TECHNICAL REPORT

ISO/TR 14906

First edition
1998-12-15

Road Transport and Traffic Telematics (RTTT) — Electronic Fee Collection (EFC) — Application interface definition for dedicated short range communications

Télématique de la circulation et du transport routier — Perception du télépéage — L'interface applicative relative aux communications dédiées aux courtes portées

Reference number
ISO/TR 14906:1998(E)

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 main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/TR 14906, which is a Technical Report of type 2, was prepared by European Committee for Standardization (CEN) in collaboration with ISO Technical Committee ISO/TC 204, *Transport information and control systems,* in accordance with the Agreement on technical cooperation between ISO and CEN (Vienna Agreement).

International Organization for Standardization
Case postale 56 • CH-1211 Genève 20 • Switzerland
Internet iso@iso.ch

Printed in Switzerland

This document is being issued in the Technical Report (type 2) series of publications (according to subclause G.3.2.2 of part 1 of the ISO/IEC Directives, 1995) as a "prospective standard for provisional application" in the field of *transport information and control systems* because there is an urgent need for guidance on how standards in this field should be used to meet an identified need.

This document is not to be regarded as an "International Standard". It is proposed for provisional application so that information and experience of its use in practice may be gathered. Comments on the content of this document should be sent to the ISO/TC 204 Secretariat.

A review of this Technical Report (type 2) will be carried out not later than three years after its publication with the options of: extension for another three years; conversion into an International Standard; or withdrawal.

Annex A forms an integral part of this Technical Report. Annexes B to D are for information only.

CONTENTS

Foreword

This European Prestandard has been prepared by Technical Committee CEN/TC 278 "Road transport and traffic telematics", the secretariat of which is held by NNI, in collaboration with Technical Committee ISO/TC 204 "Transport information and control systems".

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to announce this European Prestandard: Austria, Belgium, Czech Republic, Denmark, Finland, France, Germany, Greece, Iceland, Ireland, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland and the United Kingdom.

Introduction

This European Pre-Standard specifies an application interface for Electronic Fee Collection (EFC) systems, which are based on the Dedicated Short-Range Communication (DSRC), enabling interoperability between open EFC systems (i.e. between different EFC system operators) on an EFC-DSRC application interface level.

The European Pre-Standard provides specifications for the EFC transaction model, EFC data elements (referred to as attributes) and functions, from which an EFC transaction can be built. The EFC transaction model provides a mechanism that allows handling of different versions of EFC transactions and associated contracts. A certain EFC transaction supports a certain set of EFC attributes and EFC functions as defined in this European Pre-Standard. It is not envisaged that the complete set of EFC attributes and functions is present in each piece of EFC equipment, be OBE or RSE.

This European Pre-Standard provides the basis for agreements between operators, which are needed to achieve interoperability. Based on the tools specified in this European Pre-Standard, interoperability can be reached by operators recognising each others EFC transactions (including the exchange of security algorithms and keys) and implementing the EFC transactions in each others RSE, or they may reach an agreement to define a new transaction (and contract) that is common to both. Considerations also have to be made by each operator that the RSE has sufficient resources to implement such additional EFC transactions.

In order to achieve interoperability, operators have to agree on issues like:

- which optional features are actually being implemented and used;
- security policy (including encryption algorithms and key management, if applicable);
- operational issues, such as how many receipts may be stored for privacy reasons, how many receipts are necessary for operational reasons (e.g. as entry tickets or as proof of payment);
- the agreements needed between operators in order to regulate the handling of different EFC transactions.

This European Pre-Standard has the following structure. In the first four clauses the scope, normative references, definitions of terms and abbreviations are accounted for. Next, in clause 5, the EFC Application interface architecture is described in terms of its relation to the DSRC communication architecture, including the addressing of data attributes and of components. In the following clause 6, the EFC transaction model is introduced, defining the common steps of each EFC transaction, in particular the initialisation phase. Clauses 7 and 8 are dedicated to the detailed specification of the EFC application functions and of the EFC data attributes, respectively. Four annexes provide:

- the normative ASN.1 specifications of the used data types (EFC action parameters and attributes);
- an informative excerpt from DSRC application layer (ENV 12834), and its service primitives, parameters and functions;
- informative examples of EFC transactions using the specified EFC attributes and functions;
- an informative listing of functional requirements, which can be satisfied by using the tools provided by this European Pre-standard.

1 Scope

This European Pre-Standard specifies the application interface in the context of Electronic Fee Collection (EFC) systems using the Dedicated Short-Range Communications (DSRC).

The EFC application interface is the EFC application process interface to the DSRC Application Layer, as can be seen in figure 1 below. The scope of this European Pre-Standard comprises specifications of:

- EFC attributes (i.e. EFC application information);
- the addressing procedures of EFC attributes and (hardware) components (e.g. ICC and MMI);
- EFC application functions, i.e. further qualification of actions by definitions of the concerned services, assignment of associated ActionType values and content and meaning of action parameters;
- the EFC transaction model, which defines the common elements and steps of any EFC transaction;
- the behaviour of the interface so as to ensure interoperability on an EFC-DSRC application interface level.

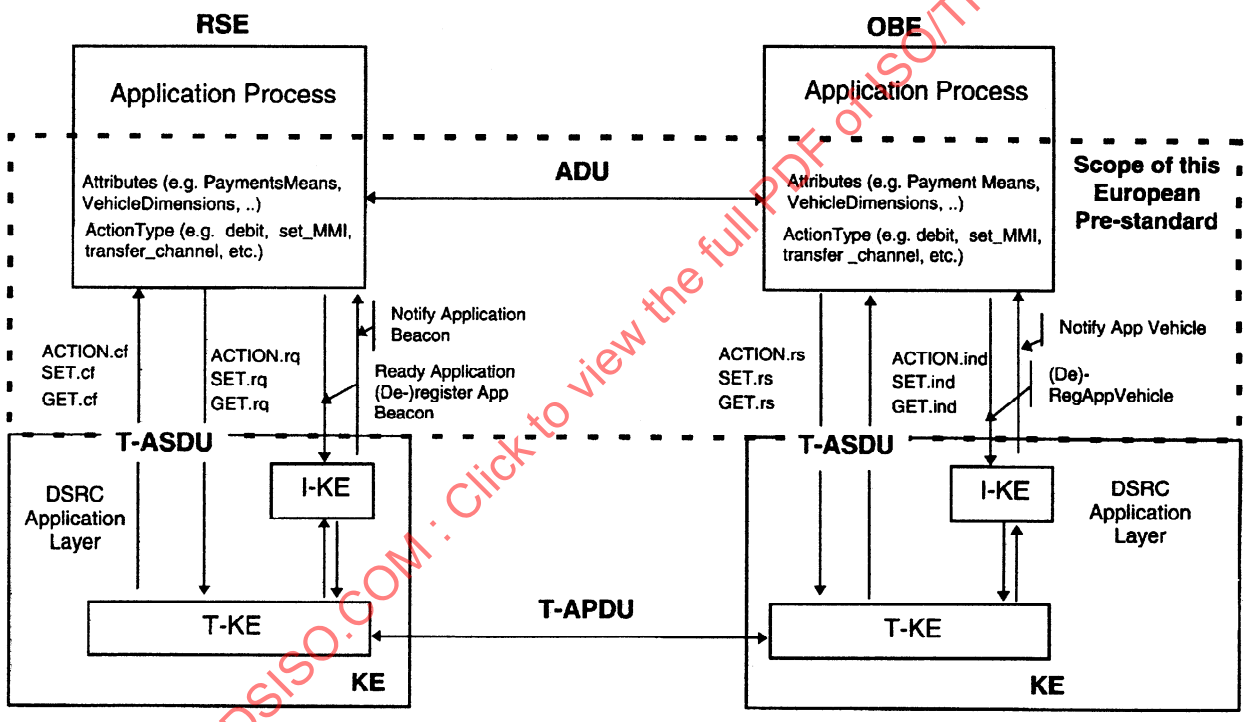


Figure 1: The EFC application interface

This is an interface standard, adhering to the open systems interconnection (OSI) philosophy (ISO/IEC 7498-1), and it is as such not concerned with the implementation choices to be realised at either side of the interface.

This European Pre-Standard provides security-specific functionality as place holders (data and functions) to enable the implementation of secure EFC transactions. Yet the specification of the security policy (including specific security algorithms and key management) remains at the discretion and under the control of the EFC operator, and hence is outside the scope of this European Pre-Standard.

2 Normative references

This European Pre-Standard incorporates by dated or undated reference, provisions from other publications.

These normative references are cited at the appropriate places in the text and the publications are listed hereafter.

For dated references, subsequent amendments to or revisions of any of these publications apply to this European Pre-Standard only when incorporated in it by amendment or revision. For undated references the latest edition of the publication referred to applies.

ISO 612:		Vehicle Measurement Definition
ISO 1176:		Vehicle Weight Definitions
ISO 3166:		Codes for the representation of names of countries
ISO 3779:	1983	Road Vehicles - Vehicle Identification Number (VIN) Content and Structure
ISO 3780:	1983	Road Vehicles World manufacturer identification code (WMI)
ISO 4217:		Codes for the representation of currencies and funds
ISO 7498-1:	1994	Information Processing Systems - Open Systems Interconnection - Basic Reference model
ISO/IEC 8824-1:	1995	Information processing systems - Open Systems Interconnection - Specification of abstract syntax notation one (ASN.1)
ISO/IEC 8825-2:	1996	Information processing systems - Open Systems Interconnection - ASN.1 encoding rules: Specification of Packed encoding rules
ENV 1545-1	1998	Identification card systems - Surface transport applications - Part 1 : General data elements
ENV 1545-2	1998	Identification card systems - Surface transport applications - Part 2 : Transport payment related data elements
prENV ISO 14816:	1998	Road Traffic and Transport Telematics (RTTT), Automatic Vehicle and Equipment Identification - Numbering and Data Structures (ISO/DTR 14816 : 1998)
ENV 12834:	1997	Road Traffic and Transport Telematics (RTTT), Dedicated Short-Range Communication (DSRC) - Application Layer

3 Definitions

For the purposes of this European Pre-Standard, the following definitions apply:

3.1 access credentials Data that is transferred to *On-Board Equipment,* in order to establish the claimed identity of an RSE application process entity.

NOTE: The access credentials carries information needed to fulfil access conditions in order to perform the operation on the addressed element in the OBE. The access credentials can carry passwords as well as cryptographic based information such as authenticators.

3.2 action Function that an application process resident at the *Roadside Equipment* can invoke in order to make the *On-Board Equipment* execute a specific operation during the *Transaction.*

3.3 attribute Application information formed by one or by a sequence of data elements, and is managed by different actions used for implementation of a *transaction.*

3.4 authenticator Data appended to, or a cryptographic transformation (see cryptography) of, a data unit that allows a recipient of the data unit to prove the source and/or the integrity of the data unit and protect against forgery.

3.5 channel An information transfer path [ISO/IEC 7498-2].

3.6 component Logical and physical entity composing an *On-Board Equipment*, supporting a specific functionality.

3.7 contract Expression of an agreement between two or more parties concerning the use of the road infrastructure.

3.8 cryptography The discipline which embodies principles, means, and methods for the transformation of data in order to hide its information content, prevent its undetected modification or/and prevent its unauthorised use [ISO/IEC 7498-2].

3.9 data group A collection of closely related EFC data attributes which together describe a distinct part of an Electronic Fee Collection transaction.

3.10 data integrity The property that data has not been altered or destroyed in an unauthorised manner [ISO 7498-2].

3.11 element In the context of DSRC, a directory containing application information in form of *Attributes.*

3.12 on-board equipment Equipment located within the vehicle and supporting the information exchange with the *Road Side Equipment.* It is composed of the *On-Board Unit* and other sub-units whose presence have to be considered optional for the execution of a *Transaction.*

3.13 on-board unit Minimum component of an *On-Board Equipment*, whose functionality always includes at least the support of the DSRC interface.

3.14 roadside equipment Equipment located at a fixed position along the road transport network, for the purpose of communication and data exchanges with the *On-Board Equipment* of passing vehicles.

3.15 service (EFC) Road transport related facility provided by a *Service Provider*. Normally a type of infrastructure, the use of which is offered to the *User* for which the *User* may be requested to pay.

3.16 service primitive (communication) Elementary communication service provided by the Application layer protocol to the application processes.

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

3.17 service provider (EFC) The operator that accepts the user's payment means and in return provides a road-use service to the user.

3.18 session The complete exchange of information and interaction occurring at a specific Electronic Fee Collection station between the *Roadside Equipment* and the user/vehicle.

3.19 transaction The whole of the exchange of information between the *Roadside Equipment* and the *On-Board Equipment* necessary for the completion of an Electronic Fee Collection operation over the DSRC.

NOTE: A transaction may require more than one session in order to be achieved, e.g. an entry session and an exit session.

3.20 transaction model Functional model describing the general structure of Electronic Fee Collection transactions.

3.21 user The entity that uses transport services provided by the *Service Provider* according to the terms of a *Contract*.

4 Abbreviations

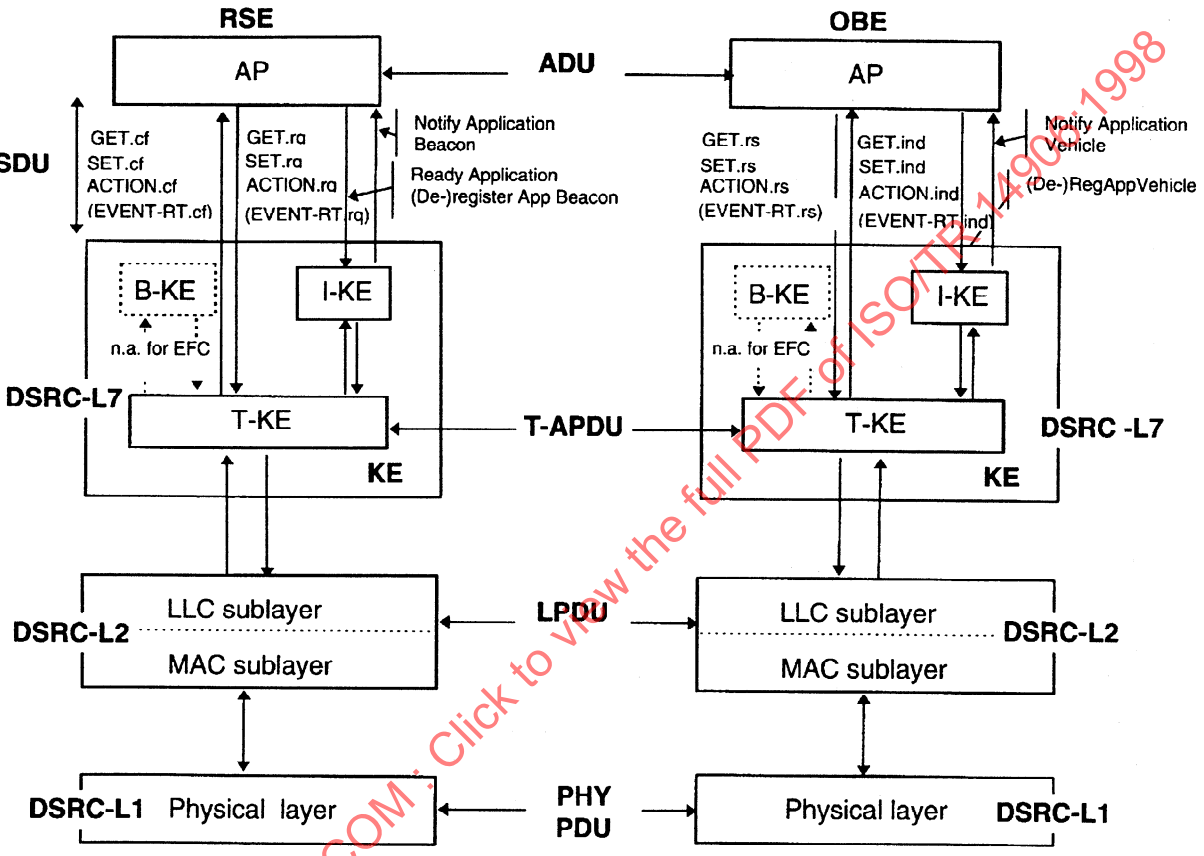
For the purpose of this European Pre-Standard, the following abbreviations apply throughout the document unless otherwise specified:

ADU	Application Data Unit
APDU	Application Protocol Data Unit
AP	Application Process
ASN.1	Abstract Syntax Notation One
BST	Beacon Service Table (DSRC Application Layer)
DSRC	Dedicated Short-Range Communications
EID	Element ID
EFC	Electronic Fee Collection
GPS	Global Positioning System
ICC	Integrated Circuit(s) Card
I-KE	Initialisation Kernel Element (DSRC Application Layer)
IID	Invoker ID
LID	Link ID
MMI	Man-Machine Interface
OBE	On-Board Equipment
OBU	On-Board Unit
PDU	Protocol Data Unit
PER	Packed Encoding Rules
RSE	Road-Side Equipment
RTTT	Road Traffic and Transport Telematics
SAM	Secure Application Module
T-APDU	Transport-Application Protocol Data Unit (DSRC Application Layer)
T-ASDU	Transport-Application Service Data Unit (DSRC Application Layer)
T-KE	Transport Kernel Element (DSRC Application Layer)
VST	Vehicle Service Table (DSRC Application Layer)

5 EFC application interface architecture

5.1 Relation to the DSRC communication architecture

The DSRC services are provided to an application process by means of the DSRC Application Layer service primitives, which are abstract implementation interactions between a communication service user and provider. The services are offered by the DSRC communication entities by means of its DSRC Application Layer (ENV 12834).



Additional abbreviations used only in this figure (for all other abbreviations see clause 4.)

B-KE	Broadcast-Kernel Element
LLC	Logical Link Control
LPDU	Link Protocol Data Unit
MAC	Medium Access Control
PHY-PDU	Physical Link Protocol Data Unit
req/ind/rs/cf	request/indication/response/confirm
EVENT-RT	EVENT-REPORT
n.a.	not applicable

Figure 2: The EFC application process on top of the DSRC communication stack

The Transfer Kernel Element (T-KE) of DSRC Application Layer offers the following services to application processes (see also figure 2 above):

- GET: The invocation of a GET service request results in retrieval (i.e. a reading) of application information (i.e. Attributes) from the peer service user (i.e. the OBE application process), a reply is always expected.
- SET: The invocation of a SET service request results in modification (i.e. writing) of application information (i.e. Attributes) of the peer service user (i.e. the OBE application process). This service may be requested in confirmed or non-confirmed mode, a reply is only expected in the former case.

- ACTION: The invocation of an ACTION service request results in a performance of an action by the peer service user (i.e. the OBE application process). An action is further qualified (see clause 7.4) by the value of the ActionType. This service may be requested in confirmed or non-confirmed mode, a reply is only expected in the former case.
- EVENT-REPORT: The invocation of an EVENT-REPORT service request forwards a notification of an event to the peer service user.
- INITIALISATION: The invocation of an initialisation service request by RSE results in an attempt to initialise communication between a RSE and each OBE that has not yet established communication with the concerned RSE. The Initialisation service is only used by the I-KE as defined in ENV 12834, which in its turn is configured by the application(s) wishing to execute applications over the DSRC link.

5.2 Usage of DSRC application layer by the EFC application interface

EFC uses the following services offered by DSRC Application Layer (as defined in ENV 12834):

- The INITIALISATION services:
  - (De-)Register Application RSE (at RSE)
  - Notify Application Beacon (at RSE)
  - Ready Application (at RSE)
  - (De-)Register Application OBE (at OBE)
  - Notify Application Vehicle (at OBE)

  are used to realise the EFC-specific initialisation mechanism (see clause 6)
- The GET service is used to retrieve EFC attributes (see annex B.2.1. For attribute specifications see clause 8)
- The SET-service is used to set EFC attributes (see annex B.2.2)
- The ACTION-services are applied to realise additional EFC specific functionality needed to support EFC application processes, such as TRANSFER_CHANNEL, SET_MMI and ECHO (see clause 7.2).

In the following, the EFC-specific usage of the DSRC Layer 7 services is specified in detail.

NOTE: The EVENT-REPORT-service can be implicitly used by EFC application processes. It is e.g. used indirectly as part of an already defined command to release an application process (see Ready Application of ENV 12834). However as the EVENT-REPORT-service is not explicitly used by EFC application processes, this service is not further referred to in this document.

5.3 Addressing of EFC attributes

5.3.1 Basic mechanism

EFC Attributes are used to transfer the EFC application-specific information.

EFC Attributes are composed of one or more data elements of specified ASN.1 types. All data elements of an ASN.1 type are mandatory. To each data element an unambiguously defined name is associated.

To each EFC Attribute, an AttributeID is associated. The AttributeId enables to unambiguously identify and address an EFC Attribute.

EXAMPLE: Figure 3 illustrates the basic addressing mechanism:

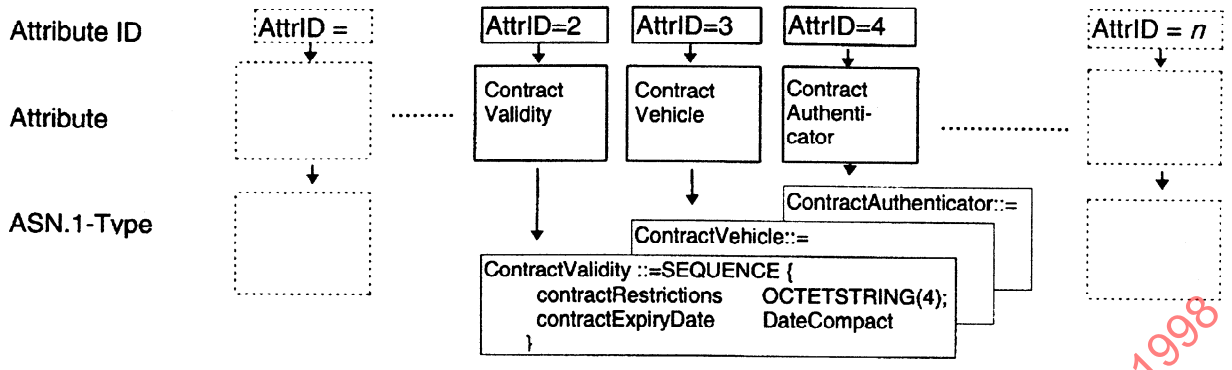


Figure 3: Basic addressing mechanism

5.3.2 Role of the EID

The DSRC-EID (different from 0) is used to identify an EFC context, given by the EFC-ContextMark (see clause 6.1.3), in which Attributes can be addressed unambiguously by AttributeIDs. In the VST, the OBE may specify several of these EFC contexts, each corresponding to a set of EFC Attributes and EFC functions supported by it.

EXAMPLE:

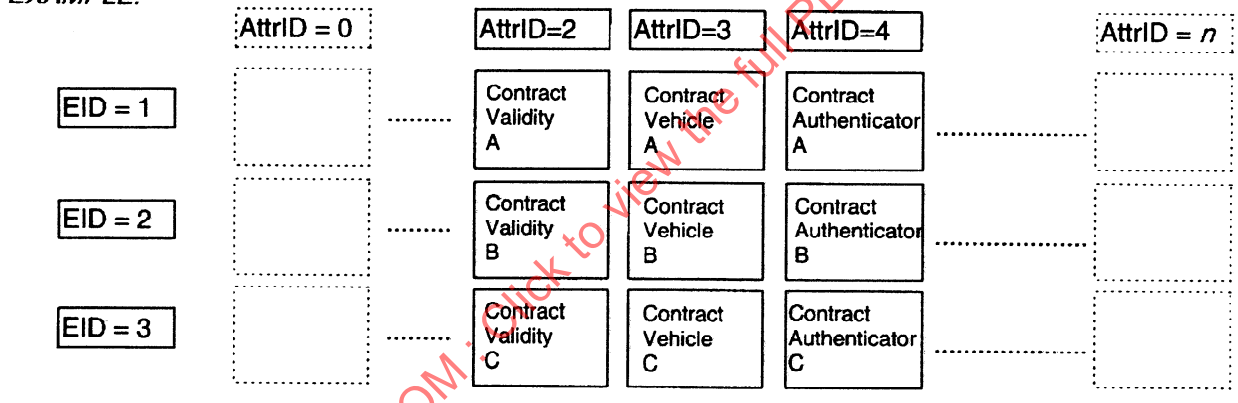


Figure 4: Role of the EID

EID equals 0 shall be used to address application-independent functions and components, e.g. SET_MMI and TRANSFER_CHANNEL (see clause 7.2).

5.3.3 Multiple Instances of Attributes

There may be n instances of an Attribute available in an OBE.
The maximum number of instances N_{max} of one Attribute may be limited according to the needs of operators and users. The default maximum number of instances is $N_{max}=1$. The value of N_{max} is determined at the time of OBE configuration.

EXAMPLE:

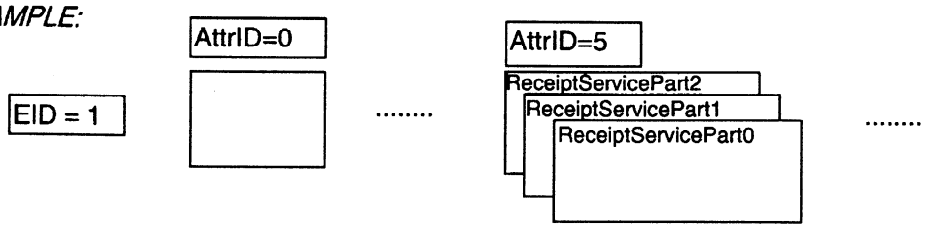


Figure 5: Multiple instances (0-2) of attribute 5

The handling of multiple instances and the corresponding addressing mechanism are described in detail as part of the behaviour specification of the corresponding functions supporting multiple instances (see clause 7.2.5 for GET_INSTANCE and clause 7.2.6 for SET_INSTANCE).

5.4 Addressing of components

Components of an OBE to be addressed via the EFC Application Interface include for example:

- OBU
- SAM 1
- SAM 2
- ICC
- Display
- Beeper
- Printer
- Serial interface
- Parallel interface
- GPS
- Tachograph

Addressing of these components is enabled on two levels, device-specific and device-independent addressing.

The ***device-specific transparent addressing mechanism*** enables the transfer of information, which shall be processed by the addressed device (such as an ICC-command). The addressed device is identified by a *channel Id*. The EFC function TRANSFER_CHANNEL (see clause 7.2.9) supports this functionality.

EXAMPLE: Transfer of a bit string to an ICC.

The ***device-independent addressing mechanism*** uses a set of commands, which describe a certain functionality, which can be performed by various OBU components. In this case, the operating system of the OBU will address the corresponding components. The EFC function SET_MMI supports this functionality (see clause 7.2.11).

EXAMPLE: Invocation of a SET_MMI(EID=0, ContactOperator) function activates an OBE MMI-device, e.g. a beeper or a display.

> *NOTE: In addition, the OBE may use a kind of implicit addressing. In an implementation, specific attributes or data elements may activate some MMI function (e.g. a SET command on the attribute ReceiptText might display the text on an LCD display. A SET command on the attribute ReceiptServicePart with data element SessionResultOperational other than SessionOK might activate an alert beep).*

6 EFC Transaction Model

The EFC Transaction Model related to the EFC Application Interface for the DSRC comprises two phases, the initialisation phase and the transaction phase.

NOTE: The purpose of the initialisation phase is to set up the communication between the RSE and OBEs that have entered the DSRC zone but have not yet established communication with the RSE, and to notify the application processes. It provides amongst others a multi-application switching mechanism, allowing for execution of several RTTT applications (in parallel) at one RSE station.

The transaction phase can only be reached after completion of the initialisation phase. The EFC functions, as defined in clause 7, can be performed in the transaction phase.

6.1 Initialisation Phase

6.1.1 Overview

This clause provides an overview of the functionality of, and the information exchanges in, the initialisation phase.

The Initialisation procedures, by means of BST and VST exchanges, are defined in ENV 12834. Clauses 6.1.2 and 6.1.3 below account for the EFC application-specific information that shall be included in the BST and VST, respectively.

NOTE 1: The OBE evaluates the received BST, and selects the applications that it wishes to perform out of the lists of applications supported by the RSE. If the OBE does not support any of application(s) supported by the RSE, then the OBE shall not exchange any information with the RSE. If the OBE supports at least one of the application(s) supported by the RSE, then the OBE shall send its corresponding VST, informing the RSE of which application it wishes to execute.

NOTE 2: Figure 6 describes the initialisation phase.

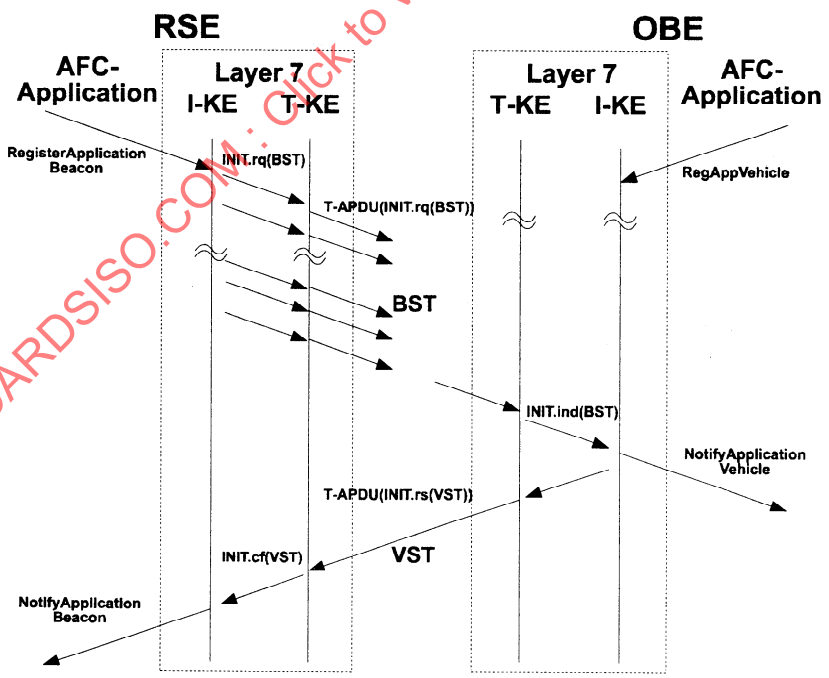


Figure 6: Initialisation phase: BST - VST exchanges

The Initialisation service associated with the initialisation phase is only used by the I-KE (of ENV 12834), which in its turn is configured by the application(s) wishing to execute applications over the DSRC link. The I-KEs of the RSE and of the concerned OBE shall have been configured, according to ENV 12834, prior to the invocation of the Initialisation service by the RSE.

6.1.2 EFC application-specific contents of the BST

An RSE supporting EFC shall have configured its I-KE to carry the following information related specifically to the EFC application(s):

- the application identifier (AID) shall be equal to 1 (i.e. the value assigned for EFC);
- the EFC application shall be qualified as a mandatory application;
- EID shall not be transmitted in the BST related to the EFC application;
- No Parameter shall be transmitted in the BST related to the EFC application.

There shall be only one EFC application present in the BST (i.e. there shall be only one instance of aid=1 in the BST) regardless of whether the RSE supports more than one EFC-ContextMark (see also clause 6.1.3) or not.

NOTE 1: The above is the EFC application-specific contents of the BST. The complete BST is defined in ENV 12834 and is given below for readability:

```
BST ::= SEQUENCE {
beacon BeaconID,
time                  Time,
profile               Profile,
mandApplications      ApplicationList,
nonmandApplications   ApplicationList  OPTIONAL,
profileList           SEQUENCE(0...127, ...) OF Profile
}

where:
ApplicationList ::= SEQUENCE (0..127,...) OF
SEQUENCE{
aid          DSRCApplicationEntityID,     --aid=1
eid          Dsrc-EID        OPTIONAL,    --empty
parameter    Container       OPTIONAL     --empty
}
```

NOTE 2: There is likely to be assigned other aid values for identification of non-European EFC contexts, where those contexts will define the usage of eid and parameter fields (e.g. non empty).

6.1.3 EFC application-specific contents of the VST

Each EFC application and corresponding contract shall be associated with an EFC-ContextMark, as defined below. An OBE may support several EFC applications. If several EFC applications are supported by an OBE, then the order in which the EFC-ContextMark are presented in the VST shall correspond to the user's order of preference. The RSE should honour the first EFC-ContextMark that it supports as presented in the VST.

An OBE supporting EFC shall have configured its I-KE to carry the following information related specifically to the concerned EFC application:

- the AID shall be equal to 1;
- the EID value shall be unique within the OBE throughout the complete DSRC session, and shall be logically associated with the corresponding EFC-ContextMark contained in the Parameter;
- the Parameter shall be of Container CHOICE type OCTET STRING and shall comprise the EFC-ContextMark as defined below, and may also be configured to carry additional EFC attributes (as defined in clause 8 and annex A).

EFC-ContextMark::=SEQUENCE{

```
ContractProvider        Provider,
TypeOfContract                  OCTET STRING (SIZE(2)),
ContextVersion                  INTEGER(0..127,..)
}
```

The **EFC-ContextMark** denotes a specific EFC context in the OBE, comprising the organisation that issued the contract, the type of contract and the context version. ContractProvider, TypeOfContract and ContextVersion are further defined in clause 8 as data elements of the Attribute EFC-ContextMark.

NOTE 1: The above is the EFC application-specific contents of the VST. The complete VST is defined in ENV 12834 and is given below for readability:

```
VST ::=   SEQUENCE {
fill                    Bit STRING (SIZE(4))
profile                 Profile,
applications            ApplicationList,
obeConfiguration                ObeConfiguration
}

where:
ApplicationList ::= SEQUENCE (0..127,...) OF
SEQUENCE{
aid             DSRCApplicationEntityID,        --aid =1
eid             Dsrc-EID        OPTIONAL,       --eid = e.g. 2
parameter       Container       OPTIONAL        --EFC-ContextMark
                                                --plus any EFC Attribute
}
```

NOTE 2: There is likely to be assigned other aid values for identification of non-European EFC contexts, where those contexts will define the usage of eid and parameter fields.

NOTE 3: An EFC application provider retains the ultimate control of his security domain, i.e. the security level and the associated security mechanisms to be used within his system.

6.2 Transaction phase

After completion of the Initialisation phase, the appropriate RSE application is informed (by means of the Notify Application Beacon service) of the EFC-ContextMark(s) and associated EID(s). The RSE shall use the functions defined in clause 7 to complete the EFC transaction.

The RSE may invoke any sequence of EFC functions to complete the EFC transaction, provided that they are supported by the EFC-ContextMark. The OBE shall respond to the EFC functions invoked by the RSE, and shall not initiate any EFC functions (by usage of a request service primitive, see further clause 7) on its side.

EXAMPLE: A transaction may consist of the following steps:

- *GET(EID, ContractValidity, ContractVehicle, ReceiptServicePart, PaymentMeansBalance)*
- *DEBIT(EID, Fee)*
- *SET(EID, ReceiptServicePart)*

Due to the construction of the EFC part of the VST, each EID identifies a certain EFC-ContextMark and shall be used by the RSE as parameter of every function to unambiguously address data elements within the context given by the EFC-ContextMark. More than one EID may be used in one session.

Both which attributes are implemented and which are not, and the maximum number of instances of an attribute is defined at time of configuration of the OBE and is outside the scope of this European Pre-Standard. These implementation dependent aspects are referenced unambiguously by the ContextVersion data element in the EFC-ContextMark.

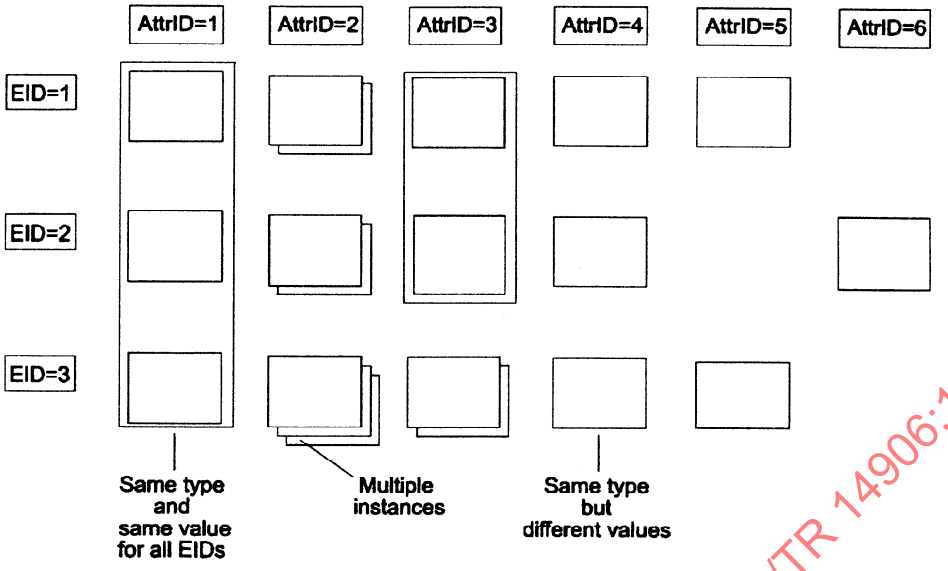


Figure 7: Context of attributes given by EFC-ContextMark and identified by the EID

NOTE 1: This construction of contexts being identified by EIDs allows amongst others to implement the following transactions:

- *booking from two contracts in one transaction. There is sometimes the need to book from two contracts in one session, e.g. when a customer has a contract with a reduced price (e.g. a commuter contract) for part of the route being tolled, plus a standard (not reduced) contract for the rest of the route. This may be implemented by having all data groups identical between two EIDs, except for the data group contract.*
- *having either two instances of the data group Vehicle to accommodate a pulling vehicle plus a trailer, or having two EIDs with separate data groups Vehicle for pulling vehicle and trailer (and probably also separate Contract)*

NOTE 2: This EFC transaction model and associated procedures allow for different levels of co-existence and interoperability between operators:

- *No agreement between operators - each operator has a completely separate application domain, i.e. there are no common data groups. Each operator books by using "his" EID.*
- *Agreement to share some data groups, but not others. E.g. the data groups Vehicle, Receipt and Payment are shared, but not Contract. Different security measures (algorithms) are used by the two operators. Or, all data groups are shared, except for Payment - each operator books from an account issued by himself.*

7 EFC Functions

7.1 Overview and general concepts

7.1.1 EFC functions and service primitives

This clause describes the EFC functions invoked by T-ASDUs exchanged between peer applications communicating via a DSRC link. The T-ASDUs are exchanged by means of service primitives of the DSRC Application Layer (ENV 12834). Exchanges of service primitives (and the corresponding T-ASDUs) associated with EFC functions adhere to the following basic pattern:

- xxx.rq (request) service primitive invoked by the RSE application to DSRC Application Layer;
- xxx.ind (indication) service primitive issued by the DSRC Application Layer to the OBE application;
- xxx.rs (response) service primitive invoked by the OBE application to DSRC Application Layer;
- xxx.cf (confirmation) service primitive issued by the DSRC Application Layer to RSE application.

The last two steps are either mandatory or optional, depending on the nature of the service primitive and on the setting of the Mode parameter (see clause 6.2.2.4 in ENV 12834, or annex B.1 in this European Pre-Standard).

The logical sequence of a successful service primitives exchange (for Mode = TRUE) is illustrated in figure 8 below. Service primitives that occur earlier in time, and are connected by dotted lines in the figure, are the logical antecedents of subsequent service primitives.

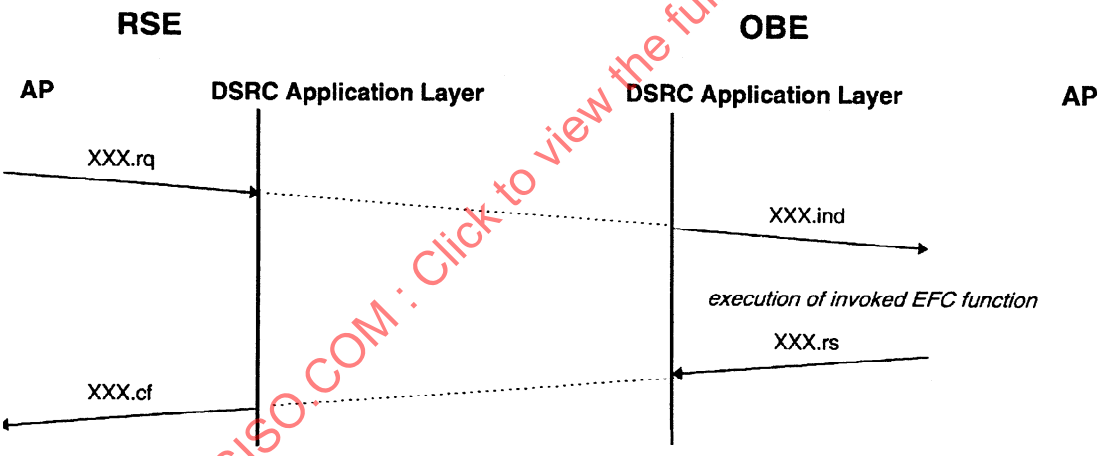


Figure 8: The logical sequencing of service primitive exchanges

For the purposes of this European Pre-Standard, the DSRC link is seen as completely transparent, i.e. in the absence of exceptions the XXX.ind is identical in content and meaning to the XXX.rq, and the XXX.cf is identical in content and meaning to the XXX.rs. For the purpose of conciseness there will be:

- one description for *request*, i.e. XXX.rq, covering both request and indication service primitives;
- one description for *response*, i.e. XXX.rs, covering both response and confirmation service primitives.

The format and the parameters of the service primitives of the DSRC application layer are defined in ENV 12834, clause 6.2.2 (T-KE Services).

> *NOTE: Annex B.1 accounts for the format and the definitions of the parameters of the service primitives in form of an excerpt from ENV 12834 in order to facilitate the reading of this European Pre-Standard.*

7.1.2 Overview of EFC functions

This clause provides an overview of the EFC functions (based on the Action service primitive of ENV 12834) that are defined in clause 7.2 of this European Pre-Standard. Each EFC function comprises a pair of service primitives, a request and its associated response service primitive, which are accounted for in clauses 7.2 but not in table 1 below.

Table 1: Overview of EFC functions

Function Name	Action Type	Action Parameter	Response Parameter	Remarks
GET_STAMPED	0	GetStampedRq	GetStampedRs	retrieves data with an authenticator from the OBE
SET_STAMPED	1	SetStampedRq	OCTET STRING	sets data in the OBE, which generates an authenticator
GET_SECURE	2	OCTET STRING	OCTET STRING	gets data securely from the OBE
SET_SECURE	3	OCTET STRING	OCTET STRING	sets data securely in the OBE
GET_INSTANCE	4	GetInstanceRq	GetInstanceRs	retrieves a number of entries out of an attribute's multiple instances
SET_INSTANCE	5	SetInstanceRq	n.a.	sets one entry at a specified position in an attribute's multiple instances
GET_NONCE	6	n.a.	OCTET STRING	retrieves a nonce - typically used against replay attacks
SET_NONCE	7	OCTET STRING	n.a.	sets a nonce - typically used against replay attacks
TRANSFER_CHANNEL	8	ChannelRq	ChannelRs	sets and/or retrieves data from the addressed OBE component (e.g. ICC)
COPY	9	CopyRq	n.a.	Copies data from a source EID to a destination EID
SET_MMI	10	SetMMIRq	n.a.	invokes an MMI function (e.g. signal Ok via buzzer)
SUBTRACT	11	SubRq	n.a.	subtracts the given value to the addressed value
ADD	12	AddRq	n.a	adds the given value to the addressed value
DEBIT	13	DebitRq	DebitRs	debits purse
CREDIT	14	CreditRq	CreditRs	credits purse
ECHO	15	OCTET STRING	OCTET STRING	OBE echoes received data
	16-31	Container	Container	*future CEN EFC use*

The GET and SET services (DSRC application layer functions) as defined in ENV 12834 (in clause 6.2) may also be used in an EFC transaction phase.

NOTE 1: GET is used to retrieve (i.e. read) value(s) of the addressed attribute(s), a reply is always expected. SET is used to set (i.e. write) value(s) of the addressed attribute(s). Annex B.1 this European Pre-Standard accounts for the format and the parameters of the GET and SET service primitives, whilst annex B.2 accounts for the behaviour of the corresponding functions in order to facilitate the reading of this European Pre-Standard.

NOTE 2: ReadyApplication initiates the EVENT-REPORT(Release) function.

7.1.3 Handling of multiple instances

For the purpose of description, the number of instances is denoted by *n*. In general the EFC functions operating on multiple instances of OBE Attributes can be divided into the following groups:

- GET, GET_STAMPED

  These functions shall always access the last instance (i.e. instance at position 0). If no instance is available, the result is undefined but may lead to the return of an error code.

- GET_INSTANCE

  This function carries parameters *N* 1 and N2, both ≥ 0. It shall return the following:
  - if N2<N1, or N1>n, then the empty list;
  - if *N2 >= N1* then the values of the instances numbered N1, N1+1, ... up to and including min(N2, n)

  NOTE: The case that zero instances are returned is legal, in this case the response carries an empty list.

- SET, SET_STAMPED

  These functions shall always set the value of instance at position 0. In addition, the previous instance number p (where p is an integer between 0 and N_{max}-1) shall become instance number p+1, and instance number N_{max} shall no longer be available.

  NOTE 1: A cyclic buffer is as acceptable as a dynamic memory allocation scheme.

  NOTE 2: The description above also covers the common case for N_{max} =1. In this case it leads to overwriting the old value of the single instance.

- SET_INSTANCE

This function carries a parameter N ≥0, and a value for the addressed attribute. It shall always set the value of instance number N.

EXAMPLE 1: Behaviour for a static memory allocation scheme - a cyclic buffer
Assume N_{max} = 3. Table 2 shows the effects of a certain sequence of functions.

Table 2: Behaviour for a static memory allocation scheme

Function	*n*	*buffer content instance position*			*result*
		0	*1*	*2*	
GET	*3*	*X*	*X*	*X*	*returns X*
GET_INSTANCE(1,0)	*3*	*X*	*X*	*X*	*returns empty list*
SET (A)	*3*	*A*	*X*	*X*	
GET	*3*	*A*	*X*	*X*	*returns value A*
SET(B)	*3*	*B*	*A*	*X*	
GET	*3*	*B*	*A*	*X*	*returns value B*
GET_INSTANCE(0,7)	*3*	*B*	*A*	*X*	*returns list (B,A,X)*
SET(C)	*3*	*C*	*B*	*A*	
GET	*3*	*C*	*B*	*A*	*returns value C*
GET_INSTANCE(1,2)	*3*	*C*	*B*	*A*	*returns list (B, A)*
SET(D)	*3*	*D*	*C*	*B*	*value A is no longer available*
SET_INSTANCE(1,E)	*3*	*D*	*E*	*B*	*value C is no longer available*

EXAMPLE 2: Behaviour for a dynamic memory allocation scheme
Assume N_{max} = 3. Let n = 0 initially. Table 3 shows the effects of a certain sequence of functions.

Table 3: Behaviour for a dynamic memory allocation scheme

Function	*n*	*buffer content instance position*			*result*
		0	*1*	*2*	
GET	*0*				*undefined (implementation dependent)*
GET_INSTANCE(1,0)	*0*				*returns empty list*
SET (A)	*1*	*A*			
GET	*1*	*A*			*returns value A*
SET(B)	*2*	*B*	*A*		
GET	*2*	*B*	*A*		*returns value B*
GET_INSTANCE(0,6)	*2*	*B*	*A*		*returns list(B, A)*
SET(C)	*3*	*C*	*B*	*A*	
GET	*3*	*C*	*B*	*A*	*returns value C*
GET_INSTANCE(1,2)	*3*	*C*	*B*	*A*	*returns list (B, A)*
SET(D)	*3*	*D*	*C*	*B*	*value A is no longer available*
SET_INSTANCE(1,E)	*3*	*D*	*E*	*B*	*value C is no longer available*

7.1.4 Security

Whilst security is an essential part of EFC applications, the actual mechanisms are outside the scope of this European Pre-Standard. It is generally recognised that security mechanisms involve many parameters, like encryption algorithm and keys (if the security mechanism is encryption based at all), hash function, key length, padding method, redundancy data etc. It is assumed that the EFC application communicating parties know everything they need, either by implementation or by deriving information from the VST. This information should suffice for the RSE to determine how to proceed. The OBE, in general, supports only a limited number of security mechanisms.

In this European Pre-Standard, only a framework is defined permitting security mechanisms to be specified unambiguously at the discretion of the application provider. It includes the Access Credentials defined in ENV12834. In addition security values, like authenticators, will often be needed for additional protection.

7.1.4.1 Use of access credentials and authenticators

Access Credentials and Authenticators are defined as being of ASN.1 type OCTET STRING. This only pertains to the ASN.1 syntax, the semantics are implicit in the context given by the EFC-Context Mark as specified in the VST, and as selected by the EID.

Access Credentials shall be used to manage access to attributes. Different access conditions can apply for different attributes, and if so different access credentials should be associated with these access conditions.

EXAMPLE: *The VehicleDimensions EFC attribute may be associated with no access conditions whilst the ContractSerialNumber and ContractValidity EFC attributes may be subject to access conditions (e.g. requiring the correct password to be presented).*

Authenticators shall primarily pertain to values, and prove the source and/or the integrity of the data unit and protect against forgery. Authenticators are used in cryptography related EFC functions such as GET_STAMPED and SET_STAMPED etc. Authenticators can be transmitted from the RSE to the OBE, as Access Credentials in order to prove the authenticity of the RSE, or from the OBE to the RSE to prove the source and/or integrity of the data unit.

The security mechanisms to be applied, and the exact role of Access Credentials and Authenticators, will be determined by the owner of the EFC-ContextMark and is outside the scope of this European Pre-standard.

7.2 EFC functions

In this clause, the EFC functions are specified in detail. The format and the parameters of the EFC functions shall adhere to the Action service primitives of the DSRC application layer as defined clause 6.2.2 (T-KE Services) in ENV 12834. Not all parameters associated with the EFC functions are accounted for in this clause, as they are either not specifically needed for the EFC applications or have the same meaning for all functions.

NOTE: Annex B.1 accounts for the format and the definitions of the all parameters of the service primitives in form of an excerpt from ENV 12834 in order to facilitate the reading of this European Pre-Standard.

The Return Codes (RET) are explicitly specified whenever additional precision is needed on top of the specifications given in clause 6.2.2.4 in ENV 12834 (see also annex B in this European Pre-Standard).

The ASN.1 type specifications of the ActionParameters and ResponseParameters are provided in the normative annex A.

7.2.1 GET_STAMPED

GET_STAMPED is used to retrieve the value(s) of the addressed attribute(s), with an authenticator appended to the retrieved data. The authenticator generation involves transformations (notably encryption) that may include a nonce value (e.g. a random number or a sequence number).

Table 4: GET_STAMPED.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	0	
AccessCredentials	OCTET STRING		optional use
ActionParameter	GetStampedRq ::= SEQUENCE { attributeIdList AttrIdList, nonce OCTET STRING, keyRef INTEGER(0..255) }		Always to be present
Mode	BOOLEAN	TRUE	

GET_STAMPED.request shall request the retrieval of the value(s) of the attributes addressed by the attributeIdList, with an authenticator given in the response. A response shall always be expected (Mode = TRUE). The parameter keyRef shall contain a reference to the key to be used for the calculation of the authenticator in the response.

NOTE: The AccessCredentials are only needed if the data attributes addressed by EID and attributeIdList require authentication of the RSE.

Table 5: GET_STAMPED.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	GetStampedRs::= SEQUENCE { attributeList AttrList, authenticator OCTET STRING }		Always to be present
Return Code (Ret)	ReturnStatus		optional use

GET_STAMPED.response shall carry the retrieved value(s) of the addressed attribute(s) in the attributeList, as the result of the corresponding GET_STAMPED.request command. An authenticator over the retrieved values shall be carried in the authenticator parameter, with the keyRef parameter of the GET_STAMPED.request being used as a reference to the (cryptographic) key to be used. When a nonce of non-zero length is given in the request, the nonce value shall be included in the cryptographic transformation.

NOTE: GET_STAMPED can be used with an empty attributeIdList to request an authenticator from the OBE to authenticate the OBE EFC application.

7.2.2 SET_STAMPED

SET_STAMPED is used to set the value(s) of the addressed attribute(s), with the OBE returning an authenticator as a proof that the data has been set. The authenticator generation involves transformations (notably encryption) which may include a nonce value (e.g. a random number or a sequence number).

Table 6: SET_STAMPED.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	1	
AccessCredentials	OCTET STRING		optional use
ActionParameter	SetStampedRq::= SEQUENCE { attributeList AttrList, nonce OCTET STRING, keyRef INTEGER(0..255) }		Always to be present
Mode	BOOLEAN	TRUE	

SET_STAMPED.request shall request the setting of the value(s) of the attributes addressed by the attributeList, with an authenticator given in the response. A response shall always be expected (Mode = TRUE).

Table 7: SET_STAMPED.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	OCTET STRING		Always to be present
Return Code (Ret)	ReturnStatus		optional use

SET_STAMPED.response shall carry an authenticator as the response parameter (being of ASN.1 type OCTET STRING) to the corresponding request to convey that the data in the attribute list of the request have been set. The authenticator shall be calculated over the values given in the request attributeList, with the keyRef parameter of the request being used as a reference to the (cryptographic) key to be used. When a nonce of non-zero length is given in the request, the nonce value shall be included in the cryptographic transformation.

7.2.3 GET_SECURE

GET_SECURE is used to retrieve the value(s) of attribute(s) subject to security measures defined implicitly by the context identification set in the initialisation phase. These measures may involve any kind of transformations (notably encryption).

Table 8: GET_SECURE.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	2	
AccessCredentials	OCTET STRING		optional use
ActionParameter	OCTET STRING		Always to be present
Mode	BOOLEAN	TRUE	

GET_SECURE.request shall request the retrieval of attributes subject to security measures implicit in the context set in the VST, amongst others by explicit reference given in the action parameter. The ActionParameter (being of ASN.1 type OCTET STRING) shall carry the AttributeIds of the requested attributes plus any information (nonce, key reference) required by the algorithm providing the security measures. A reply is always expected (Mode = TRUE).

NOTE 1: The accessCredentials are only needed if the data attributes addressed by EID and AttributeIdList require them.

NOTE 2: The interpretation of the actionParameter is defined by the security mechanism in effect, which is implicit in the context identification in the initialisation phase. The parameter includes a (possibly encrypted) AttrIDList, and it may also contain, e.g., an authenticator for non-repudiation purposes.

Table 9: GET_SECURE.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	OCTET STRING		Always to be present
Return Code (Ret)	ReturnStatus		optional use

GET_SECURE.response shall carry as the responseParameter (being of ASN.1 type OCTET STRING) to the corresponding request the requested value(s) of the addressed attribute(s) in the form (e.g. encrypted) implicitly defined in the context set in the VST, amongst others by explicit reference given in the action parameter.

NOTE: The interpretation of the responseParameter is defined by the security mechanism that is in effect. The parameter includes a (possibly encrypted, or otherwise transformed) AttrList. It may in addition contain, e.g., an authenticator for non-repudiation purposes.

7.2.4 SET_SECURE

SET_SECURE is used to set the value(s) of attribute(s) subject to security measures defined implicitly by the context identification set in the initialisation phase. These measures may involve any kind of transformations and additions (e.g. checking of authenticators).

Table 10: SET_SECURE.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	3	
AccessCredentials	OCTET STRING		optional use
ActionParameter	OCTET STRING		Always to be present
Mode	BOOLEAN		

SET_SECURE.request shall request the setting of attributes subject to security measures implicit in the context set in the VST, amongst others by explicit reference given in the action parameter. The ActionParameter (being of ASN.1 type OCTET STRING) shall carry the attributes to be set plus any information (authenticator, nonce, key reference) required by the algorithm providing the security measures. *SET_SECURE.request* can be used in confirmed or non-confirmed mode; a reply shall always be expected in the former case.

NOTE: The interpretation of the ActionParameter is defined by the security mechanism in effect, which is implicit in the context identification in the initialisation phase. The parameter includes a (possibly encrypted) attrList, and it may also contain, e.g., an authenticator for non-repudiation purposes.

Table 11: GET_SECURE.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	OCTET STRING		optional use
Return Code (Ret)	ReturnStatus		optional use

SET_SECURE.response shall carry as the ResponseParameter (being of ASN.1 type OCTET STRING) explicitly carry the confirmation of the corresponding request, if in confirmed mode. The confirmation shall be in the form implicitly defined in the context set in the VST, amongst others by explicit reference given in the action parameter.

NOTE: The interpretation of the ResponseParameter is entirely defined by the security mechanism that is in effect, which is implicit in the context identification in the initialisation phase. It may, e.g., be empty or contain an authenticator to be used for non-repudiation of receipt.

7.2.5 GET_INSTANCE

GET _INSTANCE is used to retrieve a number of values from multiple instances of the addressed attributes (See clause 7.1.3 for the handling of multiple instances).

Table 12: GET_INSTANCE.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	4	
AccessCredentials	OCTET STRING		optional use
ActionParameter	GetInstanceRq ::= SEQUENCE { posOfFirstInstance INTEGER(0..255), posOfLastInstance INTEGER(0..255), attributeIdList AttrIdList}		
Mode	BOOLEAN	TRUE	

GET_INSTANCE.request shall request the retrieval of a number of instances of the value(s) of the addressed attribute(s). The ActionParameter contains the position of the first instance and the last instance of the instances of the specified attribute(s) to be retrieved. GET_INSTANCE.*request* can only be used in confirmed mode, a reply shall always be expected.

Table 13: GET_INSTANCE.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	GetInstanceRs ::= SEQUENCE (0..127,...) OF { attributeId INTEGER(0..127,...), attributeValues Container::=OCTET STRING }		
Return Code (Ret)	ReturnStatus		optional use

GET_INSTANCE.response shall, as response to the corresponding request, contain all available values of the requested attributes, starting from the value at the first position (posOfFirstInstance) up to the value at the last position (posOfLastInstance), as asked for in the request. The ResponseParameter shall for each requested attribute in turn contain first the attribute Id of the requested attribute, followed by the values of the attribute. The value(s) of an attribute at the first position shall be transferred first in the parameter attributeValues.

7.2.6 SET_INSTANCE

SET _INSTANCE is used to set the value of a specified entry from multiple instances of the addressed attribute (See clause 7.1.3 for the handling of multiple instances).

Table 14: SET_SECURE.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	5	
AccessCredentials	OCTET STRING		optional use
ActionParameter	SetInstanceRq ::= SEQUENCE { posOfInstance INTEGER(0..255) attribute Attr		
Mode	BOOLEAN		

SET_INSTANCE.request shall request the replacement of a selected instance of the addressed attribute. The ActionParameter contains the value (posOfInstance) attempted to be replaced and the attribute Id. SET_INSTANCE.*request* can be used in confirmed or non-confirmed mode (i.e. Mode equal to TRUE or FALSE, respectively), a reply shall always be expected in the former case.

Table 15: SET_INSTANCE.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	none		
Return Code (Ret)	ReturnStatus		optional use

SET_INSTANCE.response shall explicitly convey the result of the corresponding *SET_INSTANCE.request*. If the addressed value could not be replaced, the Return Code shall indicate a failure.

7.2.7 GET_NONCE

GET_NONCE is used by the RSE to obtain a nonce value (e.g. a random number, a sequencing number or a time stamp) to be used for guaranteeing a unique relationship between a number of related data items. The retrieved value shall remain "active" throughout the session or until a new GET_NONCE service has been successfully completed within the same session.

EXAMPLE:GET_NONCE can be used to get a challenge value, which is used as an input parameter when computing the applicable access credentials (e.g. an authenticator). The resulting data can subsequently be included as access credentials in another request command.

NOTE: Guaranteeing uniqueness implies certain requirements on the value to be produced, e.g. sequencing numbers and random numbers must have a sufficiently large period, time stamps must have sufficiently high resolution. In addition, random numbers may have to be generated by a cryptographic algorithm to be "unpredictable" enough. These additional requirements are outside the scope of this European Pre-Standard.

Table 16: GET_NONCE.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	6	
AccessCredentials	OCTET STRING		n.a.
ActionParameter	n.a.		n.a.
Mode	BOOLEAN	TRUE	

GET_NONCE.request shall request the retrieval of a value to be used for guaranteeing a unique relationship between a number of related data items. *GET_NONCE.request* shall always be used in confirmed mode, a reply shall always be expected.

Table 17: GET_NONCE.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	OCTET STRING		
Return Code (Ret)	ReturnStatus		optional use

GET_NONCE.response shall, as response to the corresponding request, carry as the ResponseParameter (being of ASN.1 type OCTET STRING) a value to be used for guaranteeing a unique relationship between a number of related data items. The retrieved value shall remain "active" throughout the session or until a new GET_NONCE service has been successfully completed within the same session.
When the *GET_NONCE.request* is not supported by the OBE EFC application, then the Return Code shall indicate complexityLimitation, and ResponseParameter shall be empty.

7.2.8 SET_NONCE

SET_NONCE is used by the RSE to present a value (e.g. a sequencing number, a random number or a time stamp) to the OBE, to be used for guaranteeing a unique relationship between a number of related data items. The set value remains "active" throughout the session or until a new SET_NONCE service has been successfully completed within the same session.

Table 18: SET_NONCE.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	7	
AccessCredentials	OCTET STRING		n.a.
ActionParameter	OCTET STRING		Always to be present
Mode	BOOLEAN		

SET_NONCE.request shall request setting a nonce value to be used for guaranteeing a unique relationship between a number of related data items. The ActionParameter (being of ASN.1 type OCTET STRING) shall carry the nonce value. *SET_NONCE.request* can be used in confirmed or non-confirmed mode (i.e. Mode equal to TRUE or FALSE, respectively), a reply shall always be expected in the former case.

Table 19: SET_NONCE.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	n.a.		n.a.
Return Code (Ret)	ReturnStatus		optional use

SET_NONCE.response shall be issued as a response to the corresponding request to convey the result of the request. A receiving peer entity supporting no nonce shall return a Return Code indicating Complexity Limitation.

7.2.9 TRANSFER_CHANNEL

TRANSFER_CHANNEL is used to send and/or retrieve data from a dedicated channel of the OBU.

Table 20: TRANSFER_CHANNEL.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID	0	
ActionType	INTEGER(0..127,...)	8	
AccessCredentials	OCTET STRING		not to be used
ActionParameter	ChannelRq::= SEQUENCE { channelId ChannelId, apdu OCTET STRING }		
Mode	BOOLEAN		

TRANSFER_CHANNEL.request shall request data transfer from and/or to a dedicated channel of the OBU. The channel shall be addressed by the parameter channelId. Data to be transferred to the OBU channel shall be contained in the parameter apdu in a format recognised by the addressed component. In case no data is to be set to the OBU channel, the apdu can be empty. The direction(s) of transfer is either implicitly given by the context of the addressed channel within the context given in the VST, or is to be conveyed as part of the parameter apdu in a format appropriate for the addressed channel.

NOTE: TRANSFER_CHANNEL allows addressing of data residing in components of the OBE using a transparent application layer protocol bridge. The command can e.g. be used to address a serial interface, or an electronic purse that requires a protocol that is not covered by the DEBIT command.

Table 21: TRANSFER_CHANNEL.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	ChannelRs ::= SEQUENCE { channelId ChannelId, apdu OCTET STRING }		
Return Code (Ret)	ReturnStatus		optional use

TRANSFER_CHANNEL.response shall carry the response to the requested data transfer from and/or to a dedicated channel of the OBU. The parameter channelId shall contain the channel Id of the request. Data requested to be transferred from the OBU channel shall be contained in the parameter apdu in a format specific to the addressed component. In case no data is to be returned from the OBU channel, the apdu can be empty.

7.2.10 COPY

Copy is used to copy the values of the addressed attribute(s) to another EID (i.e. the destination Id).

EXAMPLE 1: When not implemented in the OBU itself, "sharing" of data attributes for common use between two operators (between two different context addressed by two different EIDs) can be performed with a copy command. Note that operators can protect the data under "their" EID by requiring certain Access Credentials only known to parties they have an agreement with.

EXAMPLE 2: The RSE sets (i.e. writes) the last event into the Receipt data group in the OBU, which only stores a limited number of events. The RSE invokes a copy command requesting the OBU to copy the value of the last event to the event log of the ICC.

Table 22: COPY.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	9	
AccessCredentials	OCTET STRING		optional use
ActionParameter	CopyRq::= SEQUENCE { destinationEID INTEGER(0..127,...) attributeIdList AttrIdList }		Always to be present
Mode	BOOLEAN		

COPY.request shall request the copying of the value(s) of the addressed attribute(s) to the same attribute(s) in a destination EID. *COPY.request* can be used in confirmed or unconfirmed mode, a reply shall always be expected in the former case.

Table 23: COPY.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	n.a.		n.a.
Return Code (Ret)	ReturnStatus		optional use

COPY.response shall be used to explicitly convey the result of the corresponding COPY.request command.

7.2.11 SET_MMI

SET_MMI is used to perform device-independent MMI functions.

Table 24: SET_MMI.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID	0	
ActionType	INTEGER(0..127,...)	10	
AccessCredentials	OCTET STRING		not to be used
ActionParameter	SetMMIRq::= INTEGER { ok (0), nok (1), contactOperator (2), reservedForFutureCENUse (3-127), reservedForPrivateUse (128-255) } (0..255)		Always to be present
Mode	BOOLEAN		

SET_MMI.request shall request to control the MMI in a device-independent way. The ActionParameter shall contain the MMI function that is to be invoked, e.g. signalling of a successful operation (such as a successful EFC transaction), a non-successful operation or signalling to contact the operator (e.g. due to low balance). *SET_MMI.request* can be used in confirmed or non-confirmed mode, a reply shall always be expected in the former case.

Table 25: SET_MMI.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	n.a.		n.a.
Return Code (Ret)	ReturnStatus		optional use

SET_MMI.response shall be used to explicitly convey the result of the corresponding SET_MMI.request command. If Mode was set to TRUE in the corresponding request and the operation was successfully executed then the response shall indicate no error. If an error occurred at an attempt to execute the command then the Return Code shall take the appropriate value.

7.2.12 SUBTRACT

SUBTRACT is used to subtract a given INTEGER value from another value of an INTEGER-type attribute.

Table 26: SUBTRACT.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	11	
Access Credentials	OCTET STRING	{}	OPTIONAL

ActionParameter	SubRq::= SEQUENCE {
	attributeId INTEGER(0..127,..),
	value INTEGER
	}
			MANDATORY

Mode	BOOLEAN		

SUBTRACT.request is used to request the subtraction of the value, as contained in the action parameter, from the value of the addressed attribute. *SUBTRACT.request* can be used in confirmed or non-confirmed mode, a reply shall always be expected in the former case.

Table 27: SUBTRACT.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	n.a.		
Return Code (Ret)	ReturnStatus		OPTIONAL

SUBTRACT.response is a response used to explicitly convey the result of the corresponding *SUBTRACT.request* command. *SUBTRACT.response* shall be invoked upon completion of an attempt to execute the corresponding *SUBTRACT.request* command.

7.2.13 ADD

ADD is used to add a given INTEGER value to another value of an INTEGER-type attribute.

Table 28: ADD.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		
ActionType	INTEGER(0..127,...)	12	
Access Credentials	OCTET STRING	{}	OPTIONAL

ActionParameter	AddRq::= SEQUENCE {
	attributeId INTEGER(0..127,..),
	value INTEGER
	}
			MANDATORY

Mode	BOOLEAN		

ADD.request shall request the addition of the value, as contained in the action parameter, to the value of the addressed attribute. *ADD.request* can be used in confirmed or non-confirmed mode, a reply shall always be expected in the former case.
The AccessCredentials protect the value against unauthorised modification.

Table 29: ADD.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	n.a.		
Return Code (Ret)	ReturnStatus		OPTIONAL

ADD.response is a response used to explicitly convey the result of the corresponding *ADD.request* command. *ADD.response* shall be invoked upon completion of an attempt to execute the corresponding *ADD.request* command.

7.2.14 DEBIT

DEBIT is used to perform a debit on the attribute PaymentMeansBalance. The command contains payment related data (a price) and optionally security related data. A (cryptographic) proof of payment can be returned.

Table 30: DEBIT.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		unequal 0
ActionType	INTEGER(0..127,...)	13	
AccessCredentials	OCTET STRING		optional use
ActionParameter	DebitRq::= SEQUENCE { debitFee Fee nonce OCTET STRING keyRef INTEGER(0..255) }		Always to be present
Mode	BOOLEAN	TRUE	

DEBIT.request shall request a debiting transaction to be performed on the attribute PaymentMeansBalance. *DEBIT.request* shall be used in confirmed mode, a reply shall always be expected. The parameter debitFee shall contain the price, including a currency and a multiplicator, to be subtracted from the attribute PaymentMeansBalance, where PaymentMeansUnit, the unit of PaymentMeansBalance, shall be taken into account. Nonce shall contain a nonce value to be included in the (cryptographic) calculation of the response authenticator or be empty. The parameter keyRef shall contain a reference to the key to be used for the calculation of the authenticator in the response, if required.

Table 31: DEBIT.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	DebitRs ::= SEQUENCE { debitResult Result, debitAuthenticator OCTET STRING }		Always to be present
Return Code (Ret)	ReturnStatus		optional use

DEBIT.response shall contain the response to the corresponding request. On reception of a DEBIT command, in case the currencies of the debitFee parameter of the request and of the attribute PaymentMeansUnit in the OBE match, the OBE shall attempt to subtract the requested debitFee from the attribute PaymentMeansBalance, taking the multiplicators of the debitFee parameter and of the attribute PaymentMeansUnit into account. In case of currency mismatch, the OBE shall not subtract the amount given in debitFee, and shall return debitResult = 'Transaction not successful, currency not accepted'. Depending on the context identified by the VST or implicitly given by the type of payment means, the response shall either include a proof of payment in debitAuthenticator, or return an empty debitAuthenticator.
When a nonce of non-zero length is given in the request, the nonce value shall be included in the cryptographic transformation performed to generate debitAuthenticator. If a nonce of zero length is given and if a nonce is required by the cryptographic algorithm, then the nonce given in the most recent SET_NONCE command of the session shall be used.
In case the attempt to debit failed, the parameter debitResult shall be set to the appropriate value.

7.2.15 CREDIT

CREDIT is used to perform a credit (refund) on the attribute PaymentMeansBalance. The command contains payment related data (a refund) and optionally security related data. A (cryptographic) proof can be returned.

Table 32: CREDIT.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID		unequal 0
ActionType	INTEGER(0..127,...)	14	
AccessCredentials	OCTET STRING		optional use
ActionParameter	CreditRq::= SEQUENCE { refund Fee nonce OCTET STRING keyRef INTEGER(0..255) }		Always to be present
Mode	BOOLEAN	TRUE	

CREDIT.request shall request a refunding transaction to be performed on the attribute PaymentMeansBalance. CREDIT.request shall be used in confirmed mode, a reply shall always be expected. The parameter refund shall contain the price, including a currency and a multiplicator, to be added to the attribute PaymentMeansBalance, where PaymentMeansUnit, the unit of PaymentMeansBalance, shall be taken into account. Nonce shall contain a nonce value to be included in the (cryptographic) calculation of the response authenticator or be empty. The parameter keyRef shall contain a reference to the key to be used for the calculation of the authenticator in the response, if required.

Table 33: CREDIT.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	CreditRs ::= SEQUENCE { creditAuthenticator OCTET STRING creditResult Result }		Always to be present
Return Code (Ret)	ReturnStatus		optional use

CREDIT.response shall contain the response to the corresponding request. On reception of a CREDIT command, in case the currencies of the refund parameter of the request and of the attribute PaymentMeansUnit in the OBE match, the OBE shall attempt to add the requested refund to the attribute PaymentMeansBalance, taking the multiplicators of the debitFee parameter and of the attribute PaymentMeansUnit into account. In case of currency mismatch, the OBE shall not add the amount given in debitFee, and shall return creditResult = 'Transaction not successful, currency not accepted'. Depending on the context identified by the VST or implicitly given by the type of payment means, the response shall either include a proof of refund in creditAuthenticator, or return an empty creditAuthenticator.
When a nonce of non-zero length is given in the request, the nonce value shall be included in the cryptographic transformation performed to generate creditAuthenticator. If a nonce of zero length is given and if a nonce is required by the cryptographic algorithm, then the nonce given in the most recent SET_NONCE command of the session shall be used.
In case the attempt to credit failed, the parameter creditResult shall be set to the appropriate value.

7.2.16 ECHO

ECHO is used for dummy data (i.e. bitstream) being sent to, and returned by, the peer service user. This service may be used for testing purposes and for localisation of the OBU during the passage of the DSRC-EFC station.

Table 34: ECHO.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier EID	Dsrc-EID	0	
ActionType	INTEGER(0..127,...)	15	
AccessCredentials	OCTET STRING		not to be used
ActionParameter	OCTET STRING		
Mode	BOOLEAN		

ECHO.request shall request dummy data (i.e. an octet string of variable length) to be sent to and returned by the addressed OBU. *ECHO.request* can be used in confirmed or non-confirmed mode, a reply shall always be expected in the former case.

Table 35: ECHO.response

parameter name	ASN.1 type	Value	Remark / Constraints
ResponseParameter	OCTET STRING		Same as in the request
Return Code (Ret)	ReturnStatus		optional use

ECHO.response shall be used to return the data conveyed in ActionParameter, or the result, of the corresponding *ECHO.request* command.

8 EFC Attributes

Within the context of EFC, the following EFC Attributes or a subset thereof shall be available to perform an EFC transaction:

Table 36: EFC Attributes

AttributeID	Attribute	Length in Octet	Data Group
0	EFC-ContextMark	6	Contract
1	ContractSerialNumber	4	
2	ContractValidity	6	
3	ContractVehicle	variable	
4	ContractAuthenticator	variable	
5	ReceiptServicePart	13	Receipt
6	SessionClass	2	
7	ReceiptServiceSerialNumber	3	
8	ReceiptFinancialPart	variable	
9	ReceiptContract	9	
10	ReceiptOBUId	variable	
11	ReceiptICC-Id	variable	
12	ReceiptText	variable	
13	ReceiptAuthenticator	variable	
14	ReceiptDistance	3	
15	VehicleIdentificationNumber	variable	Vehicle
16	VehicleLicencePlateNumber	variable	
17	VehicleClass	1	
18	VehicleDimensions	3	
19	VehicleAxles	2	
20	VehicleWeightLimits	6	
21	VehicleWeightLaden	2	
22	VehicleSpecificCharacteristics	4	
23	VehicleAuthenticator	variable	
24	EquipmentOBUId	variable	Equipment
25	EquipmentICC-Id	variable	
26	EquipmentStatus	2	
27	DriverCharacteristics	2	Driver
28	PaymentMeans	12	Payment
29	PaymentMeansBalance	3	
30	PaymentMeansUnit	2	
31	PaymentSecurityData	variable	
32-96	*ReservedForFutureCENuse*		
96-127	*ReservedForPrivateUse*		

In the following tables, EFC Attributes are grouped into context-specific tables (data groups) and specified in terms of:

- the name of a data attribute;
- the names of the data elements forming the EFC Attribute - there are no optional data elements within any one EFC attribute;
- the definition of the data element;
- the ASN.1 type (including possible references to other standards);
- the length in octets (PER coded);
- the value range;
- informative remarks.

EFC Attributes that describe similar aspects of EFC information are grouped into data groups. The grouping has been made to facilitate readability and imply neither any relations with regard to addressing nor to logical interdependence. The corresponding ASN.1 module and type specification are provided in the normative annex A.

Not every EFC attribute has to be present in an any one implementation of this European Pre-Standard in order to be compliant.

NOTE: Which EFC attributes are present and which are not is implementation dependent. The implementation is identified by the context given in the EFC-ContextMark of the VST.

8.1 Data group CONTRACT

Information associated with the service rights of the user of an EFC service.

Table 37: Data group Contract

EFC Attribute	*Data element*	*Definition*	*Type*	*Length in octet*	*Value range*	*Informative remarks*
EFC-ContextMark	ContractProvider	Identifies the organisation that issued the service rights given in the Contract. Numbers shall be assigned on a national basis.	Provider	3	AA..ZZ & 0.. 16383	ASN.1 Type and Value assignment defined in annex A The ContractProvider might, e.g., be a single operator or a group of operators holding an inter-company agreement. Note that the Attribute EFC-ContextMark is part of the VST.
	TypeOfContract	ContractProvider-specific designation of the rules that apply to the Contract.	OCTET STRING(2)	2		Allows, e.g., for the determination of the tariff or designating the type of purse associated with the contract.
	ContextVersion	ContextVersion denotes the implementation version of the concerned contract within the context of the given ContractProvider, value assigned at the discretion of the ContractProvider.	INTEGER(0..127,..)	1		The ContextVersion may also be used as a security key reference.
ContractSerial Number	ContractSerial Number	Serial number designating the individual contract, value assigned at the discretion of the ContractProvider.	INT4	4	0 ... 4294967263	
ContractValidity	Contract Restrictions	ContractProvider specific coding of the validity restrictions of a contract.	OCTET STRING(4)	4		Allows for finer validity restrictions in addition to TypeOfContract, like applicable vehicle classes, zones of the network, duration of validity. (TypeOfContract is given in the VST and is to be kept short.)
	ContractExpiry Date	End date of the validity of the contract. Contract validity ends at 24h of ContractExpiryDate.	DateCompact	2	[01.01.1997].. [31.12.2060]	Start date not given - it is usually implicitly given by the type of contract. When necessary it may be calculated, since duration of validity may be coded in ContractValidity or follows implicitly from TypeOfContract. Imported from ENV 1545-1.
ContractVehicle	ContractVehicle	For vehicle bound contracts, ContractVehicle gives the licence plate number to which the contract is restricted.	CS4	variable		Contracts valid for two or three vehicles may be handled with multiple instances of ContractVehicle. Imported definition from prENV ISO 14816.
Contract Authenticator	Contract Authenticator	Authenticator calculated by the ContractProvider when issuing the Contract, to prevent tampering with contract data.	OCTET STRING	variable		It is not specified over which attributes of the data group the authenticator is to be calculated.

8.2 Data group RECEIPT

Information associated to a specific session, including both financial and operational data.

Table 38: Data group Receipt (continued)

EFC Attribute	*Data element*	*Definition*	*Type*	*Length in octet*	*Value range*	*Informative remarks*
Receipt ServicePart	SessionTime	Time of session with a two-seconds resolution.	Time	4	[01.01.1997, 00:00:00] ... [31.12.2060, 21:59:58], then rollover.	Easy to decode into a displayable format by OBE.
	SessionService Provider	Organisation that provides the service of the session.	Provider	3	AA..ZZ & 0..16383	Type defined in annex A
	Station Location	Service provider specific coding of the station location.	INTEGER [0..1048575]	2.5		e.g.: toll plaza No.
	Session Location	Service provider specific coding of the session location within the station location.	BIT STRING(8)	1		e.g.: equipment No. (lane No., beacon No.) at the toll plaza.
	TypeOfSession	Designates the type of service station.	StationType (=ENUMERATED)	0.5	**0** unspecified **1** closed entry with payment **2** closed entry without payment **3** closed transit **4** closed exit **5** closed credit **6** mixed **7** passage (open exit) **8** checkpoint **9** reload **10...15** reserved	
	SessionResult Operational	Code designating whether a session has been completed successfully or not with regard to operational issues.	Result	1	Imported from ENV 1545-2	
	SessionResult Financial	Code designating whether a session has been completed successfully or not with regard to financial reasons.	Result	1	Imported from ENV 1545-2	
Session Class	Session TariffClass	Service provider specific tariff class applied in the session.	INT1	1		Enables to reproduce the price calculation (e.g. claimed or measured vehicle class that was applied.)
	Session ClaimedClass	Service provider specific vehicle class derived from claimed characteristics in the data group Vehicle.	INT1	1		Claimed class and applied class (tariff class) may differ.

Table 39: Data group Receipt (concluded)

EFC Attribute	*Data element*	*Definition*	*Type*	*Length in octet*	*Value range*	*Informative remarks*
ReceiptService SerialNumber	ReceiptService SerialNumber	ServiceProvider specific serial number of the session given by the RSE.	INT3	3	0 .. 16777215	
Receipt FinancialPart	SessionPayment MeansProvider	Organisation supplying the payment means used in the session.	Company	3-17		Value assignment as defined in ISO/IEC 7812-1. Imported from ENV 1545-1.
	SessionPayment MeansID	Payment means provider specific identification of the payment means or account.	INT4	4	0... 4294967295	e.g. serial number
	SessionFee	The amount paid for the service.	Fee	4		Both Fee and Balance contain a currency designation plus a multiplicator.
	SessionCurrent Balance	Balance of the payment means after the session.	PurseBalance	5		In case SessionCurrentBalance is not applicable, the value 0 shall be used. Imported from ENV 1545-2.
	ReceiptFinancial SerialNumber	Serial number of the financial receipt	INT4	4	0... 4294967295	
ReceiptContract	SessionContract Provider	Organisation that issued the contract applied in the session.	Provider	3	AA..ZZ & 0..16383	Type defined in annex A
	SessionTypeOf Contract	TypeOfContract applied in the session.	OCTET STRING(2)	2		
	SessionContract SerialNumber	ContractSerialNumber of the contract applied in the session.	INT4	4	0... 4294967295	
ReceiptOBUId	ReceiptOBUId	Serial number of the on-board unit used in the session, unique within the context of the manufacturer.	OCTET STRING	variable		The manufacturer ID is always exchanged as a part of the VST
ReceiptICC-Id	ReceiptICC-Id	Identification number of smart card used in the session.	OCTET STRING	variable		Multiple instances for multiple ICCs.
ReceiptText	ReceiptText	Plain text decodeable by the OBU. See also NOTE 1 below.	OCTET STRING	variable		May be used to display session information to user (e.g. session location).
Receipt Authenticator	Receipt Authenticator	Authenticator over some Attributes of the data group Receipt, calculated by the SessionServiceProvider.	OCTET STRING	variable		
Receipt Distance	Receipt Distance	Total distance covered by the vehicle, since the beginning of its existence. The default distance unit shall be kilometres.	INT3	3	0..16777215	Vehicle distance readings, e.g. via an interface to a tachograph, may be used to the determine the fee based on the travelled distance.

NOTE 1: The OCTET STRING ASN.1 type associated with RecepitText may be replaced by an UTF8String (defining rules for value assignments for multiple character sets), expected to be included in a revised version of ISO/IEC 8824-1, in a revised version of this European Pre-Standard.

NOTE 2: The distance unit, of the ReceiptDistance, may not be the default unit in a non-European context. The non-default distance unit will then be either regionally defined or be defined at the discretion of the EFC-ContextMark issuer.

8.3 Data group VEHICLE

Information pertaining to the vehicle. When more sets of vehicle characteristics are needed within the context of an EID, e.g. to cater for a pulling vehicle and a trailer, then multiple instances of the pertinent EFC attributes shall be used.

Table 40: Data group Vehicle

EFC Attribute	*Data element*	*Definition*	*Type*	*Length in octet*	*Informative remarks*
VehicleIdentification Number	VehicleIdentification Number	Identification number of vehicle according ISO 3779	CS5	variable	Imported from prENV ISO 14816
VehicleLicence PlateNumber	VehicleLicence PlateNumber	Claimed licence plate of the vehicle	CS4	variable	Imported from prENV ISO 14816
VehicleClass	VehicleClass	Service provider specific information pertaining to the vehicle.	INT1	1	
VehicleDimensions	VehicleLengthOverall	Nominal maximum overall length of the vehicle according to ISO 612, in dm, rounded to the next dm.	INT1	1	
	VehicleHeightOverall	Nominal overall unladen height, according to ISO 612, in dm, rounded to the next dm.	INT1	1	
	VehicleWidthOverall	Nominal overall width, according to ISO 612, in dm, rounded to the next dm	INT1	1	
VehicleAxles	VehicleFirstAxleHeight	Bonnet height, measured over the front axle, in dm, rounded to the next dm.	INT1	1	
	VehicleAxlesNumber	Number of axles, including drop axles	INT1	1	
VehicleWeight Limits	VehicleMaxLaden Weight	Maximum permissible total weight including payload, according to ISO 1176. 100kg units, rounded to the next 100kg step.	INT2	2	
	VehicleTrainMaximum Weight	Maximum permissible weight of the complete vehicle train, as defined in ISO 1176. 100kg units, rounded to the next 100kg step.	INT2	2	ISO 1176 Code ISO-M18 maximum design mass of vehicle combination
	VehicleWeightUnladen	Nominal unladen weight, according to ISO 1176 in 100kg units, rounded to the next 100kg step.	INT2	2	
VehicleWeightLaden	VehicleWeightLaden	Actual weight of vehicle including load in 100kg units, rounded to the next 100kg step.	INT2	2	
VehicleSpecific Characteristics	VehicleSpecific Characteristics	Further vehicle characteristics. Each enumerated value has a specific meaning assigned. The meaning of some values are defined in this standard, others are reserved for future needs.	VehicleSpecific CharacteristicsType	4	Assignment of meaning to the unassigned enumerated values is subject to registration according to the registration procedure specified in ENV 12834.
VehicleAuthenticator	VehicleAuthenticator	Authenticator calculated by the entity entering the data elements at time of entry or modification.	OCTET STRING	variable	

8.4 Data group EQUIPMENT

Information pertaining to the onboard equipment.

Table 41: Data group Equipment

EFC Attribute	*Data element*	*Definition*	*Type*	*Length in octet*	*Value range*	*Informative remarks*
EquipmentOBUId	EquipmentOBUId	Identification number of onboard unit	OCTET STRING	variable		The manufacturer ID is always exchanged as a part of the VST
EquipmentICC-Id	EquipmentICC-Id	Identification number of smart card	OCTET STRING	variable		Multiple instances shall be used for multiple smart cards
EquipmentStatus	EquipmentStatus	Operator-specific EFC application-related information pertaining to the status of the equipment	BIT STRING(16)	2		Boolean information to support an operator's handling of an OBU on application level. (E.g. „next suitably equipped gantry should take an enforcement picture")

8.5 Data group DRIVER

Driver/User-related information (groups/individuals) used to calculate the tariff to be applied. The tariff may depend on the characteristics of the driver, as for example its being part of a specific group of drivers (category) or its specific driving behaviour.

Table 42: Data group Driver

EFC Attribute	*Data Element*	*Definition*	*Type*	*Length in octet*	*Value Range*	*Informative remarks*
Driver Characteristics	DriverClass	Description of the driver´s characteristics as pertinent to the calculation of the tariff; contract provider specific coding.	INT1	1	0...255	Information that may affect the tariff to be applied.
	TripPurpose	Parameter indicating the purpose of the trip of the user as pertinent to the calculation of the tariff; contract provider specific coding.	INT1	1	0...255	

8.6 Data group PAYMENT

Data associated with the payment transaction.

Table 43: Data group Payment

EFC Attribute	*Data element*	*Definition*	*Type*	*Length in octet*	*Value range*	*Informative remarks*
PaymentMeans	PaymentMeans Provider	Purse provider or account provider identification	Company	3-17		
	PaymentMeansID	Purse serial number or Account Serial Number.	INT4	4		
	PaymentMeans ExpiryDate	Expiring date of payment means. Payment means expires at 24h of PaymentMeans ExpiryDate.	DateCompact	2	[01.01.1997].. [31.12.2060]	Imported from ENV 1545-1
PaymentMeans Balance	PaymentMeans Balance	Balance of payment means in units of PaymentMeansUnit	Value	3	-8388608... +8388607	Imported from ENV 1545-2.
PaymentMeans Unit	PaymentMeans Unit	The unit of the payment means value in multiples or fractions of a currency or in units of tokens.	PayUnit	2		Imported from ENV 1545-2.
PaymentSecurity Data	PaymentSecurity Data	Security-related data for the authentication of the data integrity	OCTET STRING	variable		

NOTE: The following requirements have been taken into account for the definition of the payment attribute. various payment types have to be supported:

- *on-Board account;*
- *central account;*
- *pre and post-payment;*
- *purse/token-based payment;*
- *open payment system/'closed' payment system;*
- *No/Zero Payment;*
- *Refunding;*

The EFC transaction can be divided into an EFC service transaction and an EFC payment transaction (for example the visit in a restaurant also involves a service transaction (ordering and serving of meals) and a payment transaction (cash, credit card)). The contract provides the link between service transaction and payment transaction.
The EFC service provider may be independent of the payment system operator/issuer. There may be also different security domains.
In order to accommodate open payment systems, the payment attribute may be transmitted transparently (as OCTET STRING) from the OBE to the RSE
The Balance and the currency may be accessed independently, in order not to disclose the balance when only currency is needed to debit the right price.

ANNEX A (normative) EFC data type specifications

The EFC data types and associated coding related to the EFC action parameters, response parameters and attributes, described in clauses 7 and 8, are defined using the Abstract Syntax Notation One (ASN.1) technique according to ISO/IEC 8824-1. The packed encoding rules according to ISO/IEC 8825-2 shall be applied.

```
EfcModule OBJECT IDENTIFIER ::= {iso standard iso 14906}

EfcModule DEFINITIONS ::= BEGIN

IMPORTS AlphabetIndicator, CountryCode, CS4, CS5,
IssuerIdentifier FROM WG12-CodingStructures,        -- defined in prENV ISO 14816
CapacityUnit, Company, CountryAlpha, CountryNumeric, DateCompact, Datef, Holder, HolderBirth, HolderName,
HolderProfile, Iai, NetworkId, Status, TimeCompact, TimeReal,
VersionNumber FROM TransportGeneral,          -- defined in ENV 1545-1
Account, Amount, ClassQualifier, Contract, ContractCustomerInfo, ContractJourneyData, ContractGuaranteeInfo,
ContractPassengerInfo, ContractRestriction, ContractSaleData, ContractTariff, ContractValidityInfo, CustomerProfile,
DateContract, LocationReference, PassengerClass, PayEventCode, PayLocationType, PaymentCertificate,
PaymentEvent, PaymentFee, PaymentReceipt, PayMethod, PayUnit, Price, PurseBalance, PurseContext,
PurseValue, Receipt, RestrictDOW, Result, RoadTollContract, RoadTollEvent , Services,
ServiceReceipt, Value FROM TransportPayment             -- defined in ENV 1545-2

Container1.0  ::= [17]   GetStampedRq
Container1.1  ::= [18]   GetStampedRs
Container1.2  ::= [19]   SetStampedRq
Container1.3  ::= [20]   GetInstanceRq
Container1.4  ::= [21]   GetInstanceRs
Container1.5  ::= [22]   SetInstanceRq
Container1.6  ::= [23]   ChannelRq
Container1.7  ::= [24]   ChannelRs
Container1.8  ::= [25]   CopyRq
Container1.9  ::= [26]   SubRq
Container1.10 ::= [27]   AddRq
Container1.11 ::= [28]   DebitRq
Container1.12 ::= [29]   DebitRs
Container1.13 ::= [30]   CreditRq
Container1.14 ::= [31]   CreditRs
Container1.15 ::= [32]   EFC-ContextMark
Container1.16 ::= [33]   ContractSerialNumber
Container1.17 ::= [34]   ContractValidity
Container1.18 ::= [35]   ContractVehicle
Container1.19 ::= [36]   ContractAuthenticator
Container1.20 ::= [37]   ReceiptServicePart
Container1.21 ::= [38]   SessionClass
Container1.22 ::= [39]   ReceiptServiceSerialNumber
Container1.23 ::= [40]   ReceiptFinancialPart
Container1.24 ::= [41]   ReceiptContract
Container1.25 ::= [42]   ReceiptOBUId
Container1.26 ::= [43]   ReceiptICC-Id
Container1.27 ::= [44]   ReceiptText
Container1.28 ::= [45]   ReceiptAuthenticator
Container1.29 ::= [46]   ReceiptDistance
Container1.30 ::= [47]   CS4                    -- vehicle licence  plate number
Container1.31 ::= [48]   CS5                    -- vehicle identification number
Container1.32 ::= [49]   VehicleClass
Container1.33 ::= [50]   VehicleDimensions
Container1.34 ::= [51]   VehicleAxles
Container1.35 ::= [52]   VehicleWeightLimits
Container1.36 ::= [53]   VehicleWeightLaden
Container1.37 ::= [54]   VehicleSpecificCharacteristics
Container1.38 ::= [55]   VehicleAuthenticator
Container1.39 ::= [56]   EquipmentOBUId
Container1.40 ::= [57]   EquipmentICC-Id
Container1.41 ::= [58]   EquipmentStatus
Container1.42 ::= [59]   DriverCharacteristics
Container1.43 ::= [60]   PaymentMeans
Container1.44 ::= [61]   PaymentMeansBalance
Container1.45 ::= [62]   PaymentMeansUnit
Container1.46 ::= [63]   PaymentSecurityData
```

```
Container1.47 ::= [64]      TransportApplicationData          -- defined in ENV 1545-1/2
Container1.x  ::= [65-96]   ReservedForFutureCENuse           -- x is a number between 48 to 79
Container1.y  ::= [97-127] ReservedForPrivateUse              -- y is a number between 80 to 110

    -- NOTE1: Container1.x (where x is an integer value from 0 to 110) denoting EFC
    -- application specific ASN.1 types. These values are assigned to the Container ASN.1
    -- type of ENV 12834. Container type values between 0 to 16 denote application
    -- independent ASN.1 types (used e.g. by the EFC functions TRANSFER_CHANNEL's action
    -- and response parameters). Container type values between 17 to 31 denote application
    -- specific ASN.1 types assigned to EFC action and response parameters). Container
    -- type values between 32 to 64 denote application specific ASN.1 types assigned to EFC
    -- attributes. The value range 65 to 96 is reserved for future CEN use, whilst 97 to 127 is
    -- allocated for proprietary use. See also annex B.3 for the definition of the Container
    --  ASN.1 type as defined in ENV 12834.

-- NOTE2: Below the definitions of the action and response parameters

AddRq::= SEQUENCE {
        attributeId             INTEGER(0..127,..),
        value                   INTEGER
}

ChannelId::= INTEGER {
        obu                     (0),
        sam 1                   (1),        -- secure application module
        sam 2                   (2),
        icc                     (3),        -- integrated circuit(s) card
        display                 (4),
        beeper                  (5),
        printer                 (6),
        serialInterface         (7),        -- serial interface: eg. RS232 and RS485
        parallelInterface       (8),
        gPS                     (9),        -- GPS
        tachograph              (10),
        privateUse              (11-15),    -- free for proprietary use
        reservedForFutureCENUse (16-255)
} (0..255)

ChannelRq::= SEQUENCE{
        channelId               ChannelId,
        apdu                    OCTET STRING -- format according to the interface of the channelId
}

ChannelRs::= SEQUENCE{
        channelId               ChannelId,
        apdu                    OCTET STRING -- format according to the interface of the channelId
}

CopyRq::= SEQUENCE {
        destinationEID          INTEGER(0..127,...),
        attributeIdList         AttrIdList
}

CreditRq::= SEQUENCE {
        refund                  Fee,
        nonce                   OCTET STRING,
        keyRef                  INTEGER(0..255)
}

CreditRs ::= SEQUENCE {
        creditResult            Result,
        creditAuthenticator     OCTET STRING
}
DebitRq::= SEQUENCE {
        debitFee                Fee,
        nonce                   OCTET STRING,
        keyRef                  INTEGER(0..255)
```

```
}

DebitRs ::= SEQUENCE {
        debitResult                 Result,
        debitAuthenticator          OCTET STRING
}

GetInstanceRq ::= SEQUENCE {
            posOfFirstInstance      INTEGER(0..255),        --position of first instance to be retrieved
            posOfLastInstance       INTEGER(0..255),        --position last instance to be retrieved
            attributeIdList         AttrIdList              --Ids of attributes to be retrieved
}                                   -

GetInstanceRs::= SEQUENCE (0..127,...) OF               {
        attributeId                 INTEGER(0..127,...),    -- number of instances retrieved
        attributeValues             Container::=OCTET STRING -- values of the concatenated instances
}

GetStampedRq::= SEQUENCE {
        attributeIdList             AttrIdList,
        nonce                       OCTET STRING,           -- e.g. a random number
        keyRef                      INTEGER(0..255),
}

GetStampedRs::= SEQUENCE {
        attributeList               AttrList,
        authenticator               OCTET STRING
}

SetInstanceRq ::= SEQUENCE {
                    posOfInstance               INTEGER(0..255),
    attribute       Attr
}

SetMMIRq::= INTEGER {
        ok                          (0),        -- the operation / transaction successfully completed
        nok                         (1),        -- the operation / transaction not successfully completed
        contactOperator             (2),        -- contact the operator e.g. due to low balance or battery
        reservedForFutureCENUse     (3-127),
        reservedForPrivateUse       (128-255)
} (0..255)

SetStampedRq::= SEQUENCE {
        attributeList               AttrList,
        nonce                       OCTET STRING,
        keyRef                      INTEGER(0..255),
}

SubRq::= SEQUENCE {
        attributeId                 INTEGER(0..127,..),
        value                       INTEGER
}

    -- NOTE: Below the definitions of the EFC attributes

ContractSerialNumber ::= Int4

ContractAuthenticator ::= OCTET STRING
```

```
ContractValidity ::= SEQUENCE {
        contractRestrictions        OCTET STRING (SIZE(4)),
        contractExpiryDate          DateCompact
        }

ContractVehicle ::= CS4

CopType ::= ENUMERATED {
        noEntry                     (0),
        reservedForCOPValue         (1..15)     -- reserved for COP-values
        }

DescriptiveCharacteristicsType ::= ENUMERATED {
        noEntry                     (0),
        vehicleShape1               (1),        -- vehicle shape x as defined in
        vehicleShape2               (2),        -- prENV/278/8/1/5 for silhouette x
        ...
        vehicleShape50              (50),
        reservecForFutureCENUse     (51..255)
        }

DriverCharacteristics ::= SEQUENCE {
        driverClass                 INT1,
        tripPurpose                 INT1
        }

EFC-ContextMark ::= SEQUENCE {
        contractProvider            Provider,
        typeOfContract              OCTET STRING SIZE(2),
        contextVersion              INTEGER(0..127,..)
        }

EnvironmentalCharacteristicsType ::= SEQUENCE {
        euroValue                   EuroType,
        copValue                    CopValue
        }

EngineCharacteristicsType ::= ENUMERATED {      -- e.g. petrol, electric, diesel, solar..
        noEntry                     (0),
        noEngine                    (1),
        petrolUnleaded              (2),
        petrolLeaded                (3),
        diesel                      (4),
        LPG                         (5),        -- LPG
        battery                     (6),
        solar                       (7),
        reservedForFutureCENUse     (8-255)
        }

EuroType ::= ENUMERATED {
        noEntry                     (0),
        euro-1                      (1),        -- EURO-Clases as defined in EC directive
        euro-2                      (2),        -- 88/77/EEC, annex 1 and in 91/542/EEC
        euro-3                      (3),
        reservedForFutureCENUse     (4..15)
        }

EquipmentICC-Id ::= ICC-Id

EquipmentOBUId ::= OCTET STRING

EquipmentStatus ::= BIT STRING (SIZE(16))
```

```
FutureCharacteristicsType ::= ENUMERATED {
                noEntry                 (0),
                registeredMeaning       (1..255)
                }

ICC-Id ::= OCTET STRING

Int1 ::= INTEGER(0..255)
Int2 ::= INTEGER(0..65535)
Int3 ::= INTEGER(0..16777216)
Int4 ::= INTEGER(0..4294967295)

PaymentMeans ::= SEQUENCE {
                paymentMeansProvider    Company,
                paymentMeansID          Int4,
                paymentMeansExpiryDate  Date
                }

PaymentMeansBalance ::= Value

PaymentMeansUnit ::= PayUnit

PaymentSecurityData ::= OCTET STRING

Provider ::= SEQUENCE {
                countryCode             CountryCode,
                providerIdentifier      IssuerIdentifier
                }

ReceiptContract ::= SEQUENCE {
                sessionContractProvider     Provider,
                sessionTypeOfContract       OCTET STRING (SIZE(2)),
                sessionContractSerialNumber Int4
                }

ReceiptDistance ::= INT3

ReceiptFinancialPart ::= SEQUENCE {
                sessionPaymentMeansProvider     Company,
                sessionPaymentMeansID           Int4,
                sessionFee                      Fee,
                sssionCurrentBalance            PurseBalance,
                receiptFinancialSerialNumber    Int4
                }

ReceiptICC-Id ::= ICC-Id

ReceiptOBUId ::= OCTET STRING                   -- coding as described for ICC-Id

ReceiptServicePart ::= SEQUENCE {
                sessionTime             Time,
                sessionServiceProvider  Provider,
                stationLocation         INTEGER(0..1048575),
                sessionLocation         BIT STRING (SIZE(8)),
                typeOfSession           StationType,
                sessionResultOperational Result,
                sessionResultFinancial  Result,
                }

ReceiptServiceSerialNumber ::= Int3

ReceiptAuthenticator ::= OCTET STRING

ReceiptText ::= OCTET STRING

SessionClass ::= SEQUENCE {
                sessionTariffClass      Int1,
                sessionClaimedClass     Int1
                }
```

```
StationType ::= ENUMERATED {
        unspecified                     (0),
        closedEntryWithPayment          (1),
        closedEntryWithoutPayment       (2),
        closedTransit                   (3),
        closedExit                      (4),
        closedCredit                    (5),
        mixed                           (6),
        passage                         (7),        -- open exit
        checkpoint                      (8),
        reload                          (9),
        reservedForFutureCENUse         (10-13),
        privateUse                      (14-15)
        }

Time ::= SEQUENCE {
        timeDate        DateCompact,                -- defined in ENV 1545-1
        timeCompact     TimeCompact                 -- defined in ENV 1545-1
        }

TransportApplicationData::= CHOICE {    -- the components of CHOICE defined in ENV 1545-1/2
        account                 [0]     Account,
        amount                  [1]     Amount,
        capacityUnit            [2]  CapacityUnit,
        classQualifier          [3]     ClassQualifier,
        company                 [4]     TimeCompact,
        contract                [5]     Contract,
        contractCustomerInfo    [6]     ContractCustomerInfo,
        contractJourneyData     [7]     ContractJourneyData,
        contractPassengerInfo   [8]     ContractPassengerInfo,
        contractRestriction     [9]     ContractRestriction,
        contractTariff          [10]    ContractTariff,
        contractValidityInfo    [11]    ContractValidityInfo,
        contractGuaranteeInfo   [12]    ContractGuaranteeInfo,
        contractSaleData        [13]    ContractSaleData,
        countryAlpha            [14]    CountryAlpha,
        countryNumeric          [15]    CountryNumeric,
        customerProfile         [16]    CustomerProfile,
        dateContract            [17]    DateContract,
        datef                   [18]    Datef,
        holder                  [19]    Holder,
        holderBirth             [20]    HolderBirth,
        holderName              [21]    HolderName,
        holderProfile           [22]    HolderProfile,
        iai                     [23]    Iai,
        locationReference       [24]    LocationReference,
        networkId               [25]    NetworkId,
        passengerClass          [26]    PassengerClass,
        payEventCode            [27]    PayEventCode,
        payLocationType         [28]    PayLocationType,
        paymentCertificate      [29]    PaymentCertificate,
        paymentEvent            [30]    PaymentEvent,
        paymentFee              [31]    PaymentFee,
        payMethod               [32]    PayMethod,
        paymentReceipt          [33]    PaymentReceipt,
        payUnit                 [34]    PayUnit,
        price                   [35]    Price,
        purseContext            [36]    PurseContext,
        purseValue              [37]    PurseValue,
        receipt                 [38]    Receipt,
        restrictDOW             [39]    RestrictDOW,
        result                  [40]    Result,
        roadTollContract        [41]    RoadTollCotract,
        roadTollEvent           [42]    RoadTollEvent,
        services                [43]    Services,
        serviceReceipt          [44]    ServiceReceipt,
        status                  [45]    Status,
        timeCompact             [46]    TimeCompact,
```

```
                timeReal                [47]        TimeReal,
                versionNumber           [48]        VersionNumber,
                value                   [49]        Value,
                reservedForFutureUse    [50-255]    -- reserved for future use,
                }

VehicleAuthenticator ::= OCTET STRING

VehicleAxles ::= SEQUENCE {
                vehicleFirstAxleHeigth  Int1,
                vehicleAxlesNumber      Int1
                }

VehicleClass ::= Int1

VehicleDimensions ::= SEQUENCE {
                vehicleLengthOverall    Int1,
                vehicleHeigthOverall    Int1,
                vehicleWidthOverall     Int1
                }

VehicleLicencePlateNumber ::= CS4

VehicleSpecificCharacteristics ::= VehicleSpecificCharacteristicsType

VehicleSpecificCharacteristicsType ::= SEQUENCE {
                environmentalCharacteristics    EnvironmentalCharacteristicsType,
                engineCharacteristics           EngineCharacterisitcsType,
                descriptiveCharacteristics      DescriptiveCharacteristicsType,
                futureCharacteristics           FutureCharacteristicsType
                }                               -- Assignment of meaning to the unassigned enumerated
                                                -- values is subject to registration according to the registration
                                                -- procedures specified in the DSRC-L7 standard.

VehicleWeightLaden ::= INT2

VehicleWeightLimits ::= SEQUENCE {
                vehicleMaxLadenWeight       Int2,
                vehicleTrainMaximumWeight   Int2,
                vehicleWeightUnladen        Int2
                }
END
```

Annex B (informative) An excerpt from DSRC application layer

This annex provides an excerpt from DSRC Application Layer (ENV 12834), and its service primitives (including format and parameters), the generic DSRC application layer functions (GET and SET) and the Container CHOICE ASN.1 type definition. ENV 12834 supersedes this European Pre-Standard in case of discrepancy, as this excerpt is informative and is included only so as to ease the reading of this European Pre-Standard.

B.1 Format of service primitives

The format for the T-ASDUs adheres to the definitions given in ENV 12834 (clause 6.2.2, T-KE Services).

The generic format of service primitives of GET, SET and ACTION are as shown in tables B.1, B.2 and B.3 respectively. The meaning of characters used in tables B.1-3 are given in table B.4.

Table B.1: The generic format of the GET service primitives

parameter name	request	indication	response	confirm	ASN.1 type
Invoker Identifier (IID)	optional	optional =	optional	optional =	Dsrc-EID
Link Identifier (LID)	mandatory	mandatory =	mandatory	mandatory =	BIT STRING
Element Identifier (EID)	mandatory	n.a.	I	n.a.	Dsrc-EID
AccessCredentials	optional	optional=	n.a.	n.a.	OCTET STRING
Attribute Id List (AttrIdList)	optional	optional =	n.a.	n.a.	AttributeIdList
FlowControl	mandatory	mandatory	mandatory	n.a.	INTEGER
Attribute List (AttrList)	n.a.	n.a.	optional	optional =	AttributeList
Return Code (Ret)	n.a.	n.a.	optional	optional =	ReturnStatus

Table B.2: The generic format of the SET service primitives

parameter name	request	indication	response	confirm	ASN.1 type
Invoker Identifier (IID)	optional	optional =	optional	optional =	Dsrc-EID
Link Identifier (LID)	mandatory	mandatory =	mandatory	mandatory =	BIT STRING
Element Identifier (EID)	mandatory	n.a.	I	n.a.	Dsrc-EID
AccessCredentials	optional	optional=	n.a.	n.a.	OCTET STRING
Attribute List (AttrList)	mandatory	mandatory	n.a.	n.a.	AttributeList
Mode	mandatory	mandatory=	n.a.	n.a.	Boolean
FlowControl	mandatory	mandatory	mandatory	n.a.	INTEGER
Return Code (Ret)	n.a.	n.a.	optional	optional =	ReturnStatus

Table B.3: The generic format of the ACTION service primitives

parameter name	request	indication	response	confirm	ASN.1 type
Invoker Identifier (IID)	optional	optional =	optional	optional =	Dsrc-EID
Link Identifier (LID)	mandatory	mandatory =	mandatory	mandatory =	BIT STRING
Element Identifier (EID)	mandatory	n.a.	I	n.a.	Dsrc-EID
ActionType	mandatory	mandatory	n.a.	n.a.	INTEGER(0..127,...)
AccessCredentials	optional	optional=	n.a.	n.a.	OCTET STRING
ActionParameter	optional	optional =	n.a.	n.a.	Container
Mode	mandatory	mandatory=	n.a.	n.a.	Boolean
FlowControl	mandatory	mandatory	mandatory	n.a.	INTEGER
ResponseParameter	n.a.	n.a.	optional	optional =	Container
Return Code (Ret)	n.a.	n.a.	optional	optional =	ReturnStatus

Table B.4: The meaning of characters used in tables B.1-3

I	Mandatory and IID of related indication if present, else Dsrc-EID of service
=	Present and same as in request and response for indication and confirm, respectively
n.a.	Not applicable

Parameters

The parameters of the GET, SET and ACTION service primitives are determined and interpreted as follows.

IID shall be the DSRC-EID of the element initiating the request or the response, respectively. This parameter is not needed if an answer shall be sent to a default invoker being the EID of the request service primitive. If IID is used, it shall give the EID for a response to this primitive. IID may be needed when executing several applications simultaneously.

LID shall be the (communication) link address of the OBU, and is chosen by the I-KE on vehicle side as specified in ENV 12834 clause 6.3. The LID has one single value during any DSRC session.

EID shall be the DSRC-EID of the element which shall receive the indication or confirm related to a request or response, respectively. This EID is used by the T-KE on the side of the receiver to deliver the indication or confirm to the addressed element. When the IID is used in a request the element invoking a response shall use this IID as the EID. EID is the logical context identifier associated with the EFC-ContextMark as exchanged in the VST (see further clause 6). EID = 0 shall be reserved for addressing of application-independent functions and components (see further clause 5.3).

AccessCredentials shall be of OCTET STRING ASN.1 type and carry the information needed to fulfil the access conditions in order to perform the EFC function on the addressed element.

EXAMPLE: Access credentials may carry a password that needs to be presented in order to retrieve the values of the addressed attribute (i.e. reading of data).

AttrIdList shall be a list of IDs of attributes of the element receiving a DSRC Application Layer service primitives.

FlowControl shall be a parameter which represents the behaviour of the underlying communication service. This parameter shall be mapped (as defined by ENV 12834) by the T-KE on a special logical link control service .

AttrList shall be a sequence of Attributes sent by a DSRC Application Layer service primitive.

Ret shall be a special return code issued by an element as an answer to a service.indication The following codes are predefined in ENV 12834:

	accessDenied:	the requested operation was not performed for reasons pertinent to the security of the system. The Response Parameter for EFC action shall be empty.
	argumentError:	one or more attribute values were not accessed because the identifier for the specified attribute was not recognised or the attribute value specified was out of range or otherwise inappropriate for one or more attributes or the action was not supported by the receiving entity.
	complexityLimitation:	the requested operation was not performed because a parameter was too complex.
processingFailure:	a general failure in processing the operation was encountered.	
	processing:	the requested operation is being processed, and the result is not yet available.

Mode shall be a Boolean parameter indicating whether there shall be a service.response to a service.indication.

ActionType shall identify an operation, which shall be invoked by an element which receives an ACTION.indication.

ActionParameter shall be the information needed for the invocation of an operation identified in an ACTION.indication.

ResponseParameter may be information resulting from the execution of the operation invoked by ACTION.indication.

B.2 Generic DSRC application layer functions

B.2.1 GET

GET is used to retrieve (i.e. read) value(s) of the addressed attribute(s), a reply is always expected.

Table B.5: GET.request

parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier (EID)	Dsrc-EID		
AccessCredentials	OCTET STRING		optional use
Attribute Id List (AttrIdList)	AttributeIdList		Always to be present

GET.request shall request the retrieval of the value(s) of the addressed attribute(s) in the Attribute Id List, a reply shall always be expected.

Table B.6: GET.response

Parameter name	ASN.1 type	Value	Remark / Constraints
Attribute List (AttrList)	AttributeList		Always used, unless error
Return Code (Ret)	ReturnStatus		optional use

GET.response shall carry the retrieved value(s) of the addressed attribute(s) or/and the result of the corresponding GET.request command. The Attribute List, if present, shall contain the attributes retrieved. If not all the addressed attributes were retrieved, a failure shall be indicated in the Return Code.

B.2.2 SET

SET is used to set (i.e. write) value(s) of the addressed attribute(s).

Table B.7: SET.request

Parameter name	ASN.1 type	Value	Remark / Constraints
Element Identifier (EID)	Dsrc-EID		
AccessCredentials	OCTET STRING		optional use
Attribute List (AttrList)	AttributeList		Always to be present
Mode	BOOLEAN		

SET.request can be used in confirmed or non-confirmed mode (i.e. Mode equal to True or False, respectively), a reply shall always be expected in the former case.

Table B.8: SET.response

parameter name	ASN.1 type	Value	Remark / Constraints
Return Code (Ret)	ReturnStatus		optional use

SET.response shall explicitly convey the result of the corresponding SET.request command If not all the addressed attributes were set then a failure shall be indicated in the Return Code.

B.3 Container ASN.1 type definition

This clause accounts for the Container ASN.1 type definition as defined in ENV 12834.

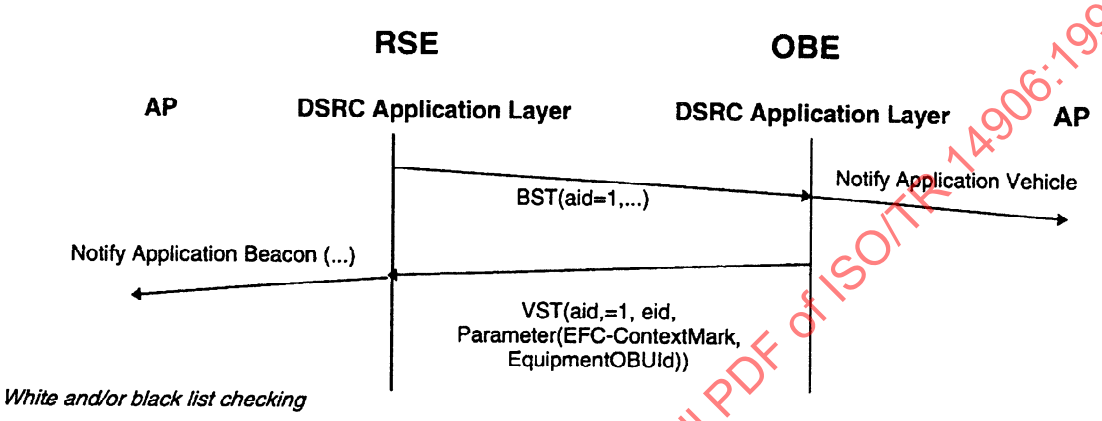
```
DSRCData-P1 DEFINITIONS::= BEGIN
 IMPORTS
                                        ContainerJ.yFROM ApplicationJ        --this line
shall be given for each application which -- defines data
of type container, J and y shall be
                                        -- replaced by an unambiguous suffix
                                        RecordJ.yFROM ApplicationJ   --this line shall be
given for each application which
                                        -- defines data of type record, J and y shall be
replaced
                                        -- by an unambiguous suffix

Container::=CHOICE{
        integer                 [0]     INTEGER,
        bitstring               [1]     BIT STRING,
        octetstring             [2]     OCTET STRING,
        universalString         [3]     UniversalString,
        beaconId                [4]     BeaconID,
        t-apdu                  [5]     T-APDUs,
        dsrcApplicationEntityId [6]     DSRCApplicationEntityID,
        dsrc-Ase-Id             [7]     Dsrc-EID,
        attrIdList              [8]     AttributeIdList
        attrList                [9]     AttributeList
        broadcastPool           [10]    BroadcastPool,
        directory               [11]    Directory,
        file                    [12]    File,
        fileType                [13]    FileType,
        record                  [14]    Record,
        time                    [15]    Time,
        vector                  [16]    SEQUENCE (0..255) OF INTEGER(0..127,...),
        dummy                   [17..127]  ReservedForFutureCENUse,

                                        , contI.x       [i]     ContainerI.x    --this line
shall be given for each imported
                                        -- ContainerI.x, where I.x is replaced by the related
                                        -- suffixand i is the registered tag starting with 0.
                                        -- Gaps shall befilled with contI.x [i] BIT STRING

}
```

Annex C (informative) Examples of EFC transactions using the EFC application interface

This annex provides informative examples of EFC transactions that conform to this European Pre-Standard.

C.1 Example of an EFC transaction - Example 1

The example described below is based on a centrally held account transaction without any dynamic security measure:



C.2 Example of an EFC transaction - Example 2

The example described below is based on a centrally held account transaction, without any dynamic security measure, performed in at an exit station within a closed system.