNoInstance, ERSessionFailed, ERNoError`

A.2.10 SendADUSetStart

Purpose: To send a request to start sending a set of structured ADUs to the remote end of the communication link. Confirmation of acceptance of transfer is by means of an event `ADUSendOK.`

Takes:	`Instance`	The instance within which this message is to be sent

Precondition: Existence of an active session with no transaction in progress

Returns: Error code

Errors: `ERNoInstance, ERBadState, ERSessionFailed, ERNoError`

A.2.11 SendADU

Purpose: To add an ADU to the set to be sent to the remote end of the communication link

Takes:

`Instance`	The instance within which this message is to be sent
`ElementLen`	The length of the element, in octets, to be sent
`Element`	The element itself

Precondition: An active session exists with no transaction in progress and an element set is open for construction (see A.2.10)

Returns: Number of octets remaining in the transmission buffer

Errors: Returns `0` in case of error

A.2.12 SendADUSetEnd

Purpose: To conclude the set of structured ADUs (as defined in other parts of ISO 17575) to be sent to the remote end of the communication link. Far end reception of the elements is confirmed by means of the event `ADUSent`.

Takes: `Instance` The instance within which this message is to be sent

Precondition: An active session exists with no transaction in progress and an element set is open for construction (See A.2.10)

Returns: Error code

Errors: `ERBadState, ERSessionFailed, ERNoError`

A.2.13 CommsQuery

Purpose: To poll for the current state of the communications instance

Takes: `Instance` The instance to which this poll request relates

Precondition: A valid instance is required

Returns: The current state of the session:

`STUnknownInstance`	The instance is invalid
`STNoSession`	No session is in progress
`STStarting`	The session is starting
`STSessionIdle`	Session open and idle
`STSendingADU`	Sending ADUs
`STSendingADURequest`	Requesting permission to send ADUs
`STSendingUnformattedADU`	Sending an unformatted ADU
`STAwaitingADUConfirm`	Awaiting confirmation of ADU reception

	`STSessionRxADU`	Receiving ADUs from far end
	`STErrored`	In error state
	`STEnding`	In the process of ending
Errors:	`STUnknownInstance`	

NOTE The error `STUnknownInstance` is reported as a state.

A.3 Up API (communications stack to FE Application)

A.3.1 InstanceStateChange

Purpose: To indicate to the FE Application that a change has occurred in the state of a communications instance

Receives:	`Instance`	The instance to which this indication relates
	`OldState`	The old state of the instance (as per A.2.13)
	`NewState`	The new state of the instance (as per A.2.13)

A.3.2 UnformattedADUReceived

Purpose: To indicate to the FE Application that an unformatted ADU has been received. After reception of this event the Back End (BE) will automatically be informed that the FE Application has received the ADU satisfactorily.

Receives:	`Instance`	The instance to which this indication relates
	`UnformattedMessageLen`	The length of the incoming message
	`UnformattedMessage`	The message itself

A.3.3 ADURequest

Purpose: To indicate to the FE Application that the BE is requesting specific element(s) from it. The FE Application should start the processes required to send an ADU set to deliver the requested elements to the BE.

Receives:	`Instance`	The instance to which this indication relates
	`Elements`	Elements to be returned to the BE

A.3.4 ADUReceived

Purpose: To deliver to the FE Application an element from the BE structured in accordance with other parts of ISO 17575. After reception of this event the BE will automatically be informed that the FE Application has received the ADU satisfactorily.

Receives:	`Instance`	The instance to which this indication relates
	`Element`	The element received

NOTE If several elements are combined into a single communications layer message then an indication of successful receipt will only be sent when the final element is delivered. This could result in re-transmission of some elements when communications are lost.

A.3.5 ADUSent

Purpose:	To confirm to the FE Application that an ADU or ADU set has been successfully received by the BE	
Receives:	`Instance`	The instance to which this indication relates

A.3.6 ADUSendOK

Purpose:	To confirm to the FE Application that it may now proceed to send a set of elements	
Receives:	`Instance`	The instance to which this indication relates
	`CanSend`	Flag indicating if send request was accepted

A.3.7 SessionRequest

Purpose:	To alert the FE Application that this instance has detected a session request. Parameters for the session may be claimed via `GetParameter`.	
Receives:	`Instance`	The instance that generated this request
	`Handle`	The handle to be used for establishing the session

Annex B
(normative)

Protocol implementation conformance statement (PICS) proforma

B.1 General

This annex contains the Protocol Implementation Conformance Statements (PICS) proforma to be used for on-board equipment (OBE) implementation of the communication services as defined in Clauses 4, 5 and 6.

NOTE Media selection policies are outside the scope of this part of ISO 17575.

B.2 Transactions support

B.2.1 General

This clause applies to implementations for Front End communications APIs, see Tables B.1 and B.2. The appropriate communication services for the Back End shall use the same statements with the view from the Back End.

B.2.2 Support of the down API

Table B.1 — Support of the down API

Description	Dispensation
API supports `InitialiseInstance`	Yes/No
`StackID` is supported	Yes/No
`Instance handle` will be provided	Yes/No
Invalid Instance returned when not possible to create an instance	Yes/No
API supports `SetParameter`	Yes/No
`Instance` is applied	Yes/No
`ERNoError` returned on successful parameter setting	Yes/No
`ERNotSet` returned on failure to set parameter	Yes/No
Maximum length of parameter handles	Number
Maximum length of value strings	Number
Parameter is stored following call	Yes/No
API supports `GetParameter`	Yes/No
`Instance` is applied	Yes/No
`Parameter` is applied	Yes/No
`Value` is returned as a string when input parameters are valid	Yes/No
An invalid string is returned when input parameters are not valid	Yes/No
API supports `DeleteParameter`	Yes/No
`Instance` is applied	Yes/No
`Parameter` is applied	Yes/No

Table B.1 *(continued)*

Description	Dispensation
ERNoError is returned when parameter is successfully deleted	Yes/No
ERNotSet is returned when parameter is not successfully deleted	Yes/No
ERNotSet is returned when Instance is not valid	Yes/No
ERNotSet is returned when parameter is not found/invalid	Yes/No
API supports DropInstance	Yes/No
Instance is applied	Yes/No
Severity is considered	Yes/No
ERUnknown instance is returned when an unknown instance is provided	Yes/No
ERBadState is returned when Severity is not SEUnconditional and not in STNoSession state	Yes/No
ERNoError is returned when instance is successfully dropped	Yes/No
API supports StartSession	Yes/No
Instance is applied	Yes/No
Reason is applied	Yes/No
SessionHandle is applied	Yes/No
ERInSession is returned if already in a session	Yes/No
ERNoInstance is returned if the instance is invalid	Yes/No
ERUnknownEndpoint is returned if the parameterised endpoint is not known	Yes/No
ERSessionFailed is returned if session establishment fails immediately	Yes/No
ERNoError is returned if session establishment starts correctly	Yes/No
API supports EndSession	Yes/No
Instance is applied	Yes/No
Reason is applied	Yes/No
ERInSession is returned if the session is not in STSessionIdle and Reason is not RERemoteDrop	Yes/No
ERNoInstance is returned if the instance is invalid	Yes/No
ERNoError is returned if the EndSession starts correctly	Yes/No
API supports SendUnformattedADU	Yes/No
Instance is applied	Yes/No
MessageLen is applied	Yes/No
Message is considered	Yes/No
ERBadState is returned if the session is not in the state STSessionIdle	Yes/No
ERNoInstance is returned if the instance is invalid	Yes/No
ERSessionFailed is returned if the session has failed	Yes/No
ERNoError is returned if progress is normal	Yes/No
API supports SendADUSetStart	Yes/No
Instance is supported	Yes/No
ERNoInstance is returned if the instance is invalid	Yes/No
ERBadState is returned if the instance is not in the state STSessionIdle	Yes/No
ERSessionFailed is returned if the session has failed	Yes/No
ERNoError is returned if the progress is normal	Yes/No

Table B.1 *(continued)*

Description	Dispensation
API supports `SendADU`	Yes/No
`Instance` is applied	Yes/No
`ElementLen` is considered	Yes/No
`Element` is considered	Yes/No
`0` is returned in case of error	Yes/No
Remaining space in the transmit buffer is returned in case of no error	Yes/No
API supports `SendADUSetEnd`	Yes/No
`Instance` is considered	Yes/No
`ERBadState` is returned if the session is not in the state `STSendingElements`	Yes/No
`ERSessionFailed` is returned if the session has failed	Yes/No
`ERNoError` is returned if the progress is normal	Yes/No
API supports `CommsQuery`	Yes/No
`Instance` is considered	Yes/No
`STUnknownInstance` is returned if the instance is unknown	Yes/No
The current state of the instance is returned in normal progress	Yes/No
API supports `StackAvail`	Yes/No
`Instance` is considered	Yes/No
`FALSE` is returned if the instance is invalid	Yes/No
`FALSE` is returned if the stack is not available	Yes/No
`TRUE` is returned if the stack is available	Yes/No

B.2.3 Support of the up API

Table B.2 — Support of the up API

Description	Dispensation
API supports `InstanceStateChange` **event**	Yes/No
`Instance` is delivered	Yes/No
`OldState` is delivered	Yes/No
`NewState` is delivered	Yes/No
Event is generated whenever an externally visible state change is generated	Yes/No
API supports `UnformattedADUReceived` **event**	Yes/No
`Instance` is delivered	Yes/No
`UnformattedMessageLen` is delivered	Yes/No
`UnformattedMessage` is delivered	Yes/No
Event is generated whenever an `UnformattedMessage` is received	Yes/No
API supports `ADUReceived` **event**	Yes/No
`Instance` is delivered	Yes/No
`Element` is delivered	Yes/No
Event is generated whenever an element is received	Yes/No
API supports `ADUSent` **event**	Yes/No
`Instance` is delivered	Yes/No

Table B.2 *(continued)*

Description	Dispensation
Event is generated whenever a message has been acknowledged by the far end	Yes/No
API supports `ADUSendOK` **event**	Yes/No
`Instance` is delivered	Yes/No
`CanSend` is delivered	Yes/No
Event is generated whenever far end has indicated its ability to receive elements	Yes/No
`CanSend` is `TRUE` when far end is able to receive elements, `FALSE` otherwise	Yes/No
API supports `SessionRequest` **event**	Yes/No
`Instance` is delivered	Yes/No
`Handle` is delivered	Yes/No
Event is generated when a stimulus is detected to indicate remote end wants to establish a session	Yes/No

B.3 Use of communication stacks

B.3.1 General

This clause applies to both Front End and Back End, see Table B.3.

B.3.2 Supported communication stacks

Table B.3 — Supported communication stacks

Description	Value(s)
Communication supports the use of TCP/IP v4	Yes/No
Communication supports the use of TCP/IP v6	Yes/No
Communication originates on IP port number	Any/Specify
Communication terminates on IP port number	Specify
The Front End accepts fully qualified domain names as end point	Yes/No
The Front End closes logical session automatically after	Specify in seconds

B.4 Front End storage capacity

B.4.1 General

This clause only applies to Front End see Tables B.4, B.5 and Table B.6.

B.4.1 Storage capacity for modules and contact details

Table B.4 — Storage capacity for modules and contact details

Description	Maximum value or range
Maximum single message size	
Maximum number of instances communication sessions in parallel	
Maximum storage available for queuing of messages	
Maximum number of keys/trust certificates available	
Maximum number of elements that can be transmitted in a single transaction	

B.4.2 Generic values

Table B.5 — Generic values

Description	Maximum value or range
Integers (minimum and maximum value)	
Strings (maximum size)	
Parameter maximum length	
Parameter value maximum length	

B.4.3 Security of communication

Table B.6 — Security of communication

Description	Dispensation
All communication is over encrypted channels	Yes/No
Specify key length for encryption	Number
Specify encryption technology used	Text
Symmetric or Asymmetric keys?	Symmetric/Asymmetric

Annex C
(informative)

API requirements

C.1 General

This annex presents an example of a concrete API. It outlines the requirements that drive the definition of the communications API.

C.2 Non-functional requirements

The communications subsystem is required, in a standards-oriented manner, to address how a link is established, data is transferred over it and it is cleared at the end of the transaction.

The communications subsystem shall establish a session asynchronously when suitable network connectivity is available and when a request for a session is outstanding.

The communications subsystem API shall be communications technology independent. The API places requirements upon the underlying technology, as presented in 6.2.

C.3 Functional requirements

The link shall be established on demand by the Front End (FE) Application to the Back End (BE).

A means shall be provided by which the BE can establish a connection to the FE Application.

NOTE The connection establishment time is undefined (e.g. the FE might be powered down).

The link shall be organized on a “session” paradigm. There will be distinct phases of activity relating to link setup, operation and teardown.

The API shall provide positive indication to the FE Application of the current link state.

Temporary outages of the link shall be transparent to the FE Application.

When a communications session is lost it shall not be automatically re-established by the communications system.

The API shall provide mechanisms to allow the BE to request specific information elements (as defined in other parts of ISO 17575 and using the definitions therein) from the FE Application.

The API shall provide mechanisms to deliver specific information elements (as defined in other parts of ISO 17575 and using the definitions therein) to the BE.

The abstract API shall be capable of carrying manufacturer-specific information, which is outside of the scope of this part of ISO 17575.

The API shall provide positive indication to the FE Application that an ADU has definitely been delivered to the BE.

All communications to and from the remote end shall be encrypted.

Annex D
(informative)

Examples of definitions for appropriate languages

D.1 General

The API has been deliberately designed to be language neutral, because there are many different environments in which it will be used. In general, the API should be implemented in a way that is compatible with the general use and idioms of the language under consideration, while retaining the general character and operation of the API.

The following example, written in the software language C, shows how the API specified in detail in Annex A can be implemented. For each of the commands listed in Annex A an appropriate C-code is defined. Together, that provides the behaviour expected and illustrated in the session state diagram in Figure 5.

D.2 API definition in C

```
#ifndef _ISO17575_
#define _ISO17575_

#ifndef BOOL
#define BOOL char
#endif

#ifndef FALSE
#define FALSE 0
#define TRUE !FALSE
#endif

#ifndef BYTE
#define BYTE char
#endif

#ifndef WORD
#define WORD unsigned short int
#endif

// These macros allow us to reuse the Enums in string form for lookup
// Assumes that macros are indexed in order from 0
//
// For any enum AAA, these macros give you;
// typedef enum { x, y, z } AAAE
// const char *AAAS [] = {"x","y","z"}
// const char *toStringAAA(AAAE x)

#define MAKEstring(a) static const char *a ## S []={a}; const char *toString ##a(a ## E x)
{ return a ##S[x]; }
#define MAKEenum(a) typedef enum {a} a ## E; const char *toString ##a(a ## E x);
#define E(x) x

// Errors returned from the API
#define ISO175752Error                      \
    E(ERNoError),                           \
    E(ERNoInstance),                        \
    E(ERInterfacePreviouslyInitialised),    \
    E(ERInsufficentPrivledge),              \
    E(ERSessionFailed),                     \
    E(ERInProgress),                        \
    E(ERUnknownStack),                      \
    E(ERNotActive),                       \
    E(ERInSession),                       \
    E(ERUnknownInstance),                 \
    E(ERUnknownEndpoint),                 \
    E(ERNoSession),                       \
    E(EREndinProgress),                   \
    E(ERHold),                            \
    E(ERDropInProgress),                  \
    E(ERMemoryError),                     \
    E(ERBadState),                        \
    E(ERNotSet),                          \
    E(ERUnknownError)

MAKEenum(ISO175752Error);

// Reasons for a callback or drop
#define ISO175752Reason   \
    E(REUnknown),       \
    E(RENormal),        \
    E(RECallback),      \
    E(RERemoteDrop),    \
    E(RETimed)

MAKEenum(ISO175752Reason);

// Responses to a session state query, or via a state change notification
#define  ISO175752State                     \
  E(STUnknownInstance),                     \
  E(STNoSession),                           \
  E(STStarting),                            \
  E(STSessionIdle),                         \
  E(STSendingElements),                     \
  E(STSendingElementsRequest),              \
  E(STSendingUnformattedMessage),           \
  E(STAwaitingElementsConfirm),             \
  E(STSessionRxElements),                   \
  E(STErrored),                             \
  E(STEnding)

MAKEenum(ISO175752State);

// Severity of a drop request
#define ISO175752Severity      \
    E(SENormal),               \
    E(SEUrgent),               \
    E(SEUnconditional)

MAKEenum(ISO175752Severity);

// This change in the definition of E is needed for the MAKEstring macro..which is used in
the C file
#undef MAKEenum
#undef E
#define E(x) #x

// Definitions to simplify creation of up API
#define ISO175752APICALLBACKInstanceStateChange(x) void (x)(ISO175752API *instance,
ISO175752StateE oldState, ISO175752StateE newState)
#define ISO175752APICALLBACKUnformattedADUReceived(x) void (x)(ISO175752API *instance, WORD
unformattedMessageLen, BYTE *unformattedMessage)
#define ISO175752APICALLBACKADURequest(x) void (x)(ISO175752API *instance, void
*elementReq)
#define ISO175752APICALLBACKADUReceived(x) void (x)(ISO175752API *instance, void *element)
#define ISO175752APICALLBACKADUSent(x) void (x)(ISO175752API *instance, BOOL sent)
#define ISO175752APICALLBACKADUSend(x) void (x)(ISO175752API *instance, BOOL canSend)
#define ISO175752APICALLBACKSessionRequest(x) void (x)(char *handle);


typedef void ISO175752API;

ISO175752ErrorE GetLastError(ISO175752API *instance);
```

```
ISO175752API *InitialiseInstance(char *stackID,

ISO175752APICALLBACKInstanceStateChange(*InstanceStateChangeSet),

ISO175752APICALLBACKUnformattedADUReceived(*UnformattedMessageReceivedSet),
             ISO175752APICALLBACKADURequest(*ElementRequestSet),
             ISO175752APICALLBACKADUReceived(*ElementReceivedSet),
             ISO175752APICALLBACKADUSent(*MessageSentSet),
             ISO175752APICALLBACKADUSend(*ElementSendSet),
             ISO175752APICALLBACKSessionRequest(*SessionRequestSet));

ISO175752ErrorE DropInstance(ISO175752API *instance,
              ISO175752SeverityE severity);

ISO175752ErrorE SetParameter(ISO175752API *instance,
                    char *param,
                 char *val);

char *GetParameter(ISO175752API *instance,
           char *param);

ISO175752ErrorE DeleteParameter(ISO175752API *instance,
                    char *param);

ISO175752ErrorE StartSession(ISO175752API *instance,
                 ISO175752ReasonE reason,
                 char *handle);

ISO175752ErrorE EndSession(ISO175752API *instance,
                    ISO175752ReasonE reason);

ISO175752ErrorE SendUnformattedADU(ISO175752API *instance,
                      WORD messageLen,
                      BYTE *message);

ISO175752ErrorE SendADUSetStart(ISO175752API *instance);

WORD SendADU(ISO175752API *instance,
       WORD messageLen,
       BYTE *message);

ISO175752ErrorE SendADUSetEnd(ISO175752API *instance);

ISO175752StateE CommsQuery(ISO175752API *instance);

BOOL StackAvail(ISO175752API *instance);

#endif
```

Annex E
(informative)

Use of this part of ISO 17575 for the EETS

E.1 General

In 2004, EU Directive 2004/52/EC of the European parliament and of the council "on the interoperability of electronic road toll systems in the community" was adopted. This EU Directive calls for the establishment of a European Electronic Toll Service (EETS).

In 2009, EC Decision 2009/750/EC "on the definition of the European Electronic Toll Service and its technical elements" was adopted. It set out the necessary technical specifications and requirements for that purpose, and contractual rules relating to EETS provision. The decision lays down rights and obligations on EETS providers, toll chargers and EETS users.

Other requirements and other EU Directives may also be applicable to the product(s) falling within the scope of this part of ISO 17575.

E.2 Overall relationship between European standardization and the EETS

EU Directive 2004/52/EC also triggered the establishment of a standardisation mandate (M/338, "Standardisation mandate to CEN, CENELEC and ETSI in support of Interoperability of electronic road toll systems in the Community") that called for development of technical standards in support of the EETS. Activities under M/338 are supervised by the "ITS co-ordination group" (ITS-CG, previously ICTSB/ITSSG).

M/338 does not explicitly call for the provision of harmonized standards (according to Directive 98/34/EC on the new approach to technical harmonization and standards), which means that this possibility is not available for the European standards that are developed in support of the EETS. Instead, this brief annex provides an outline of how this part of ISO 17575 could be used in the context of the EETS.

EC decisions can point out the use of specific standards, even if they are not formally harmonized. This is also done in EC Decision 2009/750/EC for a few standards (i.e. those that were available at the time of its approval). In case there will be more EC decisions in support of the EC directive, further European standards could be referenced there as well.

The European Commission has also published, in 2011, a "Guide for the Application of Directive on the Interoperability of Electronic Road Toll Systems" (ISBN 978-92-79-18637-0). This guide is intended to be a reference manual for all parties directly or indirectly concerned by Directive 2004/52/EC and Decision 2009/750/EC. It aims at providing help for the implementation of the EETS, including a list of standards that might be of use. The guide is only informative (e.g. the document cannot notify certain standards as "mandatory" for use in the EETS) and is intended to be updated on regular basis.

E.3 European standardization work supporting the EETS

Many of the standards developed by CEN/TC 278 have been drafted with the EETS-requirements in mind (including the use of the results from European projects such as CARDME, PISTA, CESARE and RCI). CEN representatives have also taken part as observers in working groups etc., initiated by the EC for the EETS. Hence, some work has been done in close co-operation between CEN working groups and the EC.

It should be noted that no CEN/ISO standards are "turnkey" solutions for the EETS. They are to be used as "building blocks" for the EETS, supporting the EETS legal framework and agreements between the parties concerned by the EETS. A precise EETS specification is not within the scope of CEN/ISO standards, but remains that task of the owners of the EETS scheme.

It should also be noted that CEN/ISO has a wider scope than the EETS, which is a complementary service to the national services of the Member States and optional for the users, whereas CEN/ISO standards should be applicable to all EFC services worldwide.

E.4 Correspondence between this part of ISO 17575 and the EETS

This part of ISO 17575 defines how to connect the toll specific software of on-board equipment (OBE) as well as the proxy or Back End to the lower layers of any mobile communication channel in order to abstract the toll software from the specific properties of the communication channel. With this, the toll specific-software can be developed and operated independently of the actual available communication channel.

This supports the general requirement to allow GNSS/CN-based tolling systems in all regions where at least one type of mobile communication network is available. This also supports the OBE—toll service provider communication via different mobile communication networks when roaming across regions or during several communication technology generations while this is transparent for the toll specific software.

However, the communication interface defined in this standard is not visible in the EETS role model and, hence, has no direct link to requirements in EC Decision 2009/750/EC.

Bibliography

[1] ISO/IEC 9646-7, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 7: Implementation Conformance Statements*

[2] ISO/IEC 8824-1, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*

[3] ISO 14906:2011/Amd1:2015, *Electronic fee collection — Application interface definition for dedicated short-range communication*

[4] ISO 17573:2010, *Electronic fee collection — Systems architecture for vehicle-related tolling*

[5] ISO 17575-1:2016*, Electronic fee collection — Application interface definition for autonomous systems — Part 1: Charging*

[6] ISO 17575-3:2016*, Electronic fee collection — Application interface definition for autonomous systems — Part 3: Context data*

[7] Directive 2004/52/EC of the European Parliament and of the Council of 29 April 2004 on the interoperability of electronic road toll systems in the Community. OJ L. 2004, **166** pp. 124–143

[8] 2009/750/EC, Commission Decision of 6 October 2009 on the definition of the European Electronic Toll Service and its technical elements (notified under document C(2009) 7547). OJ L. 2009, **268** pp. 11–29

[9] EC – DG for mobility and transport. Guide for the application of the directive on the interoperability of electronic road toll systems, ISBN 978-92-18637-0, 2011

ICS 35.240.60; 03.220.20

Price based on 28 pages