INTERNATIONAL STANDARD

ISO 22901-1

First edition
2008-11-15

Road vehicles — Open diagnostic data exchange (ODX) —

Part 1: Data model specification

Véhicules routiers — Échange de données de diagnostic ouvert (ODX) —

Partie 1: Spécification de modèle de données

Reference number
ISO 22901-1:2008(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 22901-1 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

ISO 22901 consists of the following parts, under the general title *Road vehicles — Open diagnostic data exchange (ODX)*:

— *Part 1: Data model specification*

The following parts are under preparation:

— *Part 2: Emissions-related diagnostic data*

Introduction

The purpose of this part of ISO 22901 is to define the data format for transferring Electronic Control Unit (ECU) diagnostic and programming data between the system supplier, vehicle manufacturer and service dealerships and diagnostic tools of different vendors.

In today's automotive industry, an informal description is generally used to document the diagnostic data stream information of vehicle ECUs. Any user wishing to use the ECU diagnostic data stream documentation to set up development tools or service diagnostic test equipment needs a manual transformation of this documentation into a format readable by these tools. This effort will no longer be required if the diagnostic data stream information is provided in Open Diagnostic Data Exchange (ODX) format and if those tools support the ODX format.

This part of ISO 22901 includes the data model definition of ECU diagnostic and programming data and the related vehicle interface description in Unified Modelling Language (UML). This part of ISO 22901 also includes an implementation by Extensible Mark-up Language (XML) schema in Annex C.

Road vehicles — Open diagnostic data exchange (ODX) —

Part 1:
Data model specification

1 Scope

This part of ISO 22901 specifies the concept of using a new industry standard diagnostic format to make diagnostic data stream information available to diagnostic tool application manufacturers, in order to simplify the support of the aftermarket automotive service industry. The Open Diagnostic Data Exchange (ODX) modelled diagnostic data are compatible with the software requirements of the Modular Vehicle Communication Interface (MVCI), as specified in ISO 22900-2 and ISO 22900-3. The ODX modelled diagnostic data will enable an MVCI device to communicate with the vehicle Electronic Control Unit(s) (ECU) and interpret the diagnostic data contained in the messages exchanged between the external test equipment and the ECU(s). For ODX compliant external test equipment, no software programming is necessary to convert diagnostic data into technician readable information to be displayed by the tester.

The ODX specification contains the data model to describe all diagnostic data of a vehicle and physical ECU, e.g. diagnostic trouble codes, data parameters, identification data, input/output parameters, ECU configuration (variant coding) data and communication parameters. ODX is described in Unified Modelling Language (UML) diagrams and the data exchange format uses Extensible Mark-up Language (XML).

The ODX modelled diagnostic data describe:

— protocol specification for diagnostic communication of ECUs;

— communication parameters for different protocols and data link layers and for ECU software;

— ECU programming data (Flash);

— related vehicle interface description (connectors and pinout);

— functional description of diagnostic capabilities of a network of ECUs;

— ECU configuration data (variant coding).

Figure 1 shows the usage of ODX in the ECU life cycle.

The purpose of this part of ISO 22901 is to ensure that diagnostic data from any vehicle manufacturer is independent of the testing hardware and protocol software supplied by any test equipment manufacturer.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC 8859-1, *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*

ISO/IEC 8859-2, *Information technology — 8-bit single-byte coded graphic character sets — Part 2: Latin alphabet No. 2*

ISO/IEC 10646, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

ISO 22900-2, *Road vehicles — Modular vehicle communication interface (MVCI) — Part 2: Diagnostic protocol data unit application programming interface (D-PDU API)*

ISO 22900-3, *Road vehicles — Modular vehicle communication interface (MVCI) — Part 3: Diagnostic server application programming interface (D-Server API)*

IEEE 754, *Binary floating-point arithmetic*

XML Schema — 2, *XML Schema Part 2: Datatypes, 2nd Edition, W3C Recommendation, 2004-10-28*

ASAM MCD 2, *Harmonized Data Objects Version 1.0*

3 Abbreviated terms

API	Application Programming Interface
ASAM	Association for Standardisation of Automation and Measuring Systems
ASCII	American Standard for Character Information Interchange
DOP	Data Object Property
ECU	Electronic Control Unit
GMT	Greenwich Mean Time
MCD	Measurement, Calibration and Diagnosis
ODX	Open Diagnostic Data Exchange
OEM	Original Equipment Manufacturer
PDU	Protocol Data Unit
PDX	Packaged ODX
UML	Unified Modelling Language
UTC	Coordinated Universal Time
VMM	Vehicle Message Matrix
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language

4 ODX use cases

4.1 General

Figure 1 — Usage of ODX data in the ECU life cycle shows the usage of ODX in the ECU life cycle. Engineering, manufacturing, and service specify communication protocol and data to be implemented in the ECU. This information will be documented in a structured format utilizing the XML standard and by an appropriate ODX authoring tool. There is potential to generate ECU software from the ODX file. Furthermore, the same ODX file is used to setup the diagnostic engineering tools to verify proper communication with the ECU and to perform functional verification and compliance testing. Once all quality goals are met, the ODX file may be released to a diagnostic database. Diagnostic information is now available to manufacturing, service, OEM franchised dealers, and aftermarket service outlets via Intranet and Internet.

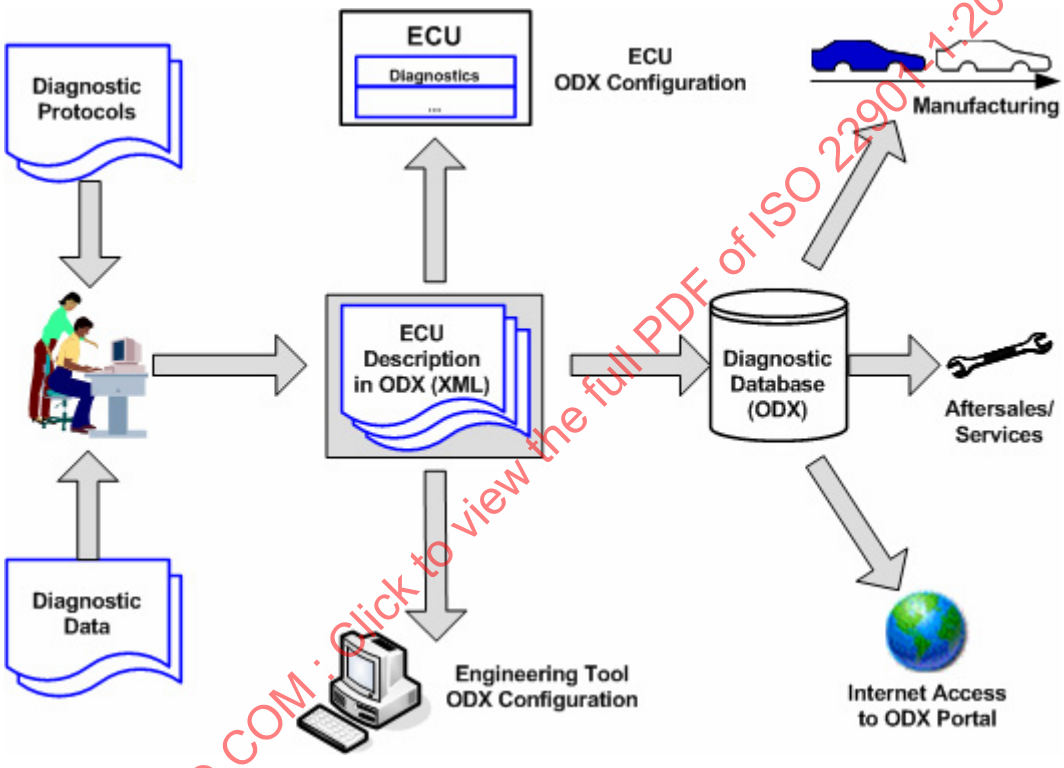


Figure 1 — Usage of ODX data in the ECU life cycle

4.2 Use case 1: ODX process chain

Figure 2 shows an example of how ODX data is used in a process chain consisting of three phases, as described below.

a) Phase A of the development process between vehicle manufacturer and system supplier comprises the exchange of ODX data to support the development of the diagnostic implementation in the ECU and the development tools.

b) In phase B of the development process at the vehicle manufacturer, the engineering departments release the ODX data into a diagnostic database. The manufacturing and service departments use the ODX data as the basis to setup the End-Of-Line test equipment and service application development tools and generate service documentation.

c) Phase C of the development process supports the service dealership diagnostic and programming tools. The service department develops service tool application software based on the ODX data model. The diagnostic and programming software is now available to all service dealerships.

The ODX data is the base for all exchange of diagnostic and programming data.

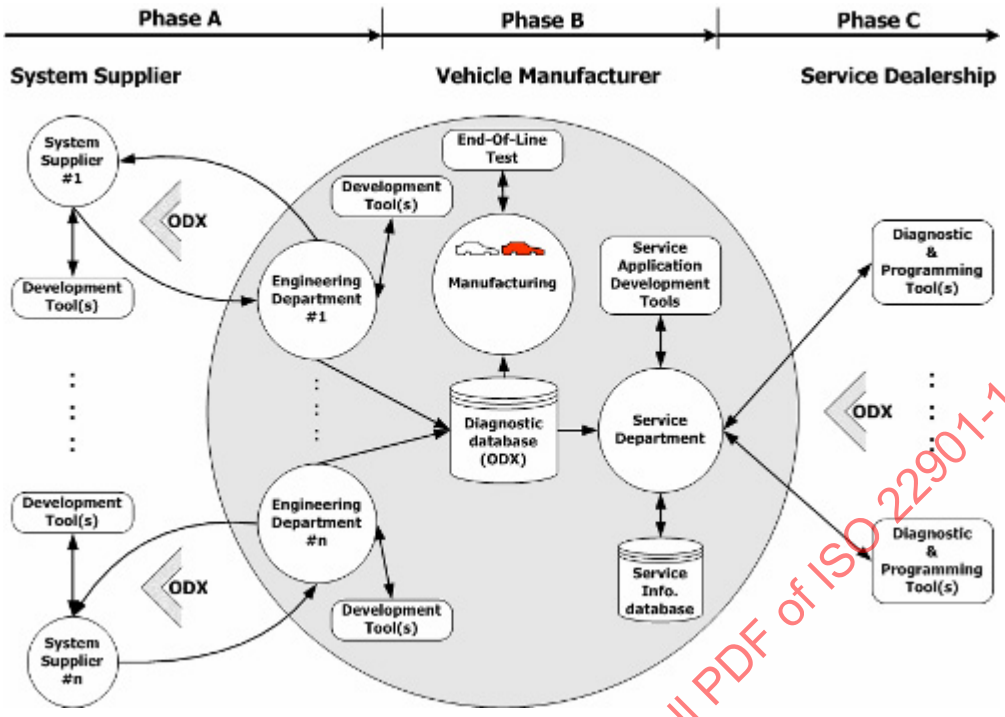


Figure 2 — Example of ODX process chain

4.3 Use case 2: Cross vehicle platform ECU diagnostic development

A vehicle manufacturer implements electronic systems into multiple new vehicle platforms. There is little variation in the electronic system across the different vehicle platforms. Utilizing the same ECU in many different vehicle platforms reduces redundant development effort. The majority of design, normal operation, and diagnostic data of an electronic system can be reused in various vehicles.

Large automotive manufacturer tend to have multiple engineering development centres. Diagnostic data exchange can be based on the ODX data format to reduce the amount of proofreading of diagnostic data at different development sites. Establishing an ODX compliant tool chain will avoid re-authoring diagnostic data into various specific formats at different engineering sites.

Figure 3 shows an example of cross vehicle platform ECU diagnostic development between two engineering sites.

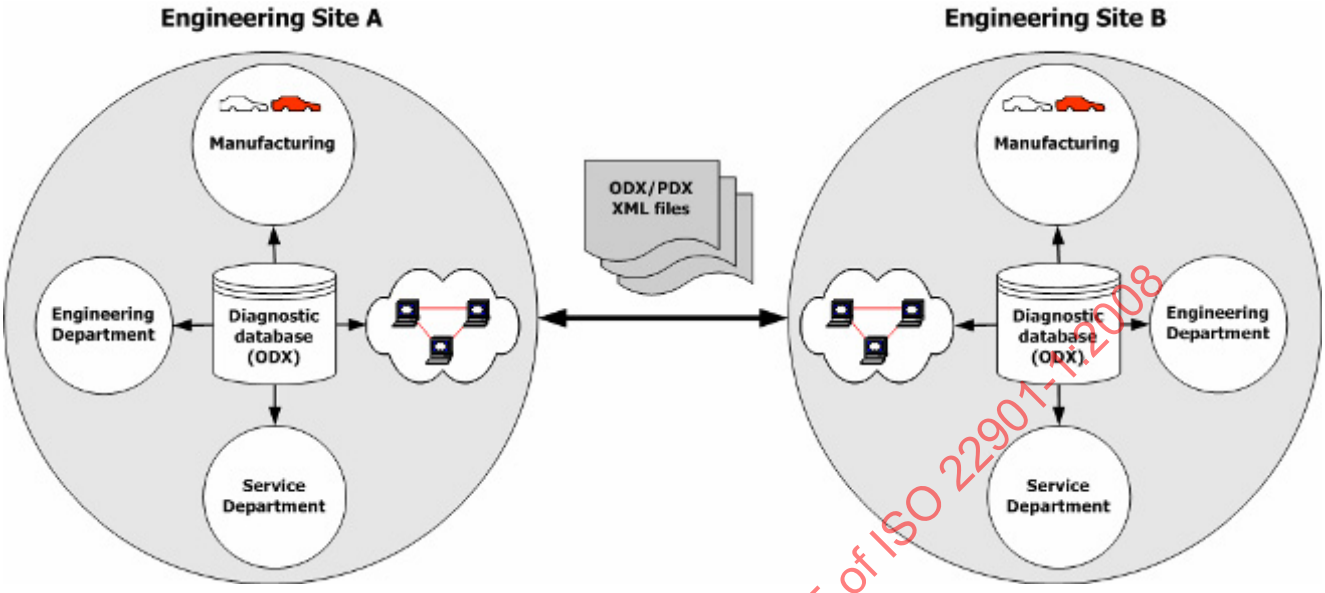


Figure 3 — Example of cross vehicle platform ECU diagnostic development

4.4 Use case 3: Franchise and aftermarket service dealership diagnostic tool support

Figure 4 shows one of many scenarios a vehicle manufacturer may implement to support the service dealership, franchise and aftermarket. ODX files may be converted into an ODX runtime format for download to the dealership diagnostic system.

IMPORTANT — The ODX runtime data format may be different between many diagnostic and programming applications and tools. In such cases, an ODX runtime converter may be used to support the data conversion to dealership specific test equipment.

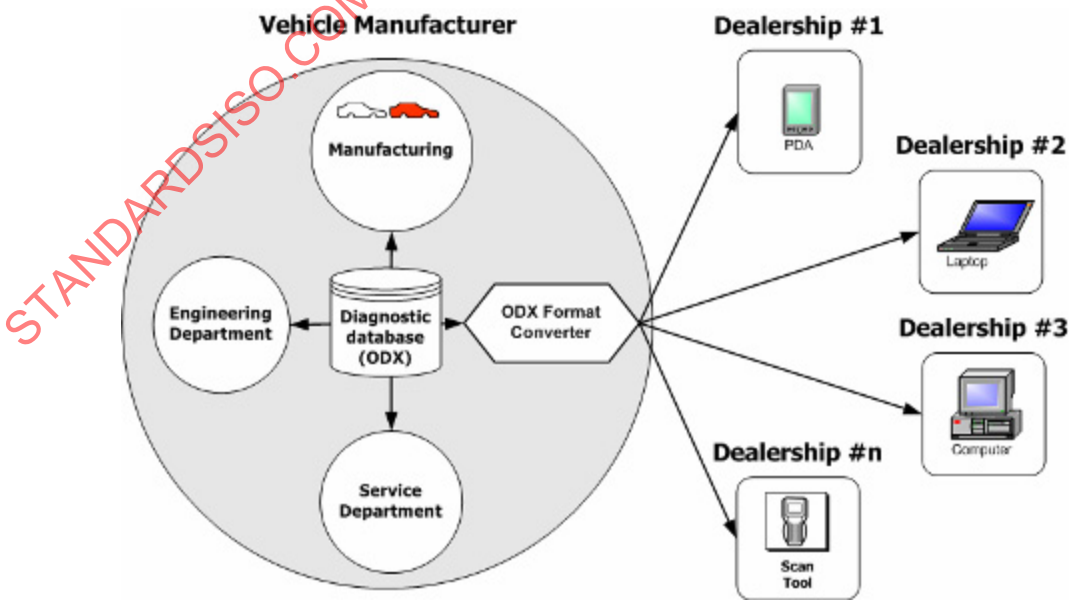


Figure 4 — Example of automotive dealership diagnostic tool support

4.5 Architecture of a Modular VCI compliant D-server

Figure 5 shows the interfaces of a D-server and the position of ODX in a diagnostic system. The D-server may use its own internal specific runtime data format. This data can be imported from the ODX using a specific format converter.

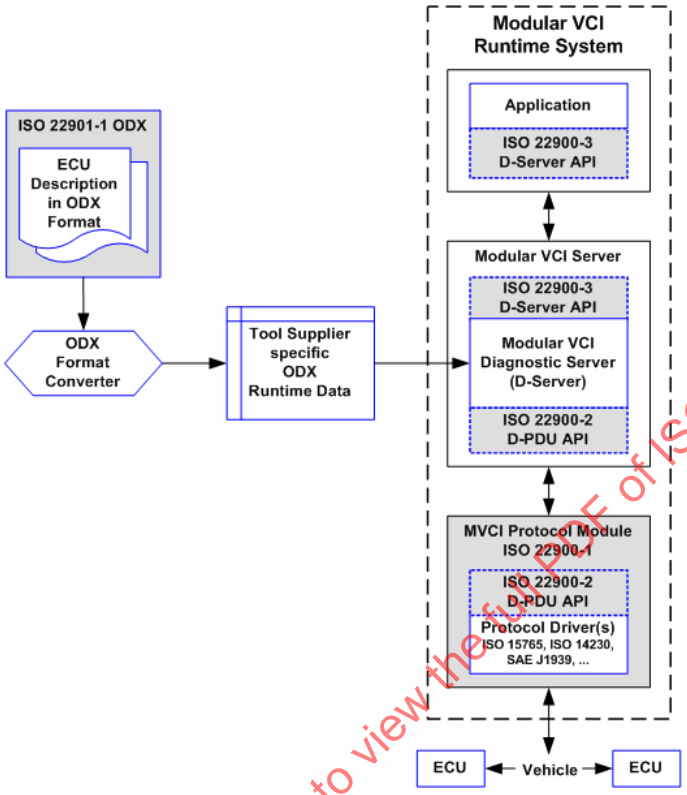


Figure 5 — Architecture of a Modular VCI compliant D-server

4.6 ODX benefit examples

4.6.1 ECU system supplier

The following benefits are applicable to the ECU system supplier:

— automatic configuration of ECU diagnostic data stream & protocol;

— documentation is generated from XML data format (ECU diagnostic content = documentation);

— automatic configuration of development tester to verify ECU diagnostic behaviour;

— XML data format provides machine readable information to import into supplier diagnostic data base;

— generation of source code to configure diagnostic kernel software components.

4.6.2 Vehicle manufacturer engineering

The following benefits are applicable to the vehicle manufacturer engineering:

— reduction of diagnostic data stream authoring effort;

— various development testers can be supported with “single source” data;

— one single file format for import and export into/from diagnostic database.

4.6.3 Vehicle manufacturer production

The following benefits are applicable to the vehicle manufacturer production:

— reduced effort for diagnostic data verification, because verification needs to be performed only once;

— reuse of verified diagnostic data;

— fewer errors because of fewer manual process steps;

— end-of-line tester uses the same diagnostic data as engineering diagnostic tester.

4.6.4 Vehicle manufacturer service department and dealerships

The following benefits are applicable to the vehicle manufacturer service department and dealerships:

— more convenient reuse of verified diagnostic data;

— less cost to distribute diagnostic data;

— “pull” (via Intranet/Internet) diagnostic data (e.g. from a portal into a D-server) versus “push” (e.g. send CD ROMs);

— one single file format for various diagnostic service systems.

4.6.5 Test equipment manufacturer

The following benefits are applicable to the test equipment manufacturer:

— less effort needed to implement high quality scan tool software by using a generic data driven approach;

— focus on “rich diagnostic application(s)” versus bits & bytes;

— more convenient reuse of vehicle manufacturer verified diagnostic data.

4.6.6 Franchise and aftermarket dealerships

The following benefits are applicable to the franchise and aftermarket dealerships:

— more convenient reuse of vehicle manufacturer verified diagnostic data;

— tester configuration by data download instead of by software modification;

— download on demand versus buying tester software update upfront.

4.6.7 Legal authorities

The following benefits are applicable to the legal authorities:

— standardized description format for enhanced diagnostic data documentation (e.g. DTCs, PIDs);

— enables road-side scan tools and I&M (inspection and maintenance) tools to be vehicle manufacturer independent;

— fulfils requirement of making enhanced diagnostic data available to independent aftermarket.

5 Specification release version information

5.1 Specification release version location

The release version of the ODX standard can be obtained from every ODX file instance. It is contained in the MODEL-VERSION element.

```
<ODX xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=" odx.xsd" MODEL-
VERSION="2.2.0">
```

5.2 Specification release version

The specification release version of this part of ISO 22901 is: 2.2.0.

6 Introduction to and use of Unified Modelling Language (UML)

6.1 General aspects

Unified Modelling Language (UML) is used to define the ODX data model formally and unambiguously. It enhances readability by graphical data model diagrams.

A short introduction to UML is given in this clause. Only those aspects of UML are described that are used in the ODX data model. Specifically, from the large set of available UML diagrams, class diagrams are only applied for data modelling.

6.2 Class diagrams

6.2.1 Class

The central modelling element in UML is a class. A class represents a set of similar objects. Generally speaking, a class can be instantiated many times. Every instance of a class is called an object. A class has attributes (defining the properties of these objects) and methods (defining the actions an object can perform). For ODX, methods are irrelevant and are not used.

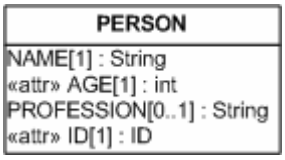


Figure 6 — UML representation of class

Figure 6 shows the representation of a class and its attributes in UML notation. A class is symbolized by a rectangle having up to three fields. The top field contains the name of the class, e.g. PERSON. The second field contains the attributes of the class, e.g. NAME, AGE, PROFESSION, and ID. Methods are defined in an optional third field. The attribute field is not always displayed in the ODX data model diagrams. Since the method-field is unused in the ODX data model, it is never displayed.

Attributes consist of the attribute name, e.g. NAME, followed by the attribute's cardinality, e.g. [1] or [0...1] and the attribute type, e.g. String. Furthermore, a default value for the attribute may be specified. Such a default value follows the type descriptor of the attribute and is separated from it by the symbol "=".

UML attributes specified with the <<attr>> stereotype in front of their name are implemented as XML attributes of XML elements. UML attributes without this stereotype are implement as XML sub-elements of XML elements.

Throughout the ODX data model, class names and attribute names are written in capital letters.

6.2.2 Inheritance relationships

Classes can inherit attributes from other classes. In Figure 7, a new class SCHOOLKID is derived from the class PERSON. This means that implicitly the class SCHOOLKID has all the same attributes as PERSON plus those that are defined specifically for the new class SCHOOLKID, e.g. GRADE. PERSON is called the parent or super class; SCHOOLKID is called the child or subclass of the inheritance relationship. Because the subclass adds more detail to the super class and is thus more specific, inheritance relationships are often called “specializations”.

Inheritance relationships can be used to build inheritance trees of arbitrary depth. A class in such a tree inherits all attributes from those classes in the transitive closure of all ancestors (parents, grandparents, etc.) in the inheritance tree.

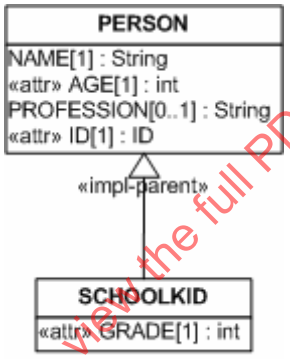


Figure 7 — UML representation of inheritance relationship

6.2.3 Aggregation and composition relationships

Besides the inheritance relationship, a pair of classes may also have an aggregation or a composition relationship. Aggregation or composition relationships are used if an object of one class is contained in an object of another class. They are drawn as a line with a diamond at the end of the containing class. A composition relationship’s diamond is filled; an aggregation’s diamond is not.

The difference between these two kinds of relationships is as follows:

a) the contained element in a composition relationship is a part of the containing element;

b) if the containing element is deleted, the contained element no longer exists.

Therefore, composition means an object may only be contained in one other object. An aggregation relationship is one of shared objects. This means that multiple objects may aggregate the same object. Consequently, an aggregated object still exists, even if the aggregating object is deleted[1)].

1) In the special case where the data model is implemented in XML (see below), aggregation and composition relationships are both mapped onto a sub element relationship between the two model elements. However, the UML semantics guide prohibits multiple classes from having a composition relationship to the same class. Therefore, aggregation relationships are used instead, even though the implementation in XML does not differ.

Figure 8 gives an example of three classes with composition and aggregation relationships defined. A PERSON may have two feet (two objects of type FOOT). A foot is generally an integral part of its owner. Therefore, a composition relationship is used. If the PERSON no longer exists, nor do the feet. By contrast, a PERSON may also have a lot of COATs. However, a COAT may generally be used by multiple PERSONs and its life-cycle is generally independent of the life-cycle of its wearers. Consequently, the relationship between a PERSON and a COAT is modelled as an aggregation relationship.

Composition relationships may carry cardinalities, as shown in Figure 8. The following cardinalities are common in UML:

— 1 exactly one (mandatory);

— 0...1 zero or one (optional);

— 1...* one or more;

— 0...* or * zero or more.

More specific cardinalities may be specified, e.g. 5 or 2...8.

Cardinalities are read as follows: To specify how many objects of class PERSON are related to one object of class FOOT, the cardinality at the PERSON end of the relationship is use. Vice versa the cardinality at the FOOT end of the relationship is used. This yields the following result: one object of class FOOT may be related to exactly one object of class PERSON (a foot always belongs to one and only one person) and one object of class PERSON shall be related to exactly two objects of class FOOT (a person usually has two feet).

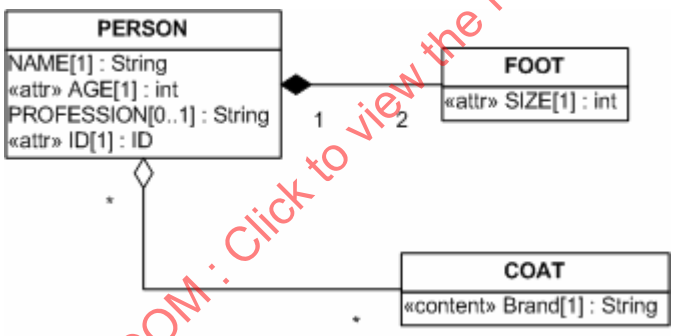


Figure 8 — UML representation of composition and aggregation relationships

6.2.4 Association relationships

The third kind of class relationship used within the ODX data model is the association relationship. It is drawn as a simple line between two classes. An association relationship has weaker semantics than the composition relationship. Here, both objects have “equal rights”. An association relationship opens visibility onto the attributes of the related objects. Figure 9 gives an example of an association. In an association, objects of one class may be related to one or more objects of the other class. To restrict the number of associations of the same type between two classes, cardinalities are used in the same way as for compositions.

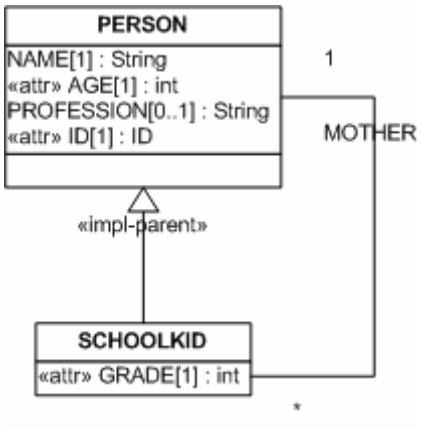


Figure 9 — UML representation of association relationship

Associations may carry a name and a role name can be given at every association end. In Figure 9, the PERSON class carries the role MOTHER in a relationship to a SCHOOLKID. The cardinalities state that a SCHOOLKID has exactly one mother, whereas a PERSON may be MOTHER or an arbitrary number of SCHOOLKIDs.

An association relationship may carry an arrow at one association end. This makes it possible to specify which of the objects may actually access attributes of the other.

6.2.5 Association classes

An association class is a hybrid of an association and a class. Generally speaking, it can be interpreted as an association carrying its own attributes (and methods). It is drawn like an association with a class symbol being connected to the centre of the association line with a dashed line (see Figure 10).

The example of a PERSON being employed by an EMPLOYER is modelled as an association class in Figure 10. Here, the employment relationship has an attribute named SALARY. This employment-specific salary is clearly not a property of a PERSON, because one person may have multiple salaries. In addition, the salary may not exist, if the PERSON is unemployed. It is also not a property of the EMPLOYER, because the EMPLOYER has many different employee-specific salaries. This is the best indication that the SALARY is a property of an employment relationship.

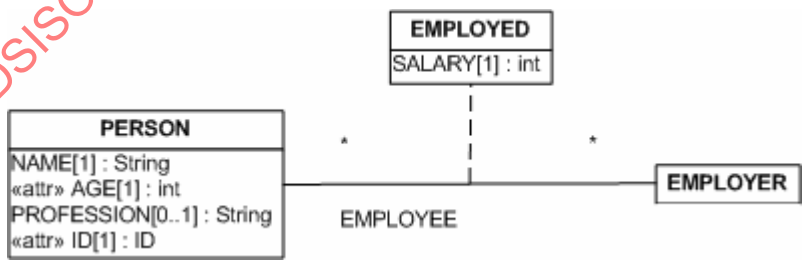


Figure 10 — UML representation of association class

6.2.6 Interfaces

An interface makes certain properties of an object available to other objects.

Interfaces are a special means to group certain aspects of a class and make other classes dependent on only one of these aspects, instead of the class as a whole.

An interface symbol looks identical to a class symbol. It can be distinguished from class through the stereotype <<interface>> within the symbol's name field.

Figure 11 contains a small example of how interfaces are used within a model. A class COMPANY implements two interfaces: EMPLOYEE and COMPETITOR. A person being employed by the company will not observe the company as a competitor and a competing company will not observe the competitor as an employer. If a class implements an interface, this is shown by a dashed arrow, similar to the inheritance arrow. A tester class of an interface (e.g. PERSON) may simply use associations or aggregations to relate to an interface. It is important to know that the tester class (PERSON) may only use those aspects of a class implementing the EMPLOYER interface that are defined within this interface. Therefore, a company also may equally employ a person by an administrative body, because the latter also implements the interface EMPLOYEE.

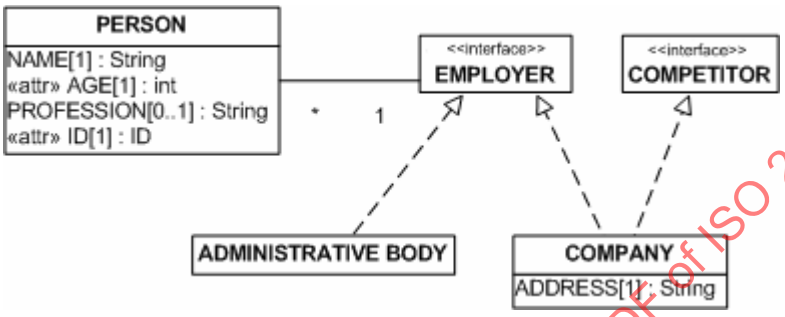


Figure 11 — UML representation of interfaces

6.2.7 Constraints

The UML makes it possible to define constraints on the model elements used. One constraint that is used frequently is the {xor} constraint between two associations (or aggregations). It enforces that an instance of one class has a relationship with instances of one or the other associated class, but never both at the same time.

Figure 12 displays an example of such a constraint. A person wears either two SHOEs or two BOOTs at any one time, but never both together.

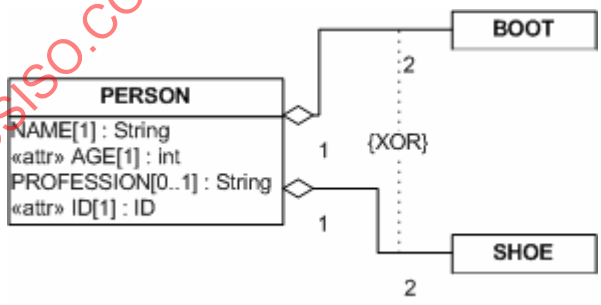


Figure 12 — UML representation of constraint

6.3 Mapping to XML

The target implementation format of the ODX exchange format is XML. This subclause explains briefly how the UML model is mapped onto XML language elements. It is not comprehensive, but rather serves as an example to simplify comprehension of the XML examples given throughout this part of ISO 22901. The complete XML mapping rules and the resulting XML schema are described in Clause 8.

Generally speaking, the rules below apply.

a) A class is generally mapped onto an XML element.

b) Attributes of a class are mapped onto XML attributes of the XML element if the <<attr>> stereotype is defined for the UML attribute, or onto a XML sub element if no stereotype is defined. If an UML attribute with stereotype <<content>> is defined, the attribute is simply mapped onto the content of the XML element.

c) Classes connected to another class via a composition or aggregation relationship are mapped onto sub elements. In cases where the target cardinality of the contained class is multiple, a wrapper element around the sub elements is generated. The wrapper element is omitted if the {nowrapper}-constraint is defined on the composition or aggregation relationship respectively.

d) Associations to other classes are mapped onto XML references to other elements. Three kinds of reference mechanisms have been defined for ODX: these are marked through stereotypes <<snref>>, <<snpathref>> and <<odxlink>>, respectively. For the detailed implementation of these reference mechanisms, see 7.3.13.

e) In ODX, classes that are part of an inheritance hierarchy are mapped in XML, depending on which UML inheritance relationship stereotype is used. The two inheritance stereotypes used in ODX are <<impl-child>> and <<impl-parent>>. Both stereotypes represent specializations, as defined in 6.2.2. When specifying an inheritance relationship using the <<impl-child>> stereotype, all child or sub-types are implemented in XML; i.e. every child type will have one XML schema type. When using the <<impl-parent>> stereotype, the generated XML element carries an attribute of name xsi:type. It contains the name of the child-type that is used in the particular instance.

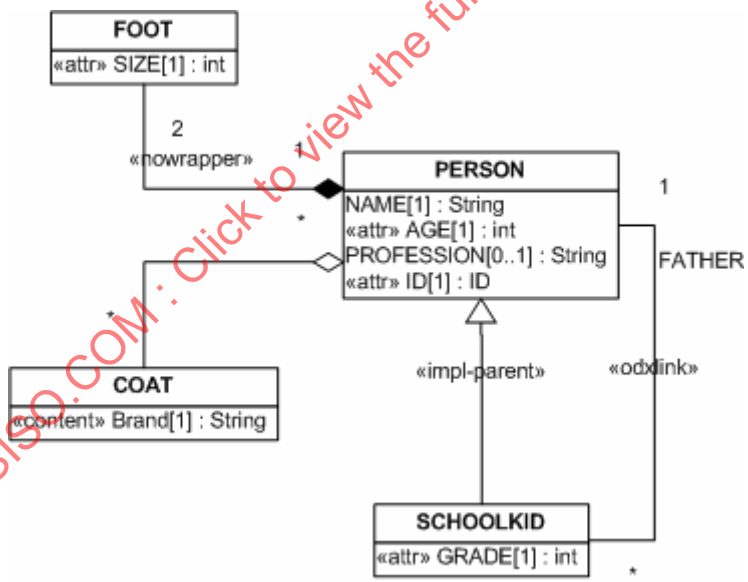


Figure 13 — UML representation of mapping example

In accordance with these rules, the example in Figure 13 is mapped onto the XML document shown below. The <EXAMPLE> root element has been included to satisfy shapeliness of the XML.

```
<EXAMPLE>
  <PERSON ID="ID_64711" AGE="33" xsi:type="PERSON">
    <NAME>Jack O. Trades</NAME>
    <PROFESSION> All Duty Service Technician </PROFESSION>
    <COAT>Gucci</COAT>
    <FOOT SIZE="8"></FOOT>
    <FOOT SIZE ="9"></FOOT>
  </PERSON>
  <PERSON ID="ID_60815" AGE = "8" xsi:type="SCHOOLKID" GRADE="3">
```

```
    <NAME>Kevin</NAME>
    <FOOT SIZE="5"></FOOT>
    <FOOT SIZE="5"></FOOT>
    <FATHER-REF ID-REF="ID_64711"></FATHER-REF>
  </PERSON>
</EXAMPLE>
```

Throughout this part of ISO 22901, some other constraints and stereotypes are used that have not been explained within this subclause. These include the following:

— <<value-inherit>> for inheritance between diagnostic layers;

— <<element-name>> to deviate from the standard naming schema that XML elements are named like their UML class counterparts;

— <<import>> to describe the import of data from another diagnostic layer;

— {ordered} to enforce that the order of XML sub elements is significant and shall not be changed by any ODX-compliant tool;

For more detailed information on the mapping to XML, see Clause 8.

7 ODX data model

7.1 General modelling principles

7.1.1 Common members

SHORT-NAME identifies an ODX object. Its length is limited to 128 characters. A short name consists of letters, digits and "_" character. The following regular expression describes the syntax of short names:

[a-zA-Z0-9_]+

LONG-NAME is a short description of the functionality of the ODX object. Its length is limited to 255 characters. The LONG-NAME should be displayed in human interface of an application instead of SHORT-NAME.

DESC describes detailed the functionality of the ODX object and has no length restriction. This element is optional and may involve several paragraphs marked by the ordered list of element <p>. The following tags known from HTML can be used to format the text:
, <i>, <b>, <u>, <sub>, <sup>, <ul>, <ol>, <li>.

LONG-NAME and DESC may also have an optional member TI (text identifier) that supports multilingualism for different kinds of text module. The mapping technology is tool-/manufacturer-specific. For example, they may occur in an external file. The TI attribute then allows the mapping between external and internal descriptions and can be used for language translations.

ELEMENT-ID is used as a common type to represent the above listed members throughout the ODX data model. An attribute HANDLE of this type is used at all elements using these common members.

ID is an identifier used by the linking concept "odxlink" described later in this part of ISO 22901. The value of ID is restricted by XML specification. No ODX compliant tool shall change the content of the ID over the whole life cycle of the data element. No system that provides import/export facilities of ODX may change existing IDs during import/modify/export cycles without explicit command by a user. Every ID needs to be unique in one coherent ODX data pool (project). The process owner may explicitly decide to change IDs and use a tool to perform the changes, i.e. the tool can assign a new ID to a new object. The process owner decides upon the method used to ensure uniqueness of IDs, so as to avoid the necessity of subsequent changes due to

conflicts. However, the use of Universally Unique Identifiers (UUIDs) as described in ISO/IEC 11578 is recommended.

OID is used for invariant identification of an object, but not for linking. Any ODX compliant tool shall not change the content of the Object-ID (if present) over the whole life cycle of the data element. Any system that provides import/export facilities of ODX into an internal format shall ensure the OIDs are maintained during an import/modify/export cycle. The process owner is responsible for ensuring the uniqueness of the OID throughout the overall process chain. The use of Universally Unique Identifiers (UUIDs) as described in ISO/IEC 11578 is recommended.

7.1.2 Common objects

7.1.2.1 Special data group

Special data groups (SDGs) are the standard extension mechanism of ODX, and are used to store any kind of data that is not covered by the standardized part of the data model in a structured way. Examples for the use of SDGs in ODX are the company-specific documentation in COMPANY-DOC-INFO. The ODX specification defines the structure of SDGs, but not its content; thus a standard diagnostic tool conforming to ODX is not required to process SDGs.

Figure 14 — UML representation of special data group (SDG) illustrates the modelling in UML.

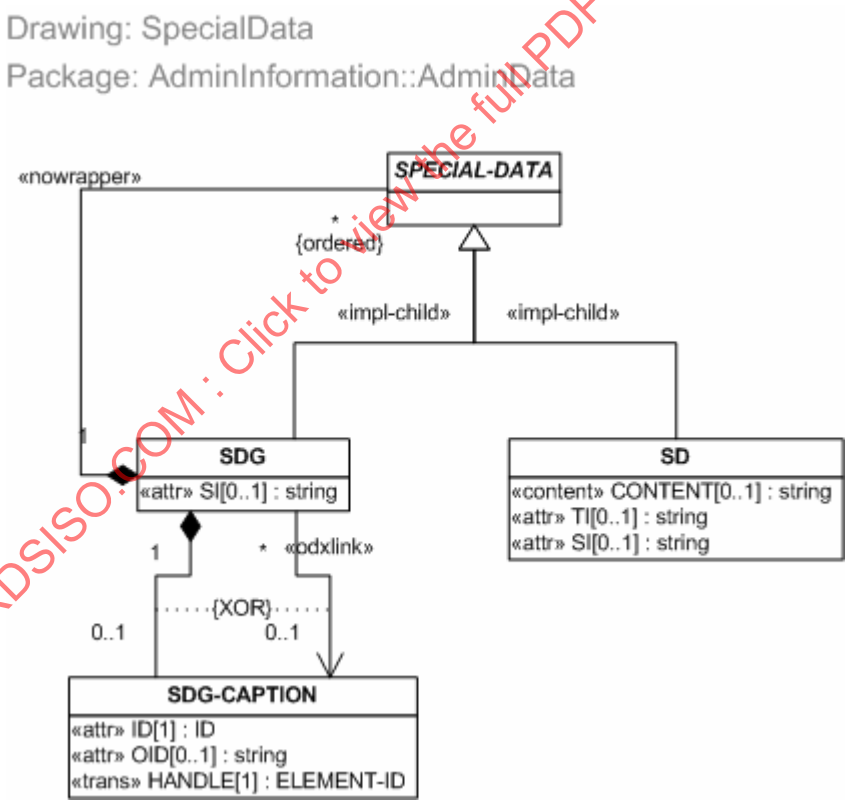


Figure 14 — UML representation of special data group (SDG)

An SDG contains an optional SDG-CAPTION to describe the SDG content, and a list of SDG and SD objects that contain the special data. This list can contain an arbitrary number of SDGs and SDs, and the ordering of these SDGs and SDs is not restricted in any way. SDGs can be nested recursively; that way, very complex data structures may be defined as SDGs. The member SI is used to add semantic information to the appropriate object, e.g. it can be used to implement a table with key-value pairs (see example 1 below). The reuse of an already defined SDG-CAPTION can be done with the SDG-CAPTION-REF, which results from the odxlink between SDG and SDG-CAPTION.

EXAMPLE 1 Definition of tables of property lists with SDGs:

```
<SDGS>
  <SDG>
    <SDG-CAPTION ID = "id123">
      <SHORT-NAME>properties</SHORT-NAME>
         <LONG-NAME>A table of key-value pairs</LONG-NAME>
      </SDG-CAPTION>
      <SD SI = "part-number">4711</SD>
      <SD SI = "index">007</SD>
      <SD SI = "SAP-number">1234567</SD>
  </SDG>
</SDGS>
```

EXAMPLE 2 Describing deep substructures with SDGs (fictitious example).

With SDGs, it is possible to create complex structures of any depth, e.g. Figure 15 — Special data group shows a number of tables, which can be combined to build a 3-dimensional matrix.

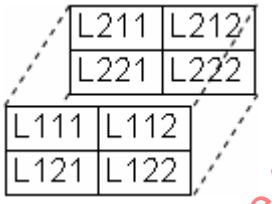


Figure 15 — Special data group

This matrix can be described by an SDG structure of depth 3:

```
<SDGS>
    <SDG>
        <SDG-CAPTION ID="imatrix">
            <SHORT-NAME>matrix</SHORT-NAME>
            <LONG-NAME>A 3-dimensional matrix mapped to an SDG structure</LONG-NAME>
        </SDG-CAPTION>
        <SDG>
            <SDG-CAPTION ID="itable1">
                <SHORT-NAME>firstTable</SHORT-NAME>
            </SDG-CAPTION>
            <SDG>
                <SDG-CAPTION ID="i1row1">
                    <SHORT-NAME>firstRow</SHORT-NAME>
                </SDG-CAPTION>
                <SD SI="example">L111</SD>
                <SD SI="example">L112</SD>
            </SDG>
            <SDG>
                <SDG-CAPTION ID="i1row2">
                    <SHORT-NAME>secondRow</SHORT-NAME>
                </SDG-CAPTION>
                <SD SI="example">L121</SD>
                <SD SI="example">L122</SD>
            </SDG>
        </SDG>
        <SDG>
            <SDG-CAPTION ID="itable2">
```

```
            <SHORT-NAME>secondTable</SHORT-NAME>
        </SDG-CAPTION>
        <SDG>
            <SDG-CAPTION ID="i2row1">
                <SHORT-NAME>firstRow</SHORT-NAME>
            </SDG-CAPTION>
            <SD SI="example">L211</SD>
            <SD SI="example">L212</SD>
        </SDG>
        <SDG>
            <SDG-CAPTION ID="i2row2">
                <SHORT-NAME>secondRow</SHORT-NAME>
            </SDG-CAPTION>
            <SD SI="example">L221</SD>
            <SD SI="example">L222</SD>
        </SDG>
    </SDG>
</SDG>
</SDGS>
```

7.1.2.2 AUDIENCE and ADDITIONAL-AUDIENCE

Audiences define the target group or groups of users for the following elements:

— DIAG-COMM;

— MULTIPLE-ECU-JOB;

— CONFIG-ITEM, CONFIG-RECORD and DATA-RECORD of ECU-CONFIG;

— BASE-FUNCTION-NODE of FUNCTION-DICTIONARY;

— SESSION-DESC of ECU-MEM-CONNECTOR;

— DATABLOCK of ECU-MEM;

— TABLE-ROW.

Five groups are predefined in ODX and thus five values can be specified there:

a) IS-SUPPLIER;

b) IS-DEVELOPMENT;

c) IS-MANUFACTURING;

d) IS-AFTERSALES;

e) IS-AFTERMARKET.

Each value can be set to “true” or “false”. By default, the elements potentially carrying an audience are available for every group, if the audience element is empty or not present.

To expand the list of enabled or disabled groups, the sub element ADDITIONAL-AUDIENCE of the DIAG-LAYER can be used. ADDITIONAL-AUDIENCE is an individual list of users that shall be enabled or disabled to read the corresponding diagnostic elements. A diagnostic element may contain either enabling or disabling references to ADDITIONAL-AUDIENCEs.

Figure 16 — UML representation of layer common – additional audience shows the detailed modelling in UML.

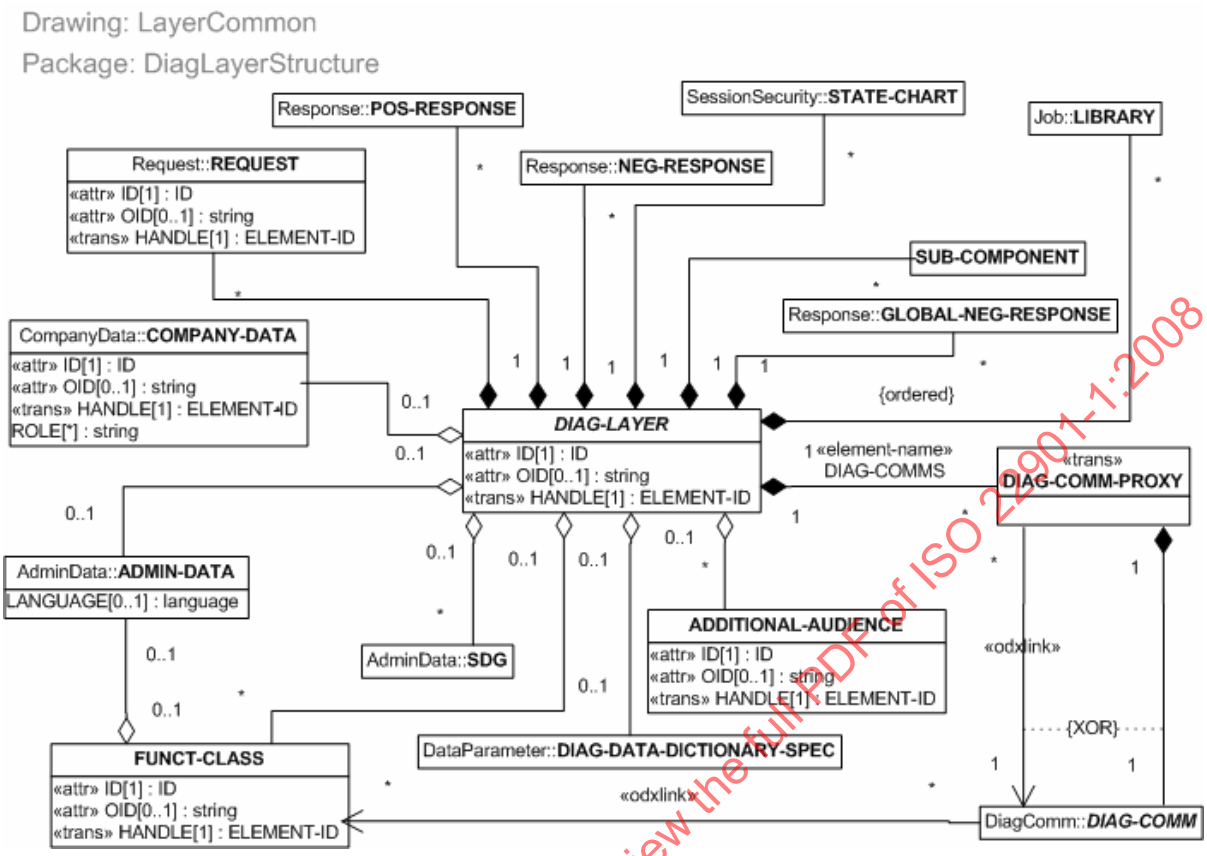


Figure 16 — UML representation of layer common – additional audience

EXAMPLE ADDITONAL-AUDIENCES:

```
<ADDITIONAL-AUDIENCES>
    <ADDITIONAL-AUDIENCE ID="IS-SUPPLIER_ID1">
        <SHORT-NAME>IS_SUPPLIER</SHORT-NAME>
        <LONG-NAME>IS-SUPPLIER</LONG-NAME>
    </ADDITIONAL-AUDIENCE>
</ADDITIONAL-AUDIENCES>
```

The “ENABLED-AUDIENCE-REF” has the meaning that this element e.g. DIAG-COMM is only available for the referenced ADDITIONAL-AUDIENCEs.

THE “DISABLED-AUDIENCE-REF” has the semantics that this element is not available for the referenced ADDITIONAL-AUDIENCEs, but available for all other listed ADDITIONAL-AUDIENCEs.

In addition to the reference ADDITIONAL-AUDIENCEs, the predefined AUDIENCEs shall be evaluated by a tool.

See Figure 17 — UML representation of DIAG-COMM and ADDITONAL-AUDIENCE for the detailed modelling in UML.

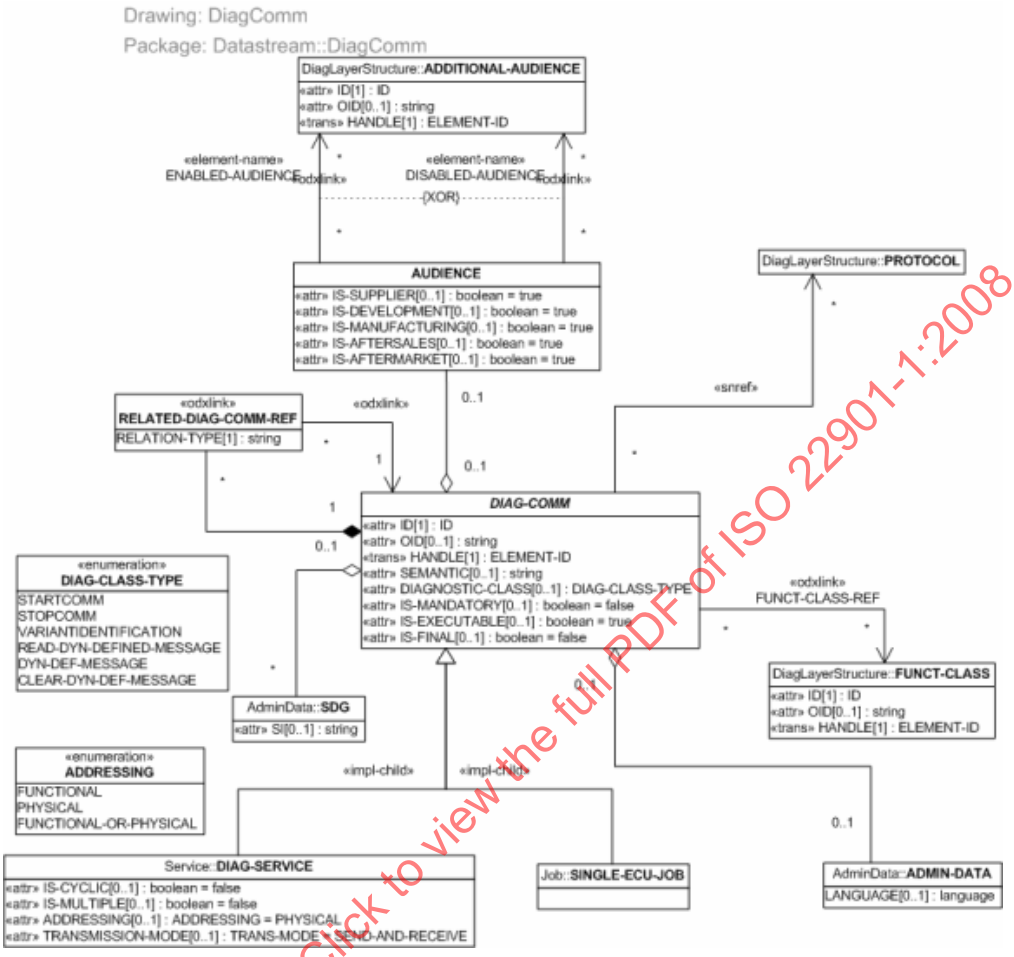


Figure 17 — UML representation of DIAG-COMM and ADDITONAL-AUDIENCE

EXAMPLE ADDITIONAL-AUDIENCES:

```
<!-- Definition of ADDITIONAL-AUDIENCES -->
<ADDITIONAL-AUDIENCES>
  <ADDITIONAL-AUDIENCE ID="MANUFACTURER_C.ID">
    <SHORT-NAME>MANUFACTURER_C</SHORT-NAME>
    <LONG-NAME>Manufacturer_C</LONG-NAME>
  </ADDITIONAL-AUDIENCE >
  <ADDITIONAL-AUDIENCE ID=„SUPPLIER_G.ID">
    <SHORT-NAME>SUPPLIER_G</SHORT-NAME>
    <LONG-NAME>SUPPLIER_G</LONG-NAME>
  </ADDITIONAL-AUDIENCE >
  <ADDITIONAL-AUDIENCE ID=„MANUFACTURER_S.ID">
    <SHORT-NAME>MANUFACTURER_S</SHORT-NAME>
    <LONG-NAME>Manufacturer_S</LONG-NAME>
  </ADDITIONAL-AUDIENCE >
  <ADDITIONAL-AUDIENCE ID="SUPPLIER_B.ID">
    <SHORT-NAME>SUPPLIER_B</SHORT-NAME>
    <LONG-NAME>SUPPLIER_B</LONG-NAME>
  </ADDITIONAL-AUDIENCE >
```

```
</ADDITIONAL-AUDIENCES >

<DIAG-SERVICE ID="SERIVCE1.ID">
  <SHORT-NAME>SERVICE_1</SHORT-NAME>
  <AUDIENCE IS-AFTERMARKET="false">
    <ENABLED-AUDIENCE-REFS>
        <ENABLED-AUDIENCE-REF ID-REF="MANUFACTURER_C.ID"/>
    </ENABLED-AUDIENCE-REFS>
  </AUDIENCE>
</DIAG-SERVICE>

<DIAG-SERVICE ID="SERIVCE2.ID">
  <SHORT-NAME>SERVICE_2</SHORT-NAME>
  <AUDIENCE>
    <DISABLED-AUDIENCE-REFS>
        <DISABLED-AUDIENCE-REF ID-REF="SUPPLIER_B.ID"/>
    </DISABLED-AUDIENCE-REFS>
 </AUDIENCE>
</DIAG-SERVICE>
```

The result is as follows:

— Service_1 is enabled for MANUFACTURER_C but disabled for all other defined ADDITIONAL-AUDIENCE.

— Service_2 is disabled for SUPPLIER_B but enabled for all other defined ADDITIONAL-AUDIENCE.

7.1.2.3 Administrative information

7.1.2.3.1 Overview

Some ODX objects like diagnostic layers or services have a very long lifetime. They may be subject to many changes, possibly performed by different people from different companies. The change history of such an object should be stored with the object. Some information about the person who is responsible for the change should be available, too; so that questions and feedback can be addressed to the right recipient. This kind of data is called “administrative data” (or short “admin data”).

Each process partner can also add his own company-specific admin data to an ODX object, e.g. revision information that is needed to handle the ODX object with a content management system or a database.

See Figure 18 — UML representation of admin data for detailed modelling information in UML.

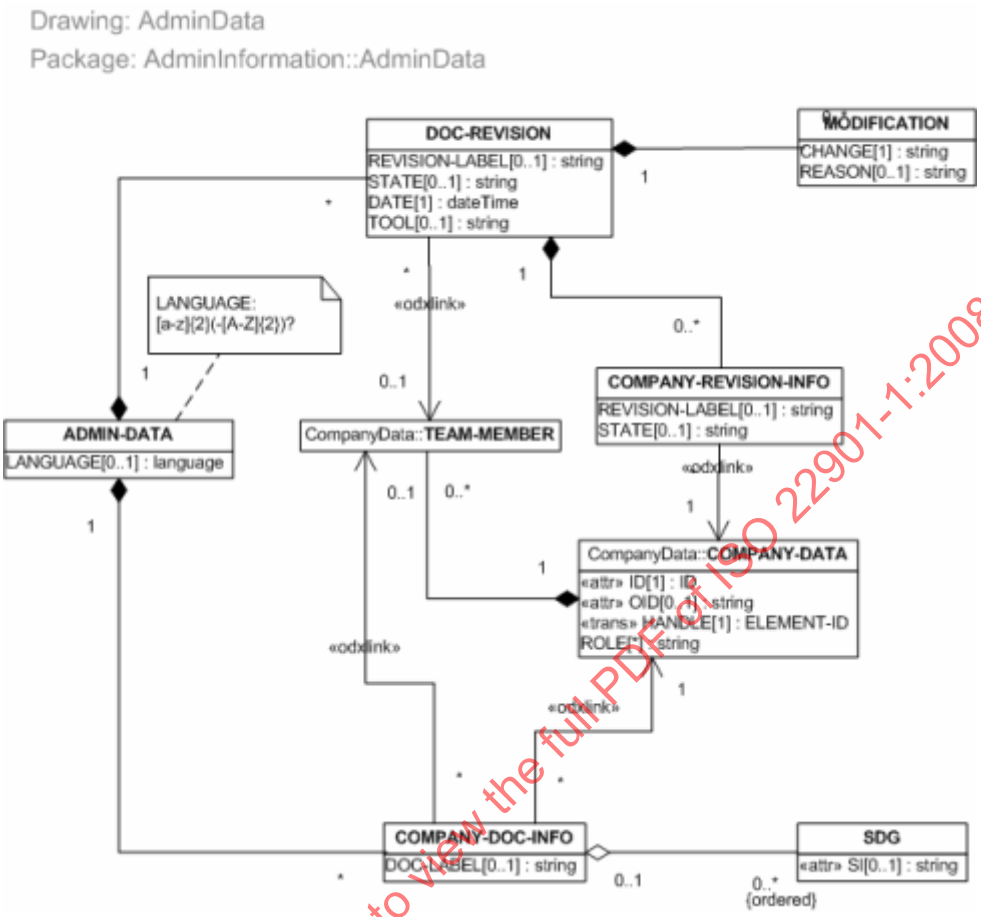


Figure 18 — UML representation of admin data

7.1.2.3.2 Location in ODX data model

ODX provides two classes of objects for admin data: ADMIN-DATA and COMPANY-DATA.

An ADMIN-DATA object contains the admin data that is specific to the ODX object it belongs to. Information about the companies that have created or modified the ODX object is stored in COMPANY-DATA objects. There should be a separate COMPANY-DATA object for each company that is involved as a process partner in the creation or modification of ODX objects. See Figure 19 — Admin info for UML modelling information.

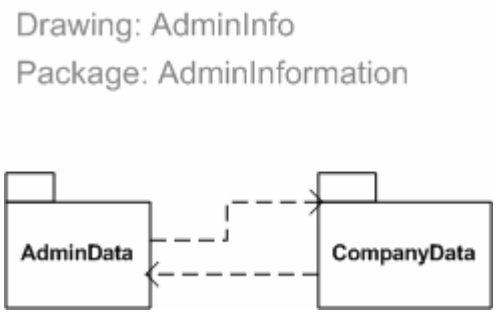


Figure 19 — Admin info

ADMIN-DATA objects are associated to a number of ODX objects. These ODX objects are typically expected to be handled (e.g. modified or exchanged) as a whole in the engineering process.

Each COMPANY-DATA object provides information about a company and its employees who participate in the engineering process. All company-specific admin data refers directly or indirectly to such a COMPANY-DATA object. With these references, a process partner can filter out his own company-specific admin data.

COMPANY-DATA is placed in those ODX objects, e.g. DIAG-LAYER-CONTAINER, that are used as data exchange units and are possibly transferred to other process partners.

An ODX container catalogue described in Clause 9 Packaged ODX data may also contain ADMIN-DATA and COMPANY-DATA objects. Admin data from an ODX container catalogue describes revisions of the files in an ODX container; it is typically used in the data exchange process.

7.1.2.3.3 Structure of ADMIN-DATA

ADMIN-DATA consists mainly of two parts: DOC-REVISIONS and COMPANY-DOC-INFOS.

The natural LANGUAGE of the text data of the whole ODX object containing ADMIN-DATA can be specified if necessary, e.g. to support cooperation with international process partners. It is a four-letter cod, e.g. LANGUAGE may be set to “de-DE” for German or “de-CH” for German, Swiss variant.

Each DOC-REVISION object describes one revision of the object to which the ADMIN-DATA belongs. A DOC-REVISION object shall contain the DATE of the revision. The person that created the revision can be referenced with a TEAM-MEMBER-REF. As an option, the REVISION-LABEL, the STATE of the revision, the used TOOL, and the MODIFICATIONS made can be added. The process partners shall agree on the use of these objects, e.g. all process partners have to apply the same revisioning scheme if they want to use the REVISION-LABEL. For each MODIFICATION, the CHANGE that was made shall be described. A REASON for the change may be added.

A DATE is formatted according to the syntax of the dateTime datatype from the XML schema specification that is based on ISO 8601. A syntactically correct DATE has the form CCYY-MM-DDThh:mm:ss, where CC is the century, YY the year of the century, MM the month of the year, DD the day of the month, T the separator between date and time, hh the hours of the day, mm the minutes and ss the seconds (e.g. 2003-08-22T15:40:25). Such a date is interpreted as being UTC. A time zone may as well be explicitly specified overriding this default.

In addition, company-specific revision information can be stored in COMPANY-REVISION-INFO objects. This way, each process partner can attach revision information required for his individual version control process to the ODX object.

The COMPANY-DOC-INFO objects contain company-specific information about the object the ADMIN-DATA belongs to. They are mainly used for documentation purposes.

Each COMPANY-DOC-INFO object contains a reference to the COMPANY that owns it. An optional reference to the TEAM-MEMBER that is responsible for this part of the document may be added, as well as a DOC-LABEL that can be used for identification of the COMPANY-DOC-INFO object. Arbitrarily structured content (e.g. a table that is to be included in a reference manual) can be stored in special data groups (SDGs). SDGs are described in further detail in 7.1.2.1.

See Figure 20 — UML representation of company data for detailed modelling information in UML.

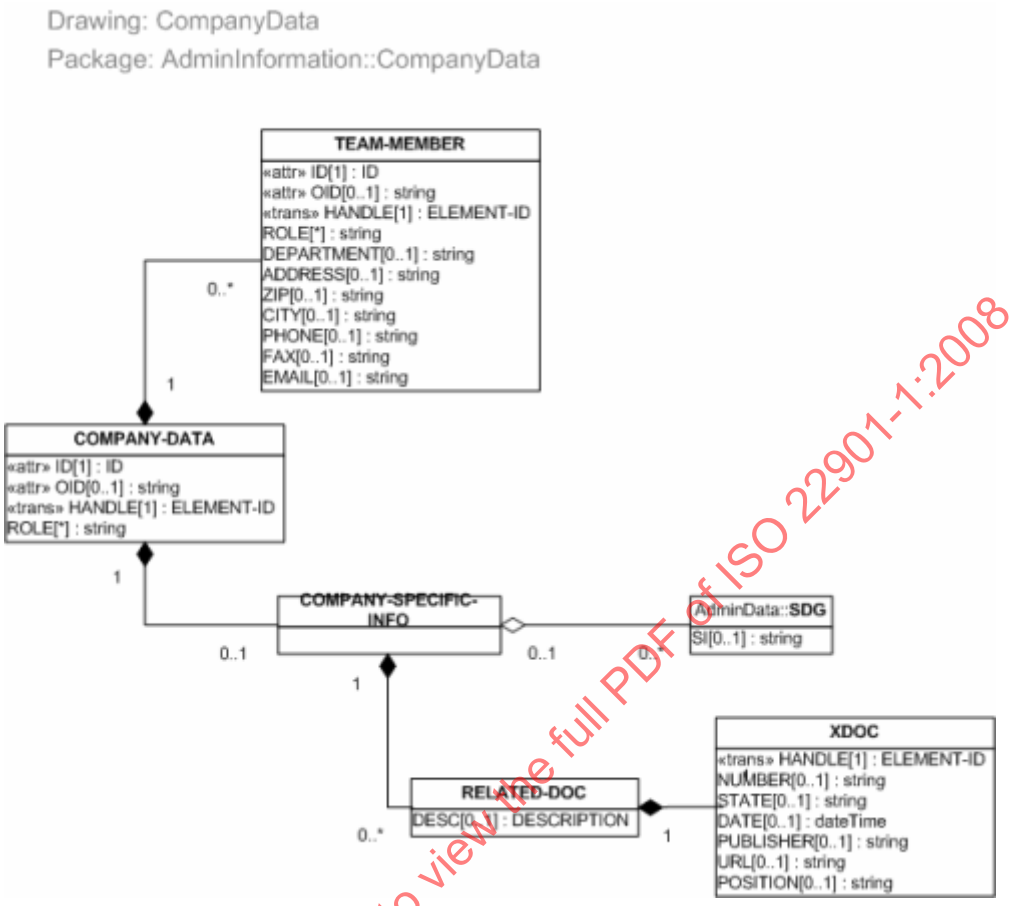


Figure 20 — UML representation of company data

EXAMPLE Admin data:

```
<ADMIN-DATA>
    <LANGUAGE>en-UK</LANGUAGE>
    <COMPANY-DOC-INFOS>
        <COMPANY-DOC-INFO>
            <COMPANY-DATA-REF ID-REF="SampleCompany"/>
        </COMPANY-DOC-INFO>
    </COMPANY-DOC-INFOS>
    <DOC-REVISIONS>
        <DOC-REVISION>
            <TEAM-MEMBER-REF ID-REF="JDo"/>
            <REVISION-LABEL>0.1</REVISION-LABEL>
            <STATE>INITIAL</STATE>
            <DATE>2008-02-25T15:00:00</DATE>
            <TOOL>XMLSPY</TOOL>
            <MODIFICATIONS>
                <MODIFICATION>
                    <CHANGE>Initial. Addition of Example Services</CHANGE>
                    <REASON>ODX Documentation</REASON>
                </MODIFICATION>
            </MODIFICATIONS>
        </DOC-REVISION>
        <DOC-REVISION>
```

```
            <TEAM-MEMBER-REF ID-REF="JSm"/>
            <REVISION-LABEL>0.2</REVISION-LABEL>
            <STATE>REVIEW</STATE>
            <DATE>2003-11-06T15:00:00</DATE>
            <MODIFICATIONS>
                <MODIFICATION>
                    <CHANGE>Review/Changes caused by Review</CHANGE>
                </MODIFICATION>
            </MODIFICATIONS>
        </DOC-REVISION>
    </DOC-REVISIONS>
</ADMIN-DATA>
```

7.1.2.3.4 Structure of COMPANY-DATA

For each employee a TEAM-MEMBER object is stored. Detailed contact information may be added with optional DEPARTMENT, ADDRESS, ZIP code, CITY, PHONE, FAX, EMAIL and ROLE objects.

Information that is not associated with a single person may be stored in the COMPANY-SPECIFIC-INFO object. RELATED-DOC objects are used for cross-referencing additional documents. For this purpose, an XDOC object is specified that may contain the optional objects NUMBER, STATE, DATE, PUBLISHER, URL, and POSITION. A DATE is formatted according to the syntax of the dateTime data type from the XML schema specification that is based on ISO 8601.

Any other company-specific data can be stored in special data groups (SDGs).

EXAMPLE Company data:

```
<COMPANY-DATA ID = "SampleCompany">
  <SHORT-NAME>SampleCompany</SHORT-NAME>
  <LONG-NAME>Sample Company</LONG-NAME>
  <TEAM-MEMBERS>
    <TEAM-MEMBER ID = "JDo">
       <SHORT-NAME>JDo</SHORT-NAME>
       <LONG-NAME>John Doe</LONG-NAME>
       <DEPARTMENT>Development</DEPARTMENT>
       <ADDRESS>5th Avenue</ADDRESS>
       <ZIP>55555</ZIP>
       <CITY>Big Apple</CITY>
       <PHONE>+1 555 1234 567</PHONE>
       <EMAIL>John.Doe@sample.com</EMAIL>
     </TEAM-MEMBER>
     <TEAM-MEMBER ID = "JSm">
       <SHORT-NAME>JSm</SHORT-NAME>
       <LONG-NAME>John Smith</LONG-NAME>
       <DEPARTMENT>Service</DEPARTMENT>
       <ADDRESS>6th Avenue</ADDRESS>
       <ZIP>55555</ZIP>
       <CITY>Big Apple</CITY>
       <PHONE>+1 555 1234 678</PHONE>
       <EMAIL>John.Smith@sample.com</EMAIL>
    </TEAM-MEMBER>
  </TEAM-MEMBERS>
</COMPANY-DATA>
```

COMPANY-DATA and TEAM-MEMBER may be referenced by ADMIN-DATA. Therefore, a unique SHORT-NAME shall be chosen for any of these objects to enable tools to offer a selection of employees and companies.

7.1.2.4 Usage of ADMIN-DATA

7.1.2.4.1 Revision information of ODX objects

The ADMIN-DATA of an ODX object may contain revision information specific to this object. Generally speaking, we distinguish two types of revision information:

— public revision information can be used by all process partners;

— internal revision information is provided for company-internal use only.

ADMIN-DATA supports the storage of both types of revision information.

When a new revision of an ODX object is created, this may be documented by adding a new DOC-REVISION to the ADMIN-DATA of the object. The list of all DOC-REVISIONs of an ODX object represents the revision history for this object.

The mandatory DATE of the DOC-REVISION denotes the creation date of this revision.

A DOC-REVISION contains both public and internal revision information. Public revision information may include a REVISION-LABEL to identify the revision, the STATE of the revision, the TOOL used to create the revision, the MODIFICATIONs made since the last revision, and the TEAM-MEMBER who created the revision. To make use of the DOC-REVISION's REVISION-LABEL, the process partners have to agree on a common versioning scheme. The interpretation of STATE, TOOL, and MODIFICATIONs is also process-specific. Therefore, it is not covered by the ODX specification in further detail.

Each process partner can add internal revision information to a DOC-REVISION with a COMPANY-REVISION-INFO. The mandatory COMPANY-DATA-REF is used to identify the COMPANY-REVISION-INFO's owner, i.e. the process partner who added the internal revision information. This allows the process partners to store their internal revision information without interfering with each other. As an option, a REVISION-LABEL and the STATE of the revision may be added. Such an internal REVISION-LABEL can be provided, e.g. by a configuration management system that applies a proprietary versioning scheme.

Any COMPANY-REVISION-INFO is only relevant to its owner and should be left untouched by the other process partners.

NOTE As ODX is not supposed to define guidelines for version control and configuration management, a D-server is not required to process any revision information associated with ODX objects. Nevertheless, revision information of ODX objects can be used by specialized tools, e.g. for looking up a specific revision of an ODX object referenced by another ODX object: the REVISION attribute of the odxlink contains the (process-specific) description of which revision of the link target is requested.

7.1.2.4.2 Updating the revision history of an ODX object

If a new revision of an ODX object is created, the revision history of any ODX object that contains the changed object shall be updated (i.e. a new revision of the containing object is added to its revision history), e.g. a new revision of a DIAG-SERVICE result in a new revision of the diagnostic layer instance to which the service instance belongs.

The DATE of the new DOC-REVISION of the containing ODX object should be set to a reasonable default value, e.g. the most recent DATE found in the DOC-REVISIONs of the contained ODX objects. The manner in which the REVISION-LABEL and other revision information are set is process-specific.

7.1.2.4.3 Versioning conflicts

When more than one process partner provide a new revision of the same ODX object at the same time, this can result in a versioning conflict. To avoid versioning conflicts, the process partners have to negotiate who is allowed to change an ODX object and its associated ADMIN-DATA. In this way, at most one new revision of each ODX object is available at the same time.

7.1.3 Value coding

Values of ASAM types are present in ODX documents, like for CODED-CONST or PHYS-CONST values. The schema type of these values is string, as they have to store content of a type that is not known while parsing the XML file. They shall be converted from string to the ASAM type, as if they were declared in the schema with the XML schema corresponding type.

Table 1 — ASAM types correspondence with XML schema types contains the correspondence between ASAM types and XML schema types.

In general, values of type number in the ODX data shall be decimal coded if no other number coding is defined by model or specification.

Table 1 — ASAM types correspondence with XML schema types

ASAM type	XML schema type	Example values
A_INT32	int	“12” or “-37”
A_UINT32	unsignedInt	“12”
A_BYTEFIELD	hexBinary	“AB05” or “ab05”
A_FLOAT32	float	“12.0” or “1.2e+1”
A_FLOAT64	double	“12.0” or “1.2e+1”
A_ASCIISTRING	string	“on” or “off”
A_UTF8STRING	string	“on” or “off”
A_UNICODE2STRING	string	“on” or “off”

NOTE 1 While the XML document can contain character references, they are already resolved by the XML parser in the string value, e.g. A is already substituted with “A”.

NOTE 2 The string representation of a byte field of zero length is the empty string.

7.2 ODX package

7.2.1 Overview

Figure 21 — ODX package overview provides for easier understanding the UML model of ODX. This is divided into packages, which mostly reflect the logical structure of the model and to improve readability. The packages and their relations are also shown in Figure 21. From the viewpoint of the data model, ODXStructure is a kind of root package.

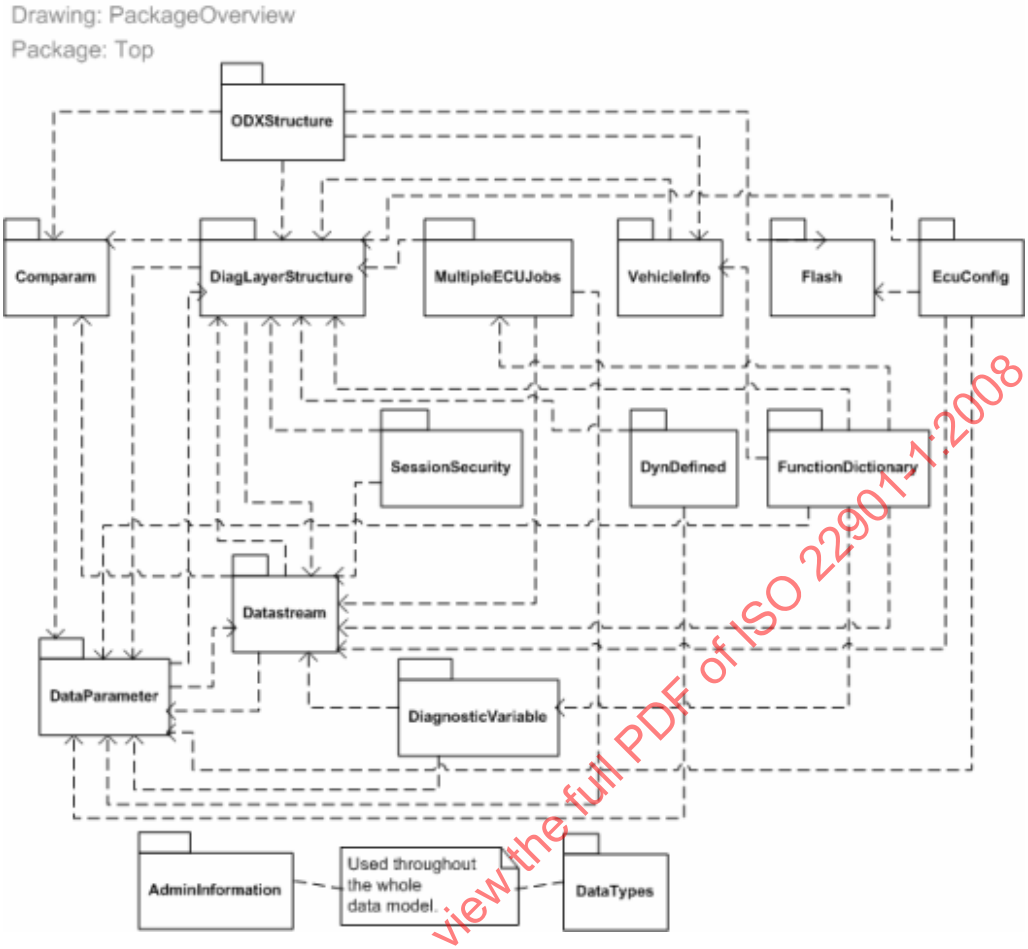


Figure 21 — ODX package overview

A short overview of the technical contents of the packages is given below.

— **DiagLayerStructure, Comparam, MultipleECUJobSpec, VehicleInfo, Flash**: See 7.2.2.

— **SessionSecurity**: This package describes a data model for a state diagram, allowing specifying an ECU's diagnostic sessions, security states and corresponding methods to switch diagnostic sessions and to perform a security access.

— **DataParameter**: This package deals with the description of simple (scalar) and complex (structured) data objects in the diagnostic data stream and with the conversion to symbolic values (name, value, unit). The important concept of DOP (data object property) is also defined in this package.

— **DataStream**: This package deals with the description of the communication between diagnostic tester and ECU on "bits and bytes" i.e. diagnostic message or signal level. Diagnostic services and diagnostic jobs are defined as well as request and response telegrams (from a tester's point of view).

— **DynDefined**: With some protocols (e.g. ISO 14230 or ISO 14229-1) it is possible to dynamically define data records at run time, which contain values from other (already existing) records. This package contains information used by services to define and to read the new data records.

— **DiagnosticVariable**: This package describes an optional, alternative access to the ODX data, which is not communication oriented but rather quantity oriented. In the first case, the diagnostic application (or user) requires the tester to communicate with the ECU using a specific service (e.g. "read by local ID 44"). In the second case the user does not care about communication details at all but rather requests a specific variable from the ECU (e.g. "read the current value of Temperature T2")

— **AdminInformation**: This package contains commonly used structures used to support the development process of diagnostic data like versioning, reference to content management systems and support of multi company projects.

— **DataTypes**: This package contains the data types used in other packages and serves as a reference only.

7.2.2 Package ODXStructure

Figure 22 — UML representation of ODX structure contains the “ODXStructure” package from the very top-level root data of an ODX data instance.

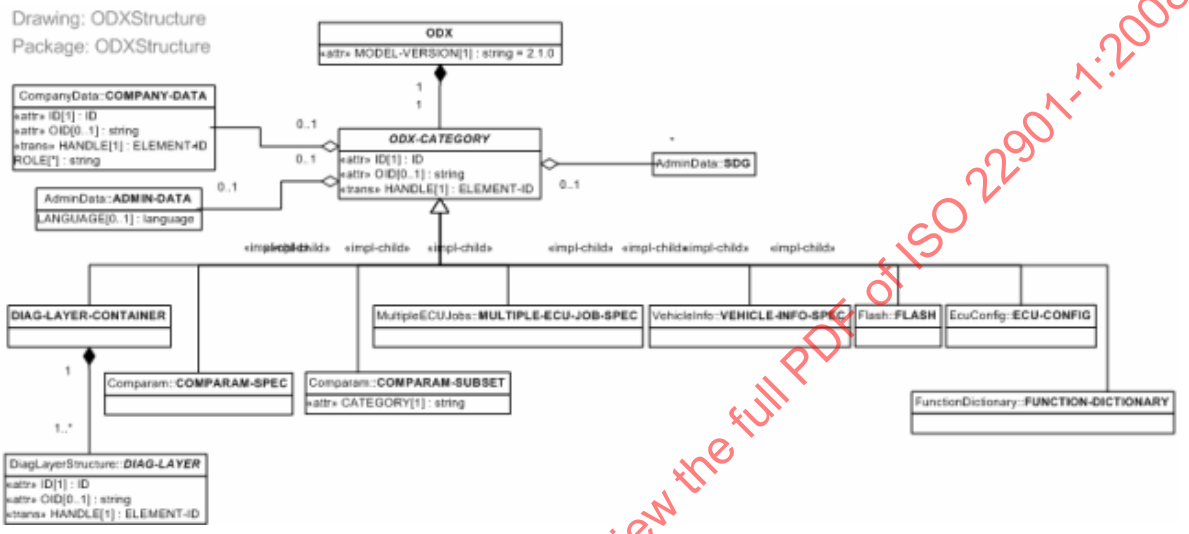


Figure 22 — UML representation of ODX structure

The “ODXStructure” package then splits into one of the ODX-CATEGORYs below.

a) A DIAG-LAYER-CONTAINER contains one or a set of DIAG-LAYER structures, i.e. logical layers for the description of diagnostic services with all necessary data, and additional process-specific information like ADMIN-DATA and COMPANY-DATA. The process partner who receives the DIAG-LAYER-CONTAINER can extract the DIAG-LAYERs to store them individually; alternatively, the complete DIAG-LAYER-CONTAINER can be handled as a single storage unit. (Package “DiagLayerStructure”.)

b) A COMPARAM-SPEC defines a specific protocol stack consisting of predefined communication parameter sets (COMPARAM-SUBSETs - specification of communication parameters, e.g. timings for physical layer, transport layer and diagnostic protocol). The communication parameters and those values are predefined in these COMPARAM-SUBSETs. The value of communication parameters may be changed in context of a layer, specific diagnostic service, LOGICAL-LINK or PHYSICAL-VEHICLE-LINK. This part of the model can also contain OEM specific parameters (Package “Comparam”).

c) The definition of diagnostic jobs that deal with quasi-parallel communication with several ECUs (MULTIPLE-ECU-JOBS defined in the package “MultipleECUJobs”).

d) Information about the access to a specific ECU in a vehicle network using gateways and using the concept of logical links between tester and a specific (physical) ECU (VEHICLE-INFO-SPEC defined in package VehicleInfo). More about vehicle information can be found in 7.3.10.

e) The description of data which is needed for ECU flash memory programming like memory layout, logical structure of flash fragments and information (references) about the diagnostic services or jobs that are to be used to flash the data (FLASH defined in package “Flash”).

f) An ECU's behaviour is strongly depending on the explicit configuration of a certain car. To reduce the amount of ECU-VARIANTS, ECU Configuration is performed. The ECU-CONFIG describes several use cases and the corresponding data model features.

g) Many functions of a car are distributed across several ECUs. The section FUNCTION-DICTIONARY describes the features of the ODX data model to cover the use case of function-oriented diagnostics.

These data often descend from different sources and have their own and sometimes independent lifecycle. The information of DIAG-LAYER structures for example can be generated by the ECU supplier who on the other hand probably will not have complete information about the vehicle network where the ECU is to be integrated. This information is available at the OEM, which illustrates that for process reasons this data should be kept in separate parts. This on the other hand does not mean that objects in the ODX files are not linked at all. For the use of the flash data for example the diagnostic jobs (or services) can be found in a DIAG-LAYER structure and are strongly linked to the information in the FLASH package.

An ODX instance may contain only one of the ODX-CATEGORYs.

7.3 ODX data model for diagnostics

7.3.1 Overview

The ODX standard is intended to contain all information that is required by a diagnostic tester to do diagnostic communication with a specific ECU or set of ECUs. This means that all ECU specific parameters for communication are completely contained in ODX and no ECU specific software in the tester software is necessary. This makes the tester completely data driven: when testing new ECUs no software shall be updated, just the appropriate ODX data shall be loaded.

The data description is designed to be independent of communication protocols, although ISO 14230 was often used to check the concepts of ODX data description practically. In order to reduce protocol specific data in the ECU description, a special place for the definition of the set of protocol specific communication parameters was created (see 7.3.3 and 7.3.4).

Due to its intention to describe the diagnostic communication between tester and ECU, the understanding of the data model part of the DiagLayerContainer naturally starts with looking at the description of the diagnostic communication services (DIAG-SERVICE). That means that a tester application that uses the ODX XML data will find ODX data – in a reduced view - to be a set of diagnostic services (i.e. the description of messages or frames on the communication bus). This corresponds to the concept of the object oriented interface of the D-server API where objects diagnostic service and job play a key role.

This does not mean that the DIAG-SERVICE is the only “entry point” of the ODX data. With respect to the use case of measurement and data acquisition the concepts of DIAG-VARIABLES (see 7.3.7) and TABLE (see 7.3.6.11) are an alternative way to use the ODX data. This means that you can look up the ODX XML data for a required measurement variable without caring about the underlying communication services that are used to obtain the current value of an internal variable of the ECU.

It is recommended to read the details of the DIAG-LAYER data model starting from the DIAG-SERVICE class. Speaking not of classes but of objects, if you pick out a DIAG-SERVICE object you get the root of a tree of related objects (aggregation or reference) that are used to describe the parameters that are necessary for a D-server to do the diagnostic communication. In general, objects not linked in this chain (or tree) with DIAG-SERVICE being the root object are - from the view of a D-server - not of relevance for communication, e.g. if a POS-RESPONSE (description of a communication telegram from the ECU) is defined which is not used (referenced) by any DIAG-SERVICE, it will never become effective, i.e. it cannot be used for communication. Exceptions for these general rules are, for example, COMPARAM-REF and GLOBAL-NEG-RESPONSE. However, other elements like TABLE or DIAG-VARIABLE can be used as an initial access to the data through a D-server. See Figure 23 — Object chain in ODX DiagLayerStructure.

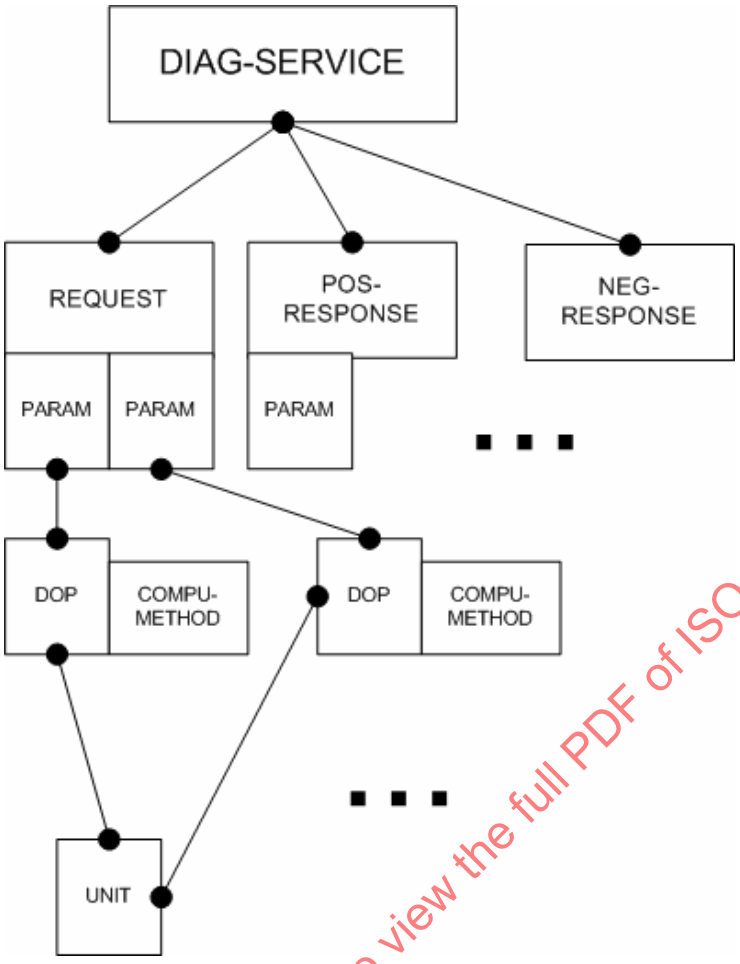


Figure 23 — Object chain in ODX DiagLayerStructure

In all cases where reuse of objects is possible, these objects are connected to other objects via references ("links" in XML). Since these links sometimes shall be created interactively with an editor tool, normally all objects that can be reused by reference do have a characteristic set of attributes: SHORT-NAME (unique identifier for this specific class of object), LONG-NAME and, DESC, and ID.

Only a set of DIAG-SERVICEs is visible from the viewpoint of an application using D-server to access the ODX data - slightly simplified. As it can be seen later, this set of DIAG-SERVICEs depends on the "location" that is used in the ODX data model.

EXAMPLE The number of supported services depends on the variant of an ECU.

If you take, for example, the ODX description of the "EU" (Europe) variant of an ECU, you may get a different set of DIAG-SERVICEs compared to another variant such as "JP" (Japan). It is of course possible to have only one set of DIAG-SERVICEs in an ODX file, but for the benefit of avoiding redundant data, it is possible to spread the DIAG-SERVICEs over several hierarchical layers of the data model, as shown in 7.3.2. The layers are top down: PROTOCOL, FUNCTIONAL-GROUP, BASE-VARIANT, ECU-VARIANT. ECU-SHARED-DATA is a special pool of information, as described later in 7.3.2.

7.3.2 Diagnostic layer structure

7.3.2.1 Overview

The ODX data model structures the diagnostic data in five so-called diagnostic layers, which pursue the following purposes:

a) implement a hierarchical model to achieve data abstraction across a set of ECU variations, protocols, functional groups and libraries in order to limit data redundancy: this special (limited) form of inheritance is called value inheritance in this part of ISO 22901, and it includes the possibility to override certain objects that are contained in a more general layer by another object defined in a more specialized layer;

b) provide a library-like mechanism via ECU-SHARED-DATA;

c) create a framework that supports ECU variant identification and base variant identification;

d) reflect the needs of the D-server by defining the set of objects that are visible at the D-server: this visibility is connected with the concept of value inheritance and only available for the instances of the classes which are the subject of value inheritance (see 7.3.2.4);

e) define the scope for the odxlink and SHORT-NAME referencing mechanisms (see 7.3.13).

Figure 24 shows a simple example of how the various diagnostic layers can be used to implement:

— a protocol (ISO 15765);

— two base variants (a body control module BCM and a door control module DCM);

— two variants of the BCM and one variant of the DCM;

— a library to collect globally defined PIDs (parameter identifiers);

— a functional group definition to allow functional communication to the complete DOORS system (functional addressing).

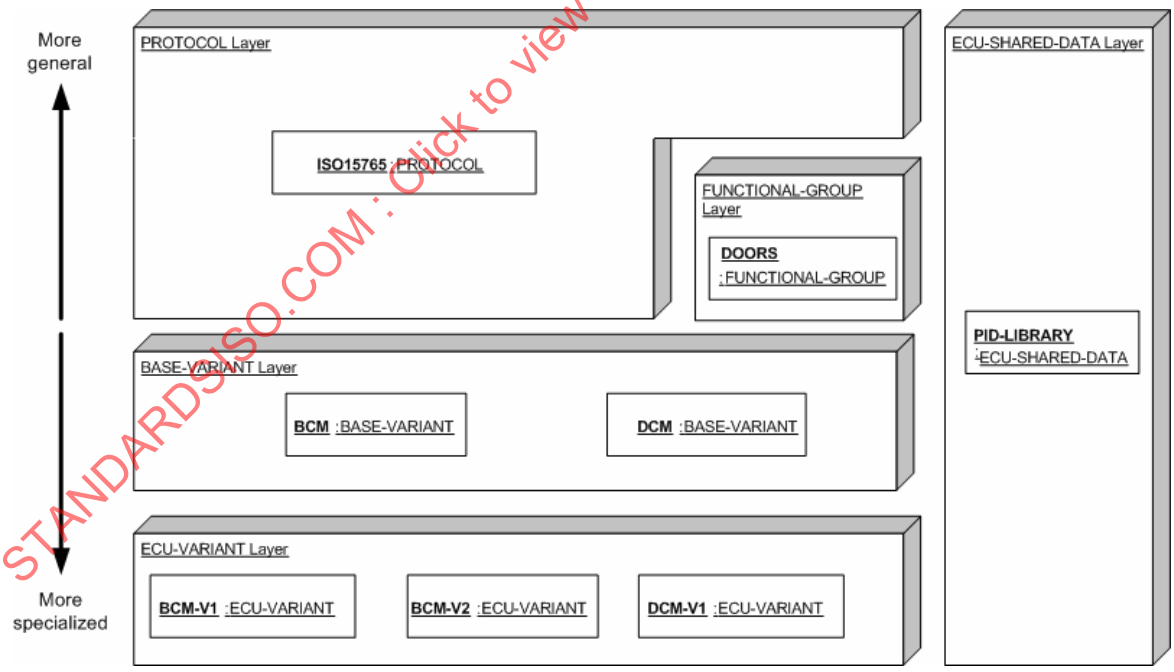


Figure 24 — Diagnostic layers overview example

7.3.2.2 Diagnostic layer modelling

The various diagnostic layers have many common characteristics that have been modelled using the common base class DIAG-LAYER. The classes PROTOCOL, FUNCTIONAL-GROUP, BASE-VARIANT and ECU-VARIANT are the building blocks of the hierarchy mentioned in 7.3.2.1. The ECU-SHARED-DATA class is somewhat different, which will be elaborated later.

See Figure 25 — UML representation of diagnostic layers for diagnostic layer modelling.

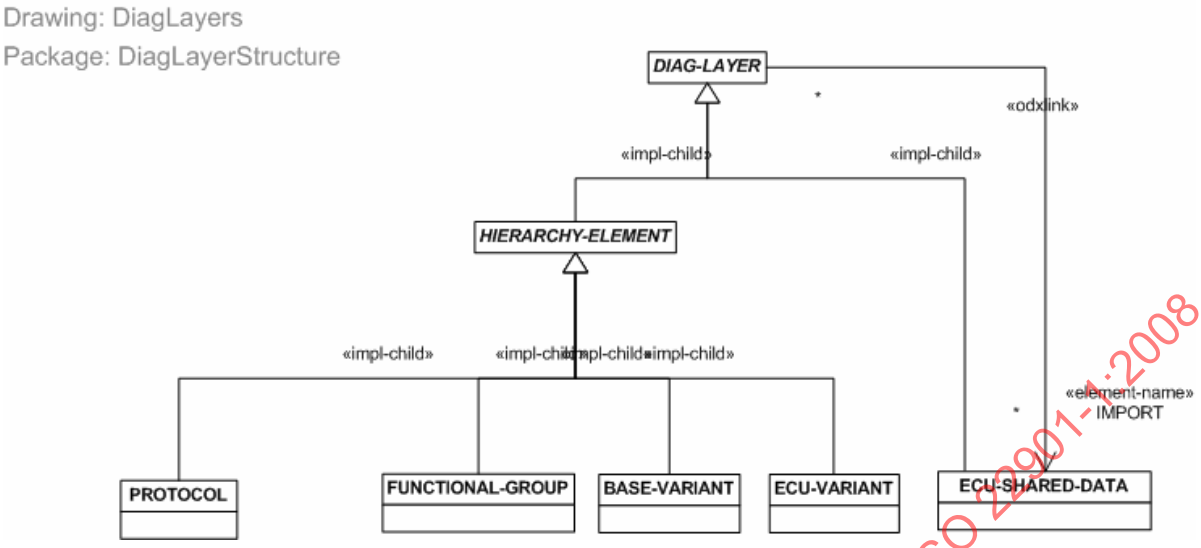


Figure 25 — UML representation of diagnostic layers

7.3.2.3 Diagnostic layer commonalities

The model section shown in Figure 26 defines the common characteristics for all diagnostic layers. The base class DIAG-LAYER aggregates the following components:

— collection of REQUEST objects (see 7.3.5.5);

— collection of response objects (POS-RESPONSE, NEG-RESPONSE, GLOBAL-NEG-RESPONSE) (see 7.3.5.6);

— document information using the ADMIN-DATA and COMPANY-DATA objects (see 7.1.2.3);

— collection of FUNCT-CLASS objects; a FUNCT-CLASS object is defined at the DIAG-LAYER and is used to categorize the DIAG-COMM objects that belong to the DIAG-LAYER; the semantics of the created categories are user specific and not defined by this part of ISO 22901;

— collection of DOP-BASE and TABLE objects: even though the element DIAG-DATA-DICTIONARY-SPEC is optional, most DIAG-LAYER instances will contain it, because it serves as the main container for the data elements necessary to decode the diagnostic messages;

— collection of DIAG-COMM objects and/or DIAG-COMM references (bundled through a DIAG-COMM-PROXY in the model – such a PROXY is not visible in an instance file); the DIAG-COMM class is an abstraction of DIAG-SERVICE and SINGLE-ECU-JOB (see 7.3.5.2), which are exposed at the D-server;

— collection of ADDITIONAL-AUDIENCEs, as described in 7.1.2.2;

— collection of STATE-CHARTs, as described in 7.3.9;

— collection of LIBRARYs, as described in 7.3.5.8;

— collection of SUB-COMPONENTs, as described in 7.9.5.

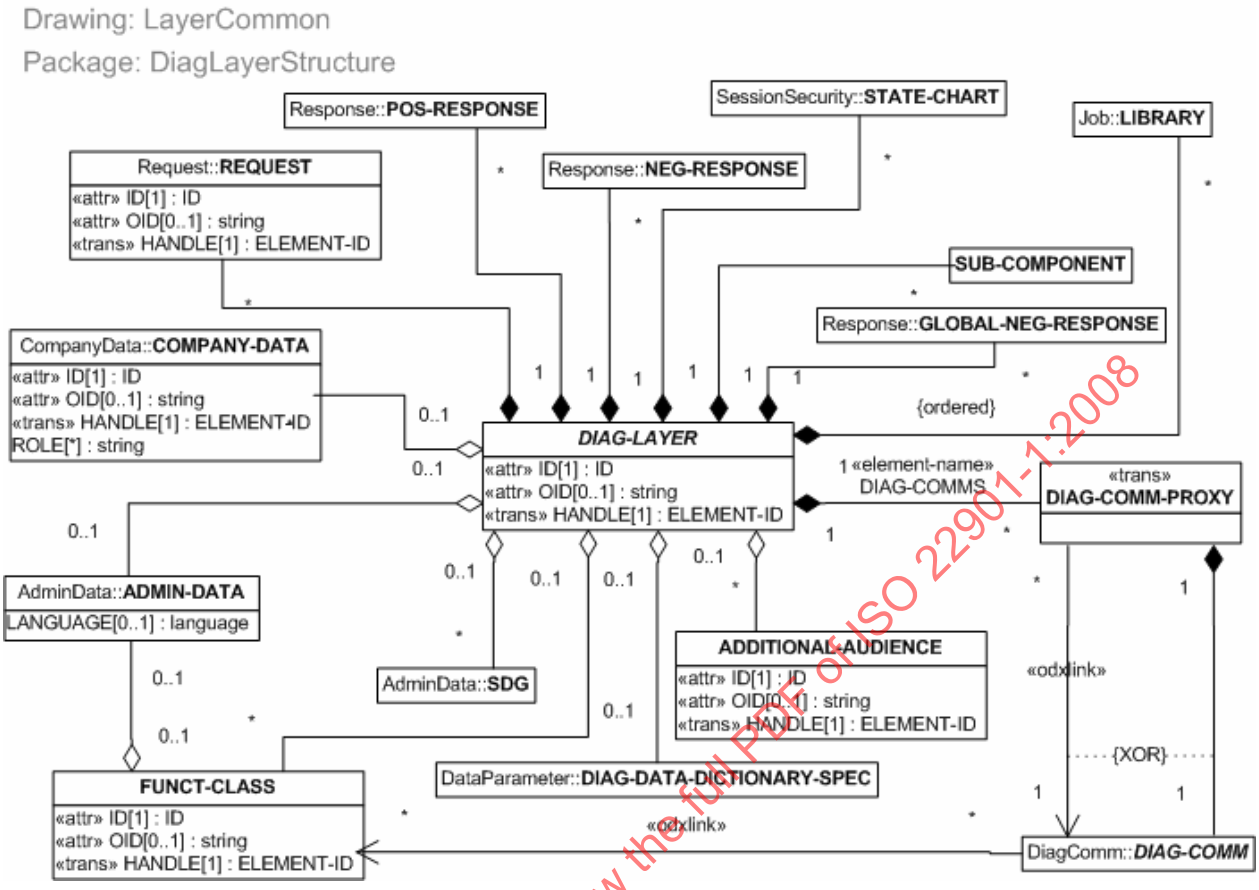


Figure 26 — UML representation of diagnostic layer commonalities

7.3.2.4 Value inheritance

7.3.2.4.1 Overview

Value inheritance is a core concept of ODX. Value inheritance makes the inheritance concept of object-oriented technology usable for diagnostic data modelling.

The benefits of value inheritance are the following:

— reuse of diagnostic data for more than one ECU or ECU-variant based on a single source principle;

— reduce the amount of data by specifying additions to and differences between ECU projects, rather than repeating data used in more than one ECU project;

— provide data security and integrity;

— avoid error-prone copies of identical data between ECU projects.

7.3.2.4.2 General

Value inheritance is a relationship between diagnostic layers, i.e. the classes PROTOCOL, FUNCTIONAL-GROUP, BASE-VARIANT, ECU-VARIANT and ECU-SHARED-DATA. Value inheritance means that data objects, which are contained within a DIAG-LAYER **A** are considered to be also contained in a DIAG-LAYER **B** that establishes a value inheritance relationship to **A**. Value inheritance is implemented by references in the inheriting layer to the inherited layer(s) (PARENT-REF).

A diagnostic layer of a specific type (PROTOCOL, FUNCTIONAL-GROUP, BASE-VARIANT, ECU-VARIANT or ECU-SHARED-DATA) can inherit only a specific set of other types of diagnostic layers, e.g. a diagnostic layer cannot inherit another diagnostic layer of the same type. In this way, an inheritance hierarchy is established between the diagnostic layer classes. Figure 27 illustrates the hierarchy and the direction of inheritance.

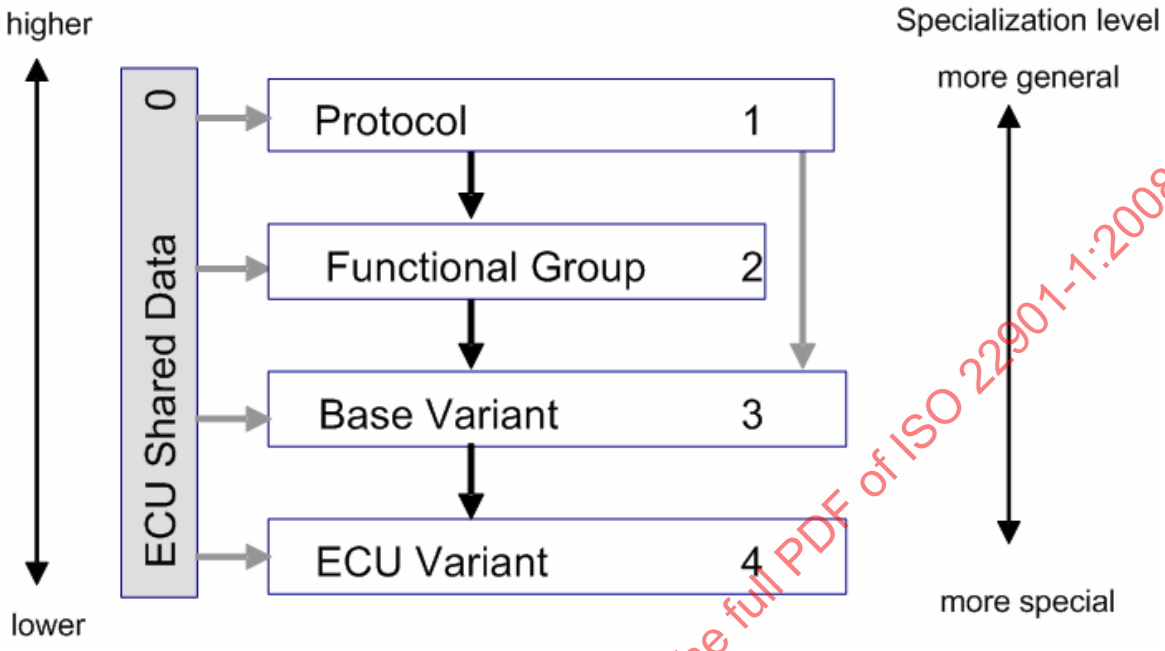


Figure 27 — Diagnostic layer hierarchy

Diagnostic layers higher in hierarchy are termed “more general”, layers lower in hierarchy are termed “more special”. Figure 28 is the UML model of the inheritance hierarchy. Table 2 — is a tabular representation of the same information.

Only objects of the following classes are subject to value inheritance:

— DIAG-COMM (its specializations);

— DIAG-VARIABLE;

— GLOBAL-NEG-RESPONSE;

— DOP-BASE (its specializations);

— TABLE;

— FUNCT-CLASS;

— VARIABLE-GROUP;

— ADDITIONAL-AUDIENCE;

— STATE-CHART;

— UNIT-GROUP.

The objects that are the subject of value inheritance have the properties described below.

a) Only objects of classes having a HANDLE attribute can be value inherited.

b) If an object having a HANDLE attribute aggregates other objects that also have HANDLE attributes, the complete aggregation is being inherited, i.e. always the outermost object of an aggregation (which has a HANDLE attribute) is subject of value inheritance.

c) Because the value inheritance is defined for objects between different instances of the DIAG-LAYER, only classes defined inside the DIAG-LAYER (and its specializations) can be value inherited.

d) The objects of these classes can be queried at the D-server API at a selected DIAG-LAYER. The result of this query will be the list of objects visible at this layer.

NOTE 1 Even if a specialization of the class DOP-BASE is not visible at the D-server API, it is nevertheless subject to value inheritance.

e) All objects of the same class shall have an unique SHORT-NAME in each DIAG-LAYER, even if the objects are not defined in this DIAG-LAYER but inherited.

f) An object which could be inherited from a more general layer can be overridden by another object of the same class with the equal SHORT-NAME in a more specialized layer. This ensures the uniqueness of the SHORT-NAME for each class. In this case, only the overriding object is visible.

g) It is possible to reference the object by only its SHORT-NAME in principle, because references by SHORT-NAME are always connected with a well-defined class and the SHORT-NAME of all objects of this class shall be unique.

NOTE 2 Some of the classes cannot be referenced by SHORT-NAME, because only a so-called «odxlink» is available in the data model, which references an object by its attribute ID (similar to regular XML references of type IDREF).

h) If an object that is subject to value inheritance is referenced by an «odxlink», it may not be visible in the current layer, e.g. it is overridden or excluded from the inheritance. Consequently, this object would not be listed in the corresponding context of the D-server, even though it shall still be used at runtime. For examples, see 7.3.2.6.

i) The inheritance of an object which is subject of value inheritance can be prevented by an explicitly NOT-INHERITED clause inside PARENT-REF. Only objects of classes DIAG-COMM, DIAG-VARIABLE, DOP-BASE, TABLE, and GLOBAL-NEG-RESPONSE can be eliminated in this way.

Some of these properties will be described in the following subclauses in more detail.

There exist other classes which are visible at the D-server API, e.g. CONFIG-DATA and SESSION-DESC. However, these objects are not defined inside the DIAG-LAYER and are therefore not subject to value inheritance. Consequently, it is not possible to override these objects.

Regardless whether an object is subject to value inheritance or not, it might be referenced by «odxlink» if it has an ID attribute. Some objects having an HANDLE attribute but are not subject to value inheritance can even be the target of a SHORT-NAME reference. Because the uniqueness is only guaranteed for value inherited classes, additional context information might be necessary to solve such a reference.

The pool of objects that can be referenced is defined by all objects inside the current DIAG-LAYER and all objects in the pools of the directly or indirectly inherited DIAG-LAYERs. This pool can be increased through the import mechanism (see 7.3.2.5). There are three exceptions: COMPARAM-REF and LINK-COMPARAM-REF, which can reference any BASE-COMPARAM, and COMPARAM-SPEC-REF, which can reference any COMPARAM-SPEC.

Value inheritance is implemented by the use of special <<odxlink>> references derived from the abstract type PARENT-REF as shown in Figure 29. Figure 28 shows the possible relationships, which can be constructed between DIAG-LAYERs using the PARENT-REF specializations.

EXAMPLE A BASE-VARIANT object establishes a value inheritance relationship to a PROTOCOL object by aggregating a PROTOCOL-REF object that in turn references the PROTOCOL object by using an <<odxlink>>.

Between a pair of DIAG-LAYER objects may exist either a value inheritance relationship as described in this subclause or an import relationship as described in 7.3.2.5. The existence of both kinds of relationships for the same pair of objects is prohibited.

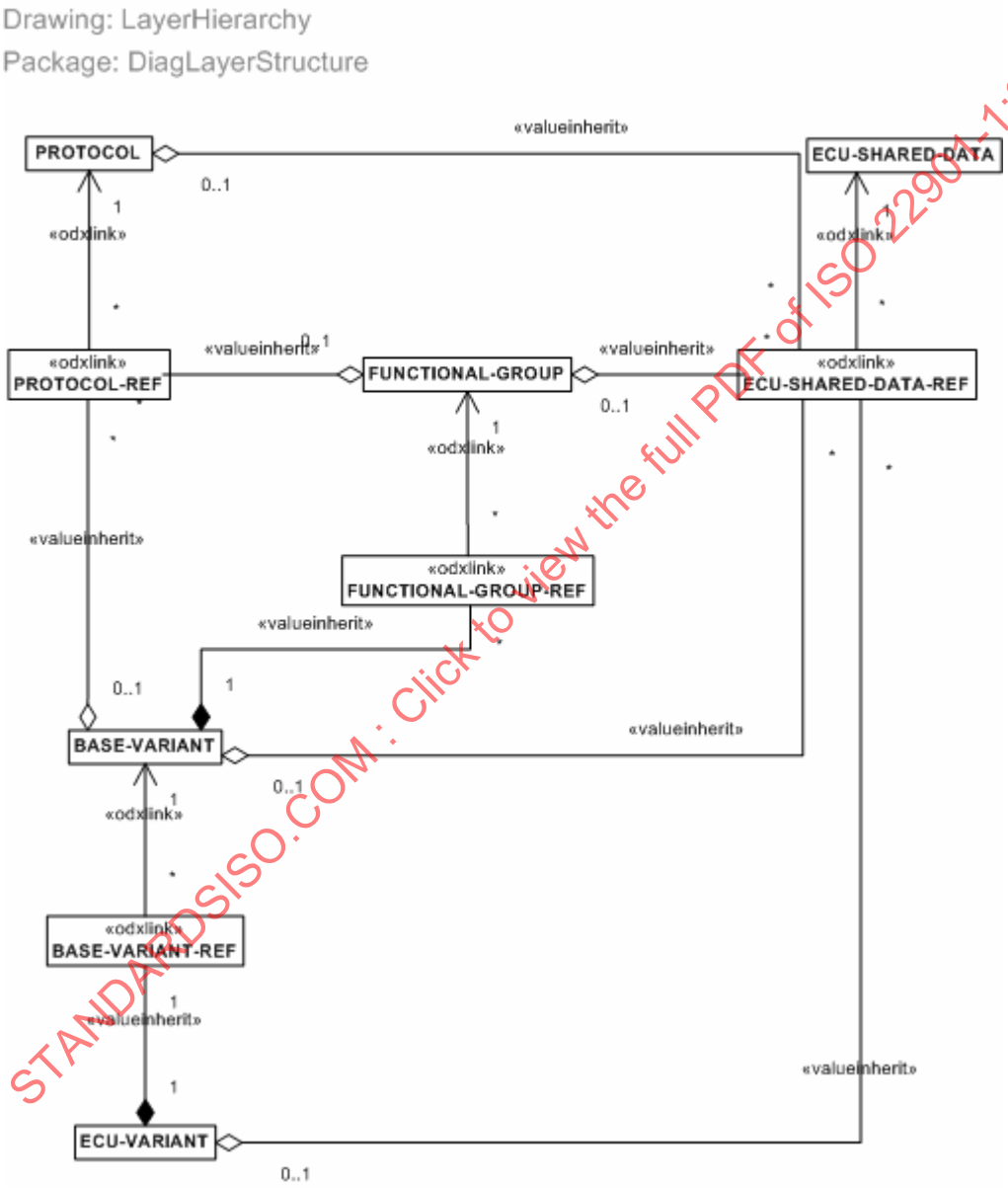


Figure 28 — UML representation of the diagnostic layer hierarchy

Table 2 — Value inheritance shows the possible value inheritance relationships among the various DIAG-LAYER specializations.

Table 2 — Value inheritance

DIAG-LAYER	May value inherit from				
	PROTOCOL	BASE-VARIANT	ECU-VARIANT	ECU-SHARED-DATA	FUNCTIONAL-GROUP
PROTOCOL	No	No	No	Multiple	No
BASE-VARIANT	Multiple	No	No	Multiple	Multiple
ECU-VARIANT	No	Exactly one	No	Multiple	No
ECU-SHARED-DATA	No	No	No	No	No
FUNCTIONAL-GROUP	Multiple	No	No	Multiple	No

7.3.2.4.3 Elimination of inherited objects

One purpose of value inheritance is to reflect the needs of the D-server by defining the set of objects that are visible at the D-server API. All of the objects that are defined within the network formed through value inheritance relationships are visible at the D-server API. To provide further control about what objects are visible, an elimination mechanism has been established by means of the NOT-INHERITED clause as shown in Figure 29 — UML representation of parent reference.

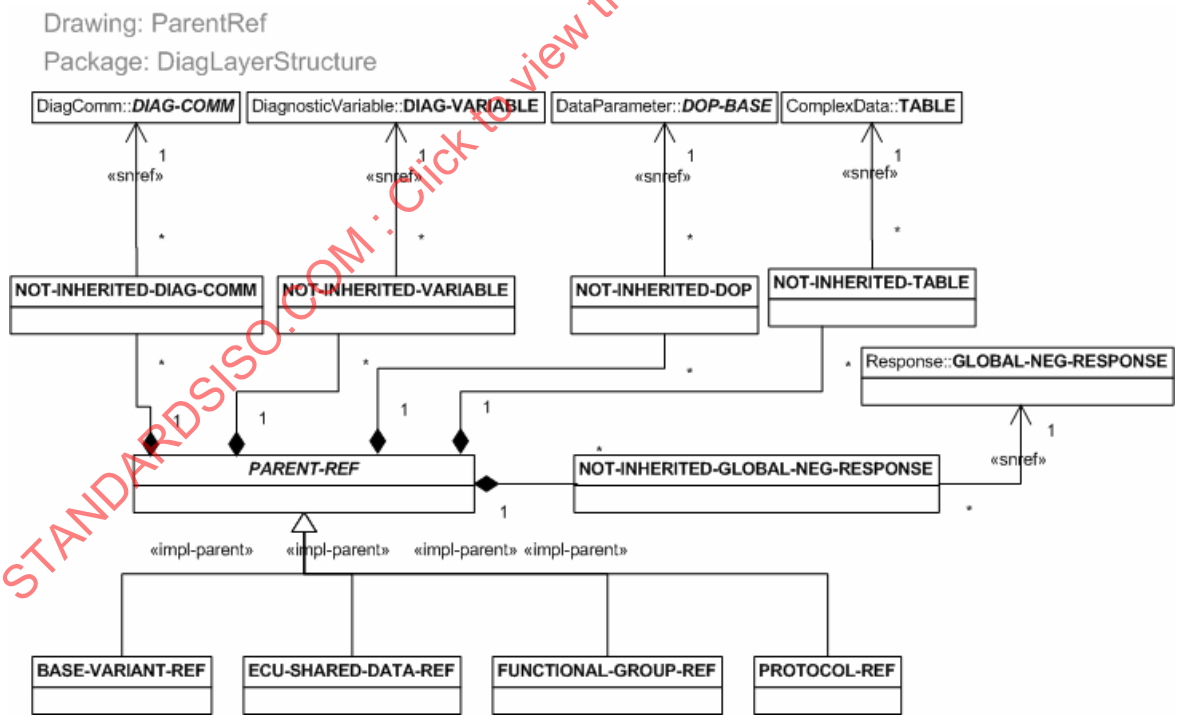


Figure 29 — UML representation of parent reference

For example, if a BASE-VARIANT does not support a certain service from a PROTOCOL it inherits, it may eliminate the service by referencing it via a NOT-INHERITED-DIAG-COMM-SNREF. All other services of the PROTOCOL are inherited normally.

7.3.2.4.4 Overriding of objects

As mentioned above, the concept of value inheritance includes the possibility to override objects that could be inherited from a more general layer. This is being accomplished by matching the SHORT-NAME of the object to be overridden. There are distinct namespaces within which the SHORT-NAME resolution takes place. The implemented class hierarchy within the ODX model implicitly gives these namespaces, i.e. all specializations of the classes which are subject of value inheritance (see 7.3.2.4.3) share the same namespace for their SHORT-NAMEs. 7.3.13 gives a comprehensive overview about the namespaces, e.g. if a DIAG-LAYER is able to value inherit a DIAG-SERVICE with the same SHORT-NAME as a SINGLE-ECU-JOB it defines locally, the locally defined SINGLE-ECU-JOB will override the DIAG-SERVICE, because SINGLE-ECU-JOB and DIAG-SERVICE belong to the same value inherited class DIAG-COMM and the own DIAG-LAYER is always more specialized than the inherited one.

Because a layer can inherit several layers, it could happen that several objects of the same class and the same SHORT-NAME are available for inheritance. The statement that the definition in the more specialized layer overrides the object in the more general layer is applied, too. For example, if a DIAG-LAYER is able to value inherit several DIAG-COMMs with the same SHORT-NAME as a DIAG-SERVICE it defines locally, the locally defined DIAG-SERVICE will override all the other services, because the own DIAG-LAYER is always more specialized than the inherited one.

For example, if a BASE-VARIANT is able to value inherit DIAG-SERVICEs with the same SHORT-NAME from a FUNCTIONAL-GROUP layer and a PROTOCOL layer, the object of the FUNCTIONAL-GROUP layer has the higher specialization level and overrides that of the PROTOCOL layer. For example, if a BASE-VARIANT is able to value inherit DIAG-SERVICEs with the same SHORT-NAME from two FUNCTIONAL-GROUP layers, both services have the same specialization level (regardless of how they became part of the FUNCTIONAL-GROUP, i.e. whether they are defined in the FUNCTIONAL-GROUP or inherited from another more general inherited layer) and an invalid constellation occurs. This forbidden multiple inheritance issue shall be solved, as described in 7.3.2.4.5.

Objects that are imported from an ECU-SHARED-DATA layer (see 7.3.13) acquire the level of specialization from the importing layer. Objects that are value inherited from an ECU-SHARED-DATA layer obtain a specialization level that is less specialized than the inheriting layer but more specialized than the next more general layer within the layer hierarchy. For example, if an ECU-VARIANT is able to value inherit DIAG-SERVICEs with the same SHORT-NAME from the BASE-VARIANT as well as from an ECU-SHARED-DATA the DIAG-SERVICE from the ECU-SHARED-DATA is supposed to override the one from the BASE-VARIANT.

Inheritance is affected by a NOT-INHERITED clause inside the PARENT-REF. Such a clause removes a specific object from a value inheritance branch. For example, if a BASE-VARIANT inherits from a FUNCTIONAL-GROUP layer and from a PROTOCOL layer and in both layers a DIAG-SERVICE with the same SHORT-NAME is visible, but a NOT-INHERITED-DIAG-COMM to this DIAG-SERVICE exists inside the FUNCTIONAL-GROUP-REF, the situation is the same as if only the PROTOCOL layer defines a DIAG-SERVICE with this SHORT-NAME. Therefore, the BASE-VARIANT can inherit the DIAG-COMM from inside the PROTOCOL layer although the PROTOCOL is more general than the FUNCTIONAL-GROUP.

NOTE 1 This rule could result in a behaviour which is unexpected for the author. To summarize the last example, before the not-inherited statement was added, the service from the FUNCTIONAL-GROUP was inherited. By adding this statement, the author could expect that no service of this SHORT-NAME is visible any longer, but in fact, the initially overridden service inside the PROTOCOL layer becomes visible again.

The overridden object from an inheriting layer is hidden by the overriding object. Therefore, only the overriding object is visible at the D-server API. Because all SHORT-NAME references (either contained in objects inherited from more general layers or defined within the current layer) are solved on the base of the objects visible in the current layer, the overriding object (i.e. the object defined in the more specialized layer) is the target. This implements the “late-binding” mechanism of ODX.

NOTE 2 Because only objects of those classes that are subject to value inheritance can be inherited, the overriding concept is only applicable to objects of these classes, too, e.g. an object of class POS-RESPONSE is not subject to value inheritance. It is possible to define an object of this type with the same SHORT-NAME, as already defined in one or several inherited layers. All of these objects exist at the same time and could be used simultaneously inside the same DIAG-LAYER. The D-server will distinguish all of them and uses that one which is referenced by an «odxlink». There exist

no SHORT-NAME references to these objects. By contrast, objects of class DOP-BASE can be referenced by SHORT-NAME to use this late-binding mechanism and, therefore, these are subject of value inheritance.

7.3.2.4.5 Multiple inheritance

Because the ODX data model allows multiple value inheritance, a diagnostic layer could inherit multiple data objects of the same namespace and the same SHORT-NAME. As described in the 7.3.2.4.3, the data objects in the highest level of specialization override any data objects in the lower levels. If more than one data object is inherited from the same highest specialization level, the situations below may occur.

— The definition of an object with the same SHORT-NAME in the same namespace in the more specialized diagnostic layer overrides all inherited definitions. This solves any ambiguity and describes a valid situation.

— All but one of these objects is eliminated by using a NOT-INHERITED section. Therefore, in the considered diagnostic layer only the remaining data object is actually inherited and no ambiguity remains. It is not permitted to eliminate a data object which is marked as mandatory[2)].

— A diagnostic layer inherits the same data object through multiple inheritance paths.

EXAMPLE Each of two PROTOCOL instances **A** and **B** establishes a value inheritance relationship to an ECU-SHARED-DATA instance **C**. All data objects contained within **C** virtually exist now within **A** and **B** at the same level of specialization. If now a BASE-VARIANT **D** establishes value inheritance relationships to **A** and **B**, it seems to get a duplicated set of **C**'s data objects. This is resolved in a way that only a single set of **C**'s data objects exists within **D**. This can be detected using the ID of data objects.

— If an ambiguity remains, the situation is forbidden.

7.3.2.5 Importing and referencing of objects

The ODX data model provides further methods to avoid redundant data by use of ECU-SHARED-DATA layers, which fulfil the purpose of libraries. An ECU-SHARED-DATA object is almost identical to the other DIAG-LAYERs with the additional property that it can also be the target of an IMPORT-REF. If a DIAG-LAYER establishes an IMPORT-REF to an ECU-SHARED-DATA object **E**, it enlarges its data pool for odxlinks by the data objects contained within **E**. In order to make use of the data objects of **E**, they shall be referenced via odxlink from an object in the referencing layer.

With regard to a specific DIAG-LAYER "**DL**", objects of the layer itself and other layers fall into one or more of the following sets, "**A**", "**I**" and "**R**":

a) "**A**": every element that can be referenced from DL via «odxlink» is in "**A**"; this includes all elements in the inheritance tree and the elements of any ECU-SHARED-DATA, that DL references via IMPORT-REF;

b) "**I**": an element from "**A**" that is target of an «odxlink» and that «odxlink» has a DIAG-COMM-PROXY, DTC-PROXY, DIAG-VARIABLE-PROXY or ROW-WRAPPER as source is in "**I**"; elements in "**I**" are called "imported"; it is not possible to import a data object by a SHORT-NAME reference only;

c) "**R**": an element from "**A**" that is target of an «odxlink» but is not in "**I**", is in "**R**".

All DIAG-LAYER specializations can establish IMPORT-REFs to multiple ECU-SHARED-DATA layers. In this way, an ECU-SHARED-DATA can reference other ECU-SHARED-DATA. Cyclic references are not allowed.

Only an ECU-SHARED-DATA can be a target of an IMPORT-REF.

2) Such an object can be eliminated, but shall then be redefined or inherited from the other parent.

Elements of "**I**" behave as if locally defined at the place of the source of the «odxlink». That is, their SHORT-NAMEs contribute to the namespace of the referencing layer. These objects are possible targets of SNREFs or SNPATH-REFs and are value inherited if their object type is subject to value inheritance. That means imported data objects can be targets of SHORT-NAME references. This does not automatically include targets of «odxlinks» from these elements, i.e. a DIAG-COMM that is part of "**I**" does not make its REQUEST become part of "**I**".

For example, if an imported DOP **E1** from an ECU-SHARED-DATA **E** references another DOP **E2** inside **E** by DOP-SNREF, the data object **E2** is not imported. Therefore, the DOP **E2** shall be defined elsewhere outside **E** as well**.**

If a DIAG-LAYER **D** imports a data object with the same SHORT-NAME as a data object that is being value inherited, the imported data object overrides the inherited data object, because it is treated as if it had been defined in **D**. Every layer which references data objects of ECU-SHARED-DATA **E** is required to establish an IMPORT-REF to **E** regardless of whether such an IMPORT-REF already exists in any of the inherited DIAG-LAYERs. If a PARENT-REF to **E** exists somewhere in the inheritance tree, any object in **E** can be referenced as described in 7.3.2.4.2.

EXAMPLE A DIAG-LAYER D references an ECU-SHARED-DATA E via IMPORT-REF. For the definition of a DIAG-SERVICE DS in D, D references the DOP D1 inside E. The DIAG-LAYER C inherits DIAG-LAYER D including service DS. So far it is not necessary that C imports E. But, if an «odxlink» inside C refers to D1 or any other data object inside E, an explicit IMPORT-REF to E is required in C. Assuming that DOP D1 from E references another DOP D2 inside E by DOP-SNREF, a DOP of the SHORT-NAME D2 is visible in the layers D and C. Because D2 inside E is neither visible in C nor D, another definition of D2 needs to make **D2** visible in both layers.

A further referencing mechanism is defined between ECU-VARIANTs. This mechanism can only be used between objects in ECU-VARIANTs inheriting from a common BASE-VARIANT. Every element of an ECU-VARIANT that can be the target of an «odxlink» can be referenced by an object from any other ECU-VARIANT inheriting from the same BASE-VARIANT. The objects of an ECU-VARIANT fall into sets analogous to the sets "**A**", "**I**" and "**R**" above with respect to each other ECU-VARIANT. However, an IMPORT-REF shall not be used for this type of referencing.

7.3.2.6 Value inheritance and referencing examples

Some examples are provided to illustrate the concept of value inheritance. A, B, C are DIAG-LAYERS and objects O, defined in a specific layer are drawn in black. Layers further down in the diagram inherit from the layer just above, indicated by an open arrow. Objects inherited are drawn in grey. Each example exhibits four columns: Column 1 identifies the layers and the objects defined in each layer. Dashed arrows are used to illustrate references (odxlinks as well as SNREFS). Column 2 shows how inheritance and/or overriding of objects will impact referencing. Column 3 lists those objects, that a D-server will return, if asked to list the objects of the respective layer. Column 4 lists all objects usable for communication by a LOGICAL-LINK on the respective layer. Therefore, column 3 depicts the visibility while column 4 depicts the usability. As mentioned above the D-server will use objects that are hidden at the API.

An object O1' represents an object of the identical type (class) and identical SHORT-NAME to object O1.

The symbols used in the examples below are shown in Figure 30 — Symbols for the value inheritance examples.

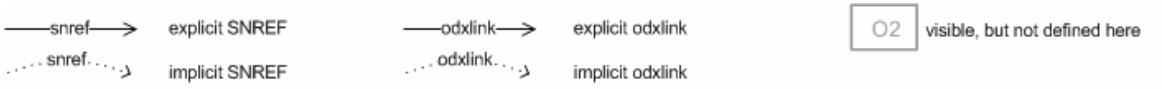


Figure 30 — Symbols for the value inheritance examples

EXAMPLE 1 The example in Figure 31 — Inheritance and odxlink referencing illustrates the case, where object O2, defined in layer A is successively overridden on both inheriting layers B and C. Yet because the object O1 references O2 in A via odxlink, O2 remains to be usable from within layers B and C. Example 2 shows a very similar situation, however, since SNREFS are used in place of the odxlinks, this impacts visibility and usability of object O2 on layers, lower in the inheritance hierarchy.

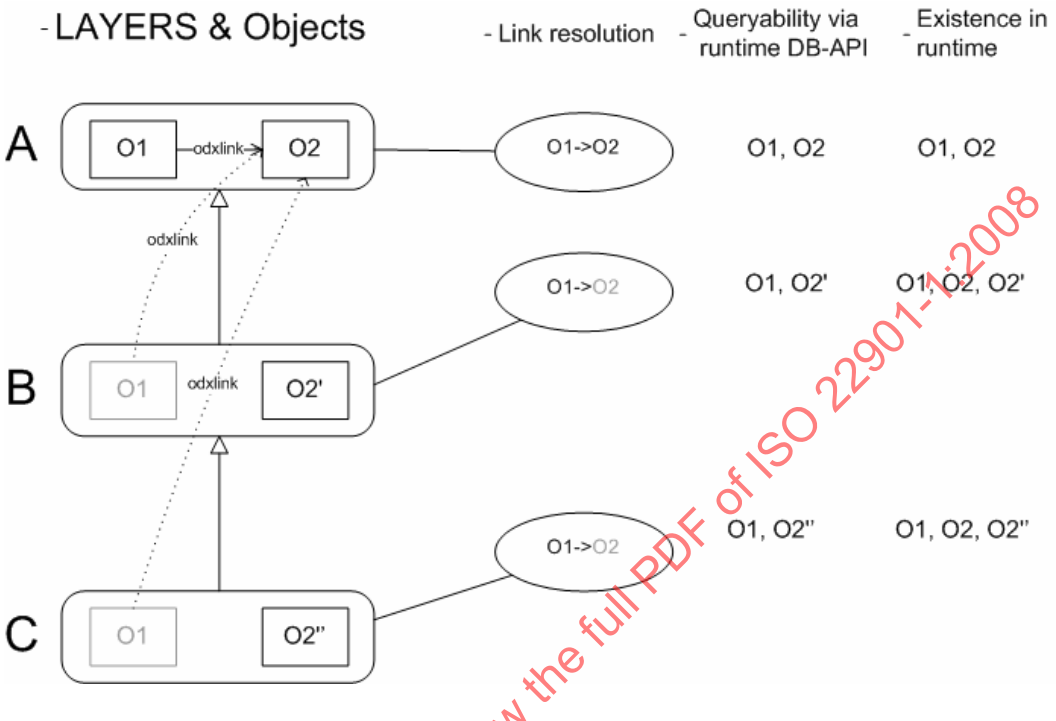


Figure 31 — Inheritance and odxlink referencing

EXAMPLE 2 Figure 32 — Inheritance and SNREFs shows references via SNREF to overridden objects O2' and O2'' respectively, rather than to the original object from higher layers, like odxlinks do. Objects overridden on lower layers are no longer usable at runtime on the respective specialization level, unless they are references via odxlink (see Example 1).

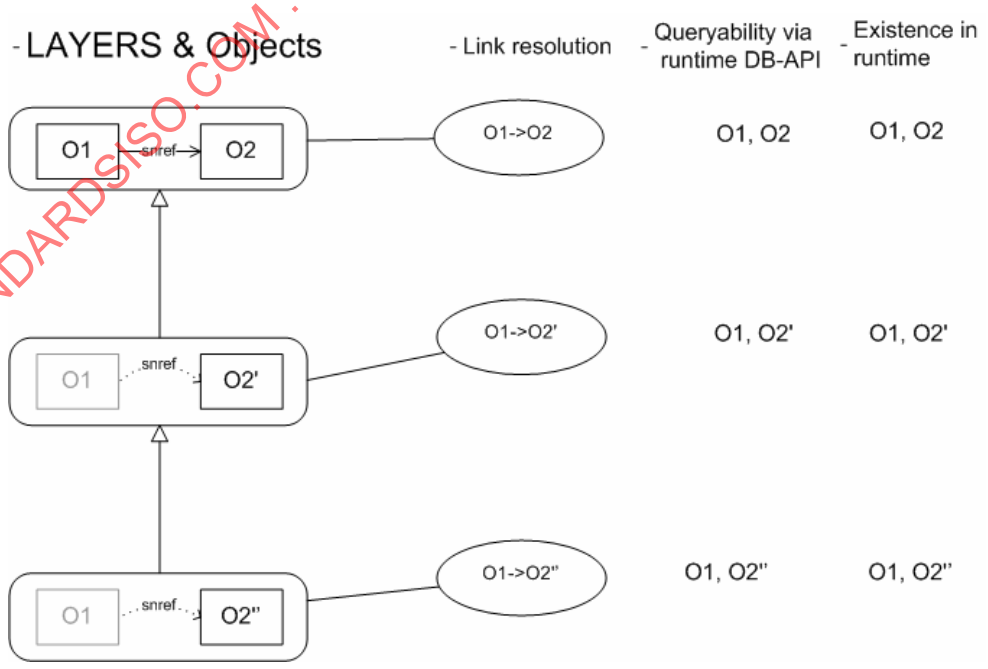


Figure 32 — Inheritance and SNREFs

EXAMPLE 3 In Figure 33 — Visibility of overridden objects the object O2 retains its usability on layers B and C even though it is overridden, however, it will only be visible as an object in layer A, not in B and C.

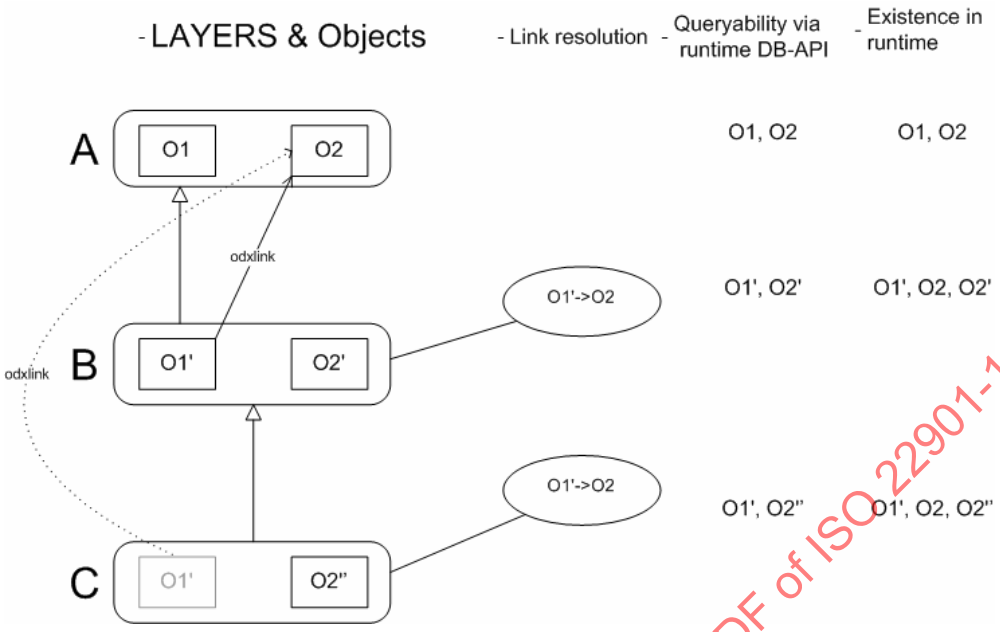


Figure 33 — Visibility of overridden objects

EXAMPLE 4 The example in Figure 34 — Overriding objects referenced by SNREFs illustrates a case where an object O2, which has SNREFs pointing to it, is overridden by O2', SNREF to O2 will point to O2' on the layer where O2' is defined and on layers lower in the inheritance hierarchy.

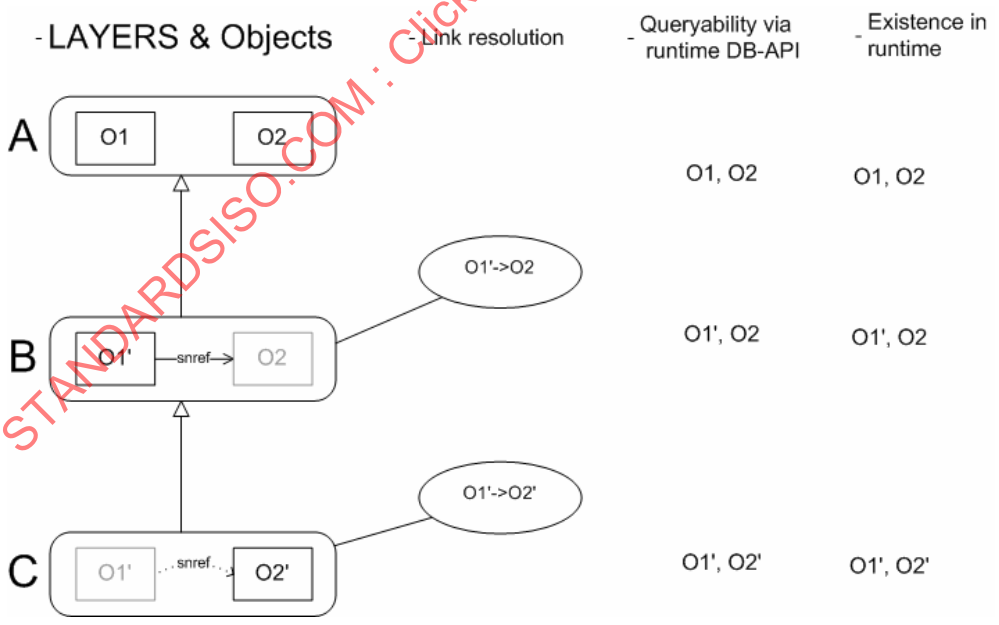


Figure 34 — Overriding objects referenced by SNREFs

EXAMPLE 5 In contrast to SNREFs, in Figure 35 — Overriding objects referenced by odxlink odxlinks preserve the identity of the objects they point to, e.g. O2, even if O2 is overridden on lower layers.

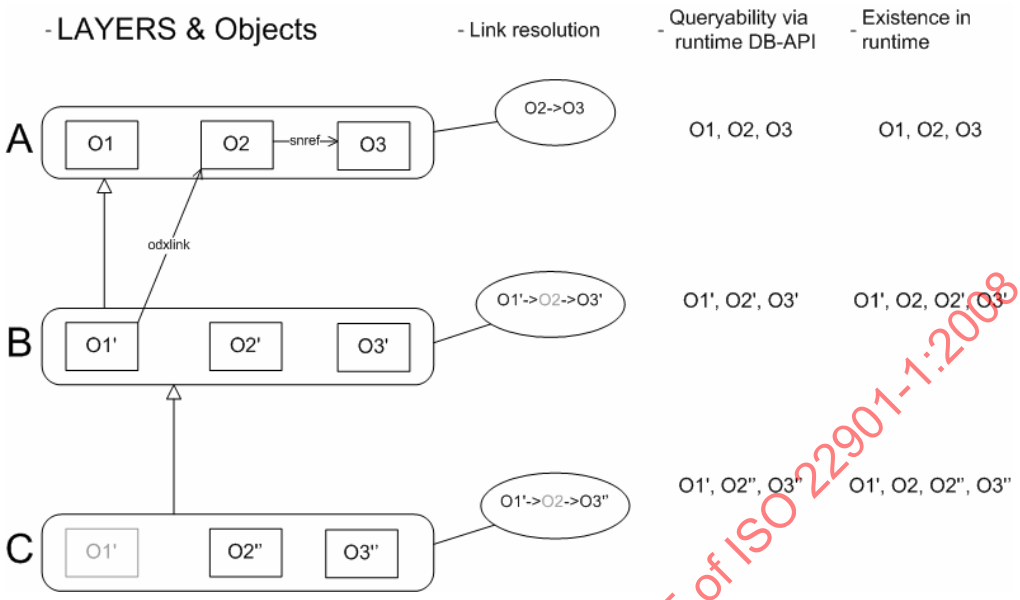


Figure 35 — Overriding objects referenced by odxlink

EXAMPLE 6 In the example in Figure 36 — Referencing object in ECU-SHARED-DATA the SNREF from O2 to O3 in the ECU-SHARED-DATA can be resolved inside the ECU-SHARED DATA, however, it cannot be resolved in layer B since O3 is not imported into B. This situation constitutes a configuration invalid in ODX and should be flagged as an error.

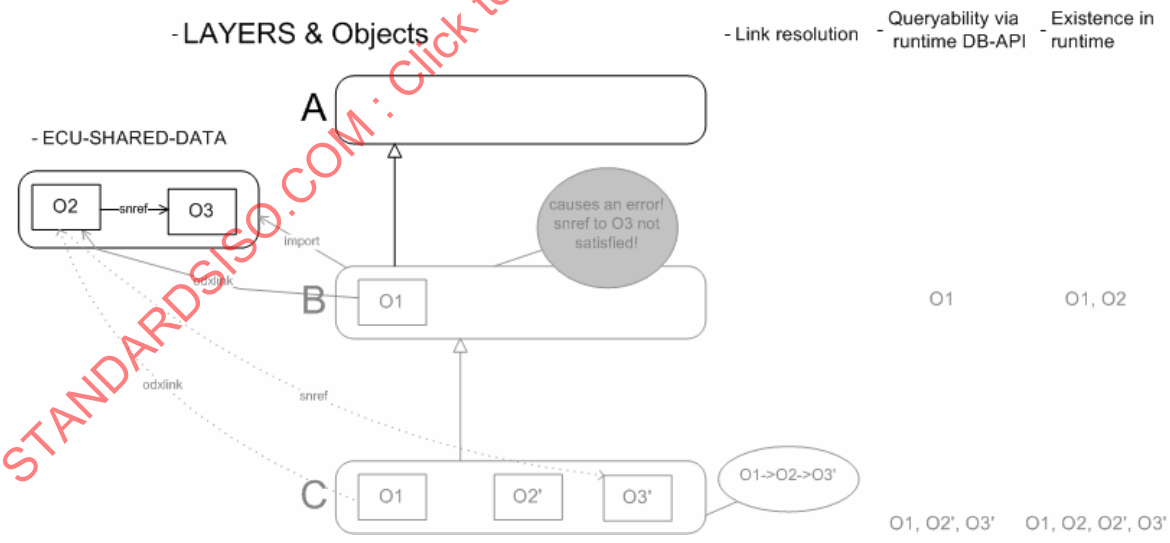


Figure 36 — Referencing object in ECU-SHARED-DATA

EXAMPLE 7 When there is an object O3 in B, as shown in Figure 37 — Overriding object from ECU-SHARED-DATA in other layers, the SNREF between O2 and O3' in the ECU-SHARED-DATA can also be resolved in B. This example is valid in ODX.

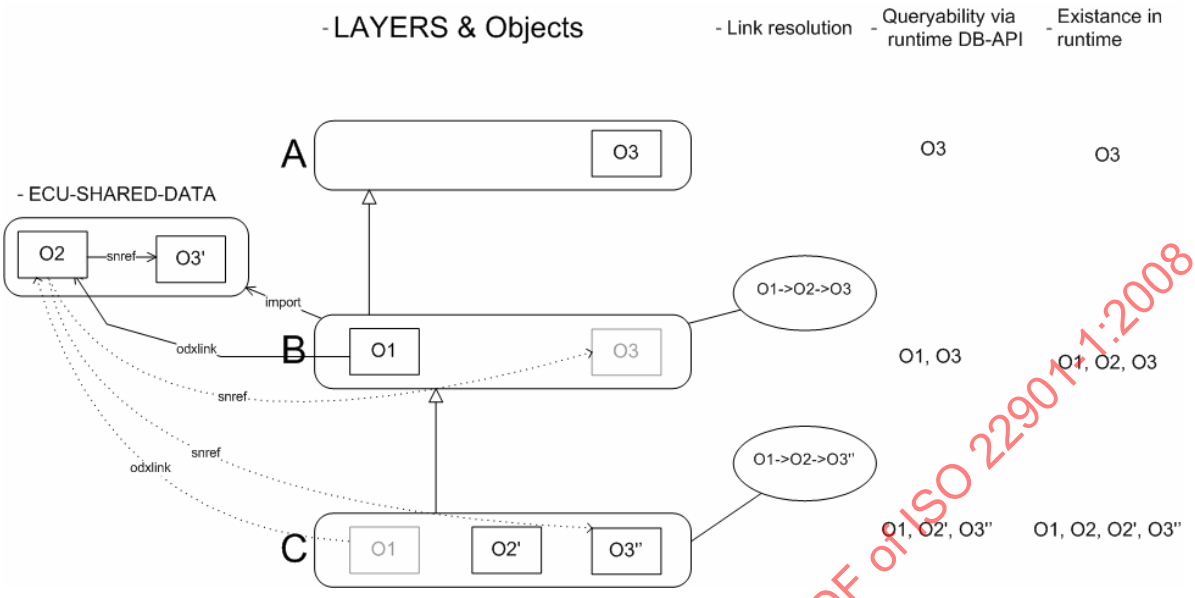


Figure 37 — Overriding object from ECU-SHARED-DATA in other layers

EXAMPLE 8 Like in any other layer, odxlinks preserve the identity of the objects they point to, see Figure 38 — odxlinks inside ECU-SHARED-DATA. Overriding odxlink targets affects the visibility of objects. Overridden objects are replaced by the overriding object at and below where they are defined in the inheritance tree.

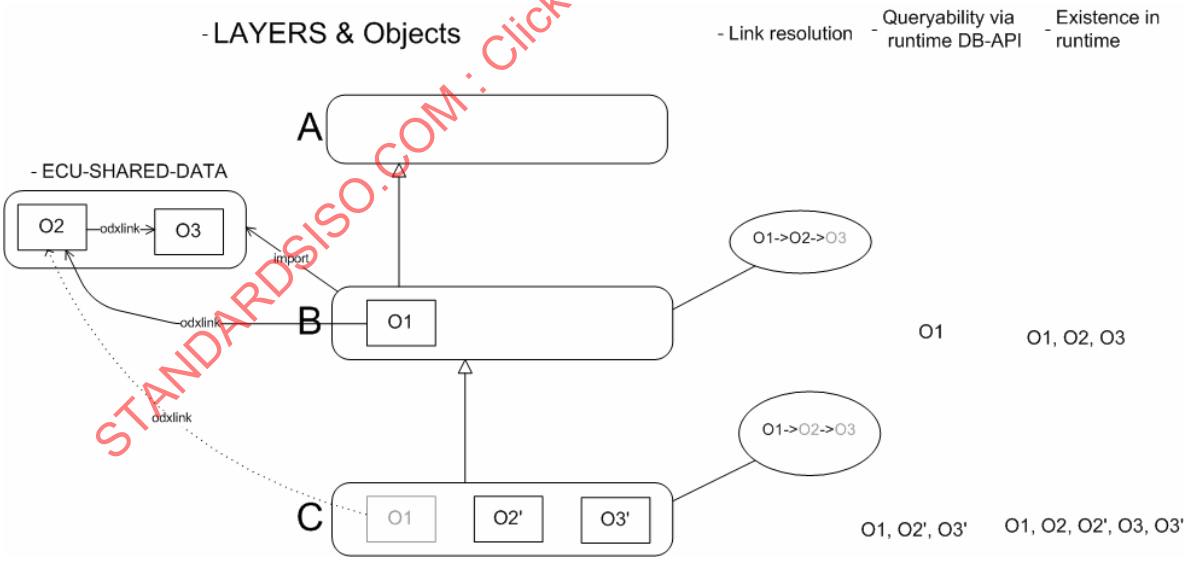


Figure 38 — odxlinks inside ECU-SHARED-DATA

EXAMPLE 9 The situation shown in Figure 39 — odxlinks inside ECU-SHARED-DATA to overridden objects is identical to example 5. Overriding an object O3' in an ECU-SHARED-DATA, that an odxlink points to will preserve the identity of the referenced object O3', whether it is defined in an ECU-SHARED-DATA or elsewhere.

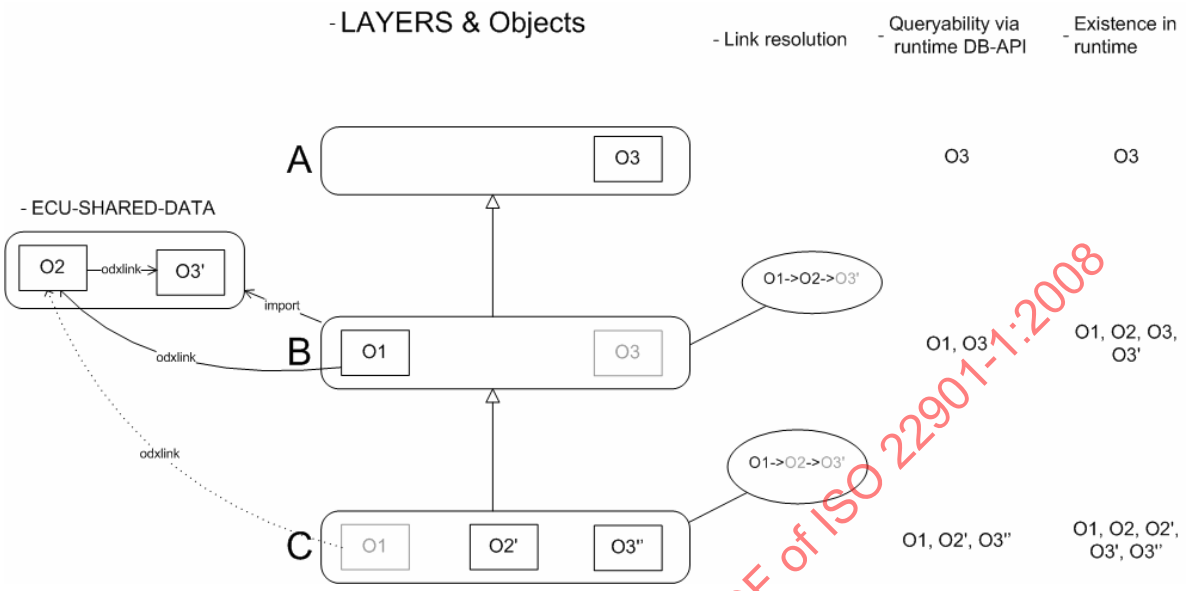


Figure 39 — odxlinks inside ECU-SHARED-DATA to overridden objects

EXAMPLE 10 As shown in Figure 40 — Overriding objects with references, if an object O1 is overridden by O1', the references of O1 are not automatically preserved in O1'.

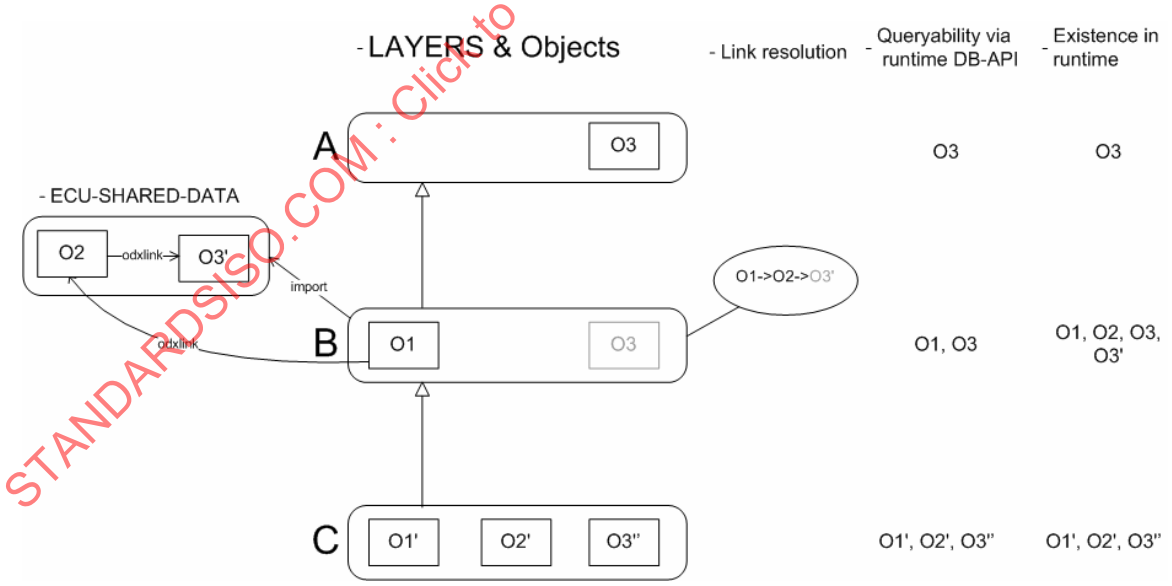


Figure 40 — Overriding objects with references

EXAMPLE 11 As shown in Figure 41 — NOT-INHERITED mechanism and odxlinks, an object O3, that has been excluded from inheritance remains usable in layer C if valid odxlinks are established to it, even though O3 is not listed when all objects of layer C are queried.

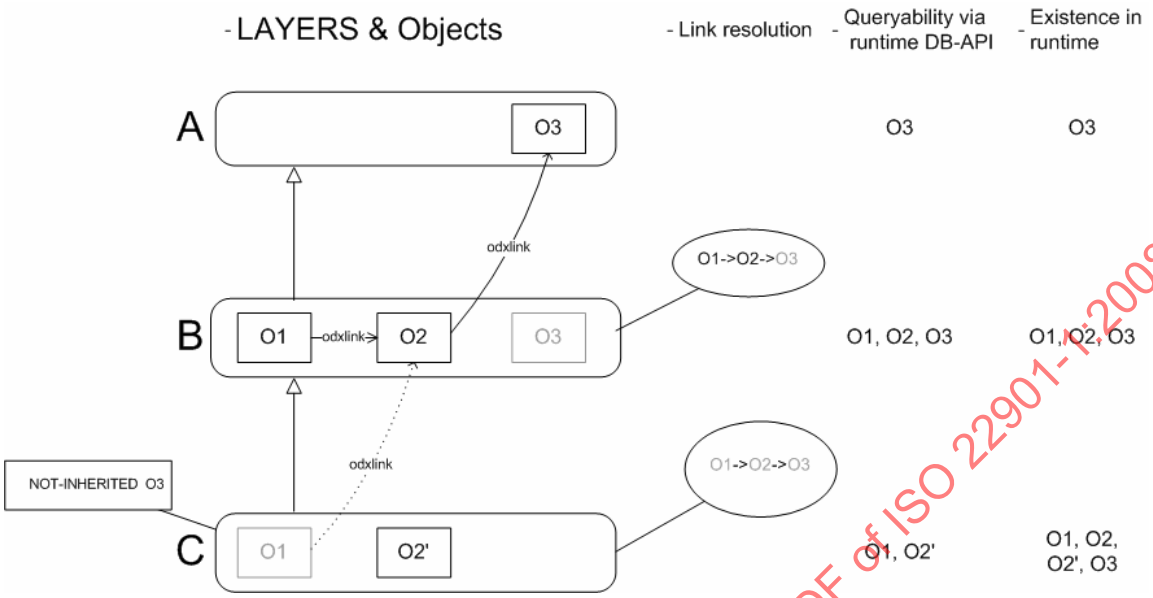


Figure 41 — NOT-INHERITED mechanism and odxlinks

EXAMPLE 12 As shown in Figure 42 — NOT-INHERITED mechanism and SNREFs, SNREFs, unlike odxlinks, are valid only, when the referenced object is visible in the respective layer. Example 12 constitutes an invalid specification because O3 is not a visible object in layer C. This situation can be resolved by using an odxlink in place of the SNREF (see Example 13) or by making O3 visible inside layer C.

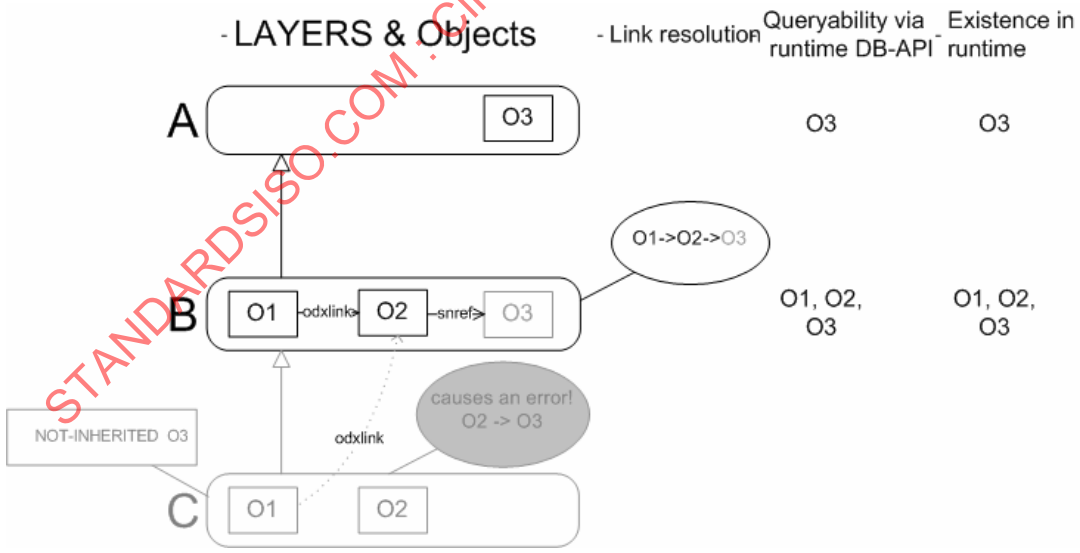


Figure 42 — NOT-INHERITED mechanism and SNREFs

EXAMPLE 13 Figure 43 — NOT-INHERITED mechanism, resolving an invalid SNREF by replacing it by an odxlink shows how to resolve the reference resolving problem of Example 12.

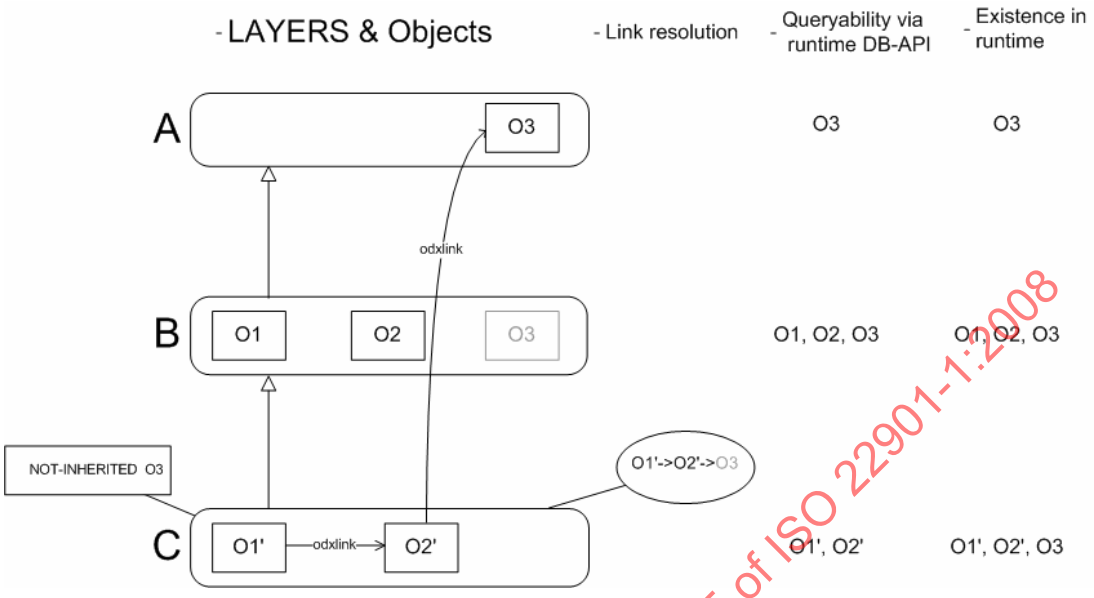


Figure 43 — NOT-INHERITED mechanism, resolving an invalid SNREF by replacing it by an odxlink

7.3.2.7 Variant identification

The ECU-VARIANT specialization of a DIAG-LAYER may also contain data to support variant identification at runtime. Figure 44 — UML representation of data structures for variant identification shows the data structures that are dedicated to that purpose. See 7.4.3 for details.

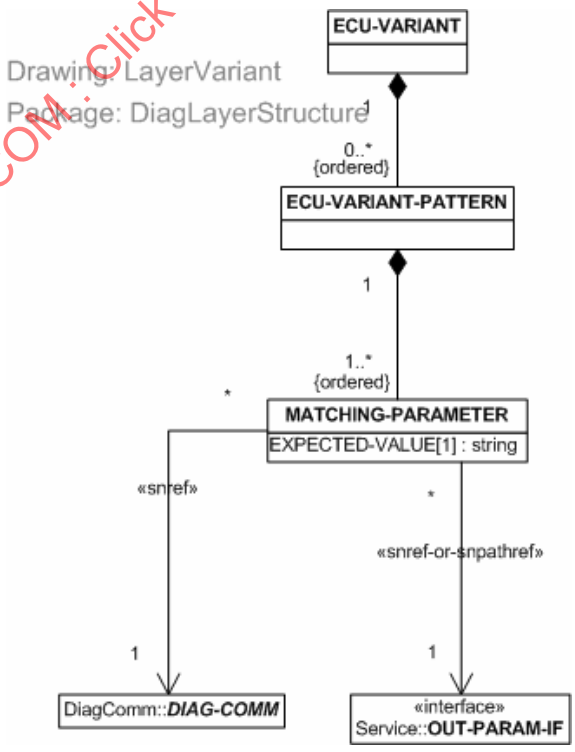


Figure 44 — UML representation of data structures for variant identification

7.3.2.8 Base variant identification

The BASE-VARIANT-PATTERN element makes it possible to specify how a base variant can be identified by communicating functionally or physically to an ECU. Figure 45 — UML representation of BASE-VARIANT-PATTERN shows the data structures that are dedicated to that purpose. See 7.4.4 for details on how to set up a base variant identification data structure.

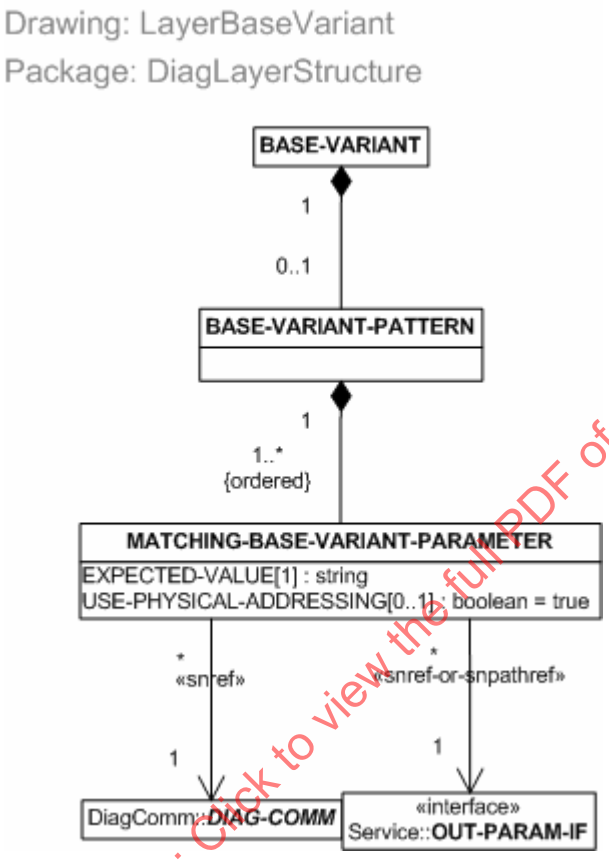


Figure 45 — UML representation of BASE-VARIANT-PATTERN

7.3.3 Communication parameter

The timing and the logical behaviour of the D-server is parameterized by communication parameters. These are represented in the ODX data by COMPARAM and COMPLEX-COMPARAM elements. For the definition of communication parameters with simple values (e.g. a P2 timing parameter, etc.), the COMPARAM element shall be used. Structured communication parameters consisting of multiple simple COMPARAMs (e.g. an array of CAN-IDs for functional addressing) shall be defined using the COMPLEX-COMPARAM element. The ALLOW-MULTIPLE-VALUES attribute of a COMPLEX-COMPARAM makes it possible to override a COMPLEX-COMPARAM with multiple value structures within one DIAG-LAYER. This is needed, e.g. in the use case of functional addressing, where the CAN-Ids of multiple responding ECUs need to be set up. The set of COMPARAMs and COMPLEX-COMPARAMs usually do not change the data bytes of the telegram from a diagnostic tester to the ECU, defined elsewhere in the data model, but may manipulate parts of the telegram header. An example is the ECU address in the header, or the CAN identifier. Communication parameters can only be defined within a COMPARAM-SUBSET as part of a COMPARAM-SPEC structure. See Figure 46 — Split up of the COMPARAM-SPEC structure (simplified).

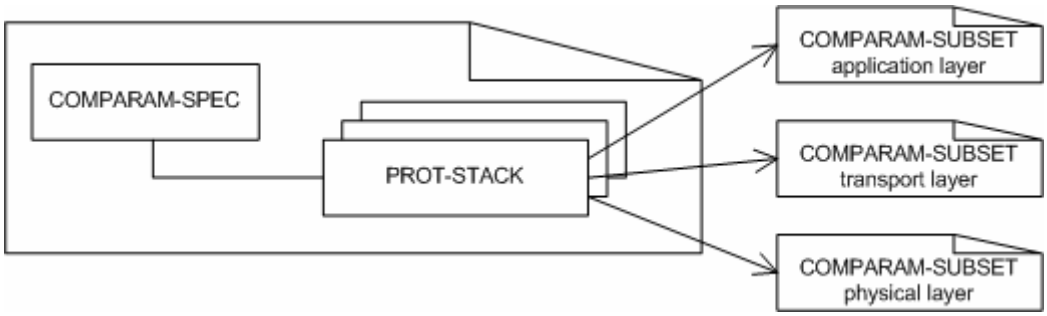


Figure 46 — Split up of the COMPARAM-SPEC structure (simplified)

A COMPARAM-SPEC structure consists of one or more sub-structures called PROT-STACK. A PROT-STACK contains a protocol specific set of predefined communication parameters. The parameter values may be changed in context of a DIAG-LAYER or specific diagnostic service. The attribute PDU-PROTOCOL-TYPE at PROT-STACK defines the standard protocol where the communication parameter set is designed for. The standard protocol shall be well known to the protocol driver. This protocol type name should be a concatenation of the application layer specification name plus the transport layer specification name, connected by the additional string "_on_". The master reference for the list of valid protocol type names is the D-PDU API specification.

The manner in which the PROT-STACKs are grouped by the COMPARAM-SPEC structure is not defined by this part of ISO 22901. It is recommended, however, to group all the PROT-STACK structures together into a COMPARAM-SPEC which belong to a combination of an application layer and a transport layer specification. The set of communication parameters defined by a PROT-STACK is subdivided into several COMPARAM-SUBSET structures according to the ISO OSI layer model. Usually there is one structure for each application layer, transport layer and physical layer. The attribute CATEGORY defines which layer a COMPARAM-SUBSET is designed for.

To ensure the exchangeability of ODX data, an ODX compliant tool shall support the exchange of COMPARAM-SPEC structures. The D-server identifies COMPARAMs and COMPLEX-COMPARAMs in the ODX data that matches the internally used communication parameters via their SHORT-NAME. As a consequence, the SHORT-NAME of the COMPARAMs and COMPLEX-COMPARAMs is standardized in ISO 22900-2, which is part of the standardized COMPARAM-SPEC structures which shall not be changed. In case of the non-standardized (system-specific) COMPARAMs and COMPLEX-COMPARAMs with a different semantic to the standardized ones, the SHORT-NAMEs can be freely chosen. The mapping of such parameters is not specified within this part of ISO 22901.

The communication parameters defined in a COMPARAM-SPEC or rather their values can be changed outside the COMPARAM-SPEC structure by referencing the COMPARAM or COMPLEX-COMPARAM element via «odxlink» and defining a different value. This can be done using a COMPARAM-REF element.

For a COMPARAM-REF to a COMPARAM, the new value shall be defined as SIMPLE-VALUE.

```
<COMPARAM-REF ID-REF="CP_14291">
  <SIMPLE-VALUE>2001</SIMPLE-VALUE>
</COMPARAM-REF>
```

A COMPARAM-REF to a COMPLEX-COMPARAM requires all SIMPLE-VALUEs of the including COMPARAMs to be covered by a COMPLEX-VALUE. That shall be done in the same order like the COMPLEX-COMPARAM is defined in the COMPARAM-SPEC.

```
<COMPARAM-REF ID-REF = "CCP_11381">
  <COMPLEX-VALUE>
     <SIMPLE-VALUE>4</SIMPLE-VALUE>
      <SIMPLE-VALUE>2018</SIMPLE-VALUE>
      <SIMPLE-VALUE>0</SIMPLE-VALUE>
  </COMPLEX-VALUE>
</COMPARAM-REF>
```

A COMPARAM-REF may also be directed at a COMPARAM that is specified within a COMPLEX-COMPARAM to only override parts of a COMPLEX-COMPARAM's default values. In this case, the COMPARAM-REF is used in the exact same manner as for a regular COMPARAM.

COMPARAM-REF elements can be attached at a DIAG-LAYER in case of changing a communication parameter value globally for all DIAG-COMMs of this diagnostic layer. The use of the COMPARAM-REF can be called "overriding of a communication parameter value" because the communication parameter values are inherited from DIAG-LAYER to DIAG-LAYER (diagnostic layer). The inheritance of data between DIAG-LAYERs is usually called "value inheritance" within this part of ISO 22901. As already mentioned, a PROT-STACK contains a protocol specific set of predefined communication parameters. This indicates that at any time during runtime exactly one PROT-STACK is active. The valid PROT-STACK at a time is referenced via SHORT-NAME by the active PROTOCOL or by the active LOGICAL-LINK. If there is a reference defined from the active PROTOCOL and also from the LOGICAL-LINK they shall be identical. See Figure 47 — UML representation of communication parameters.

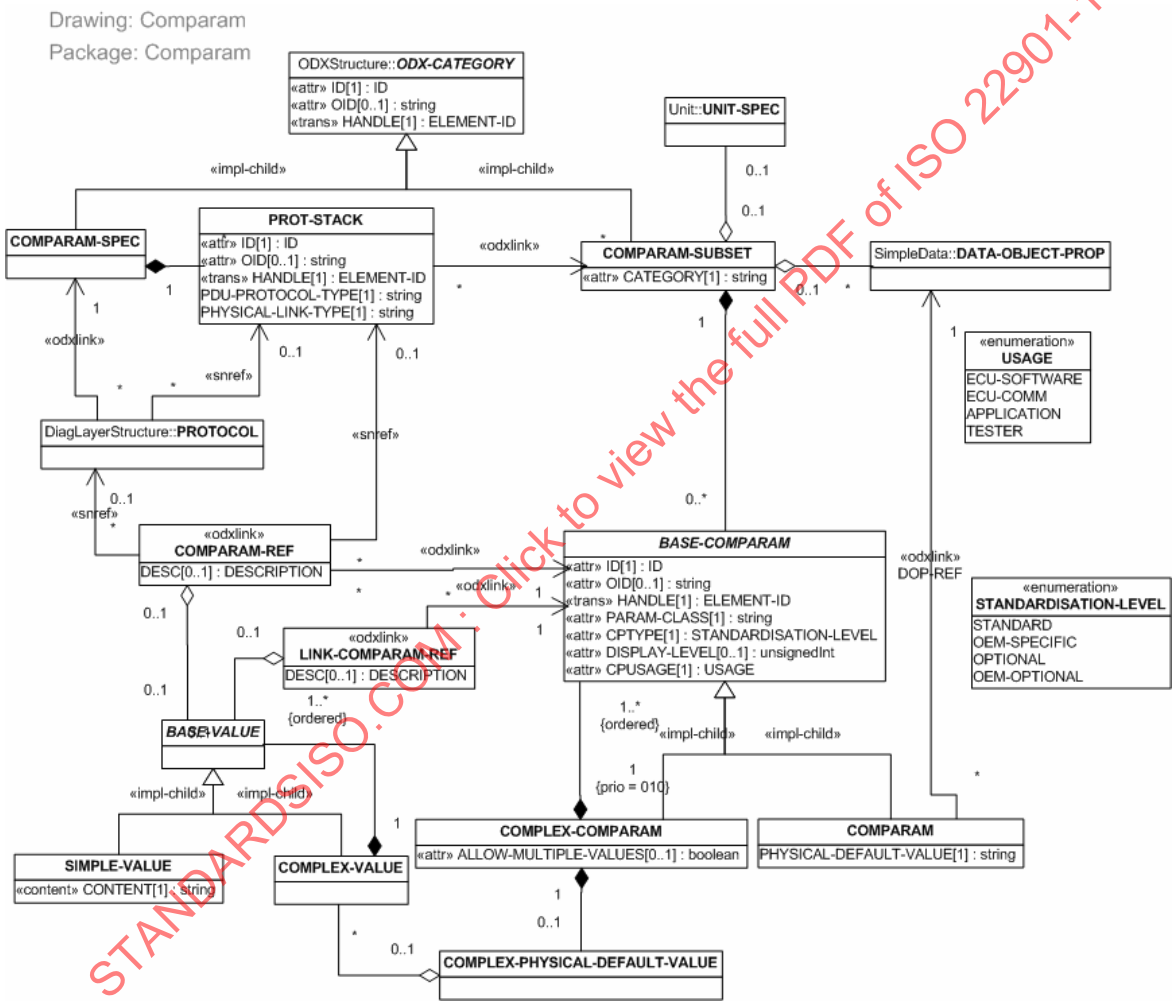


Figure 47 — UML representation of communication parameters

Each COMPARAM has a name (SHORT-NAME), a physical default value (PHYSICAL-DEFAULT-VALUE), and a reference to a data object property (DATA-OBJECT-PROP-REF) for the conversion of physical to internal values. The referenced DATA-OBJECT-PROPs (including their UNITs) are also stored within the COMPARAM-SUBSET.

COMPARAMs and COMPLEX-COMPARAMs have attributes to add semantic information necessary for the D-server or application software. With the attribute PARAM-CLASS the communication parameter can be semantically classified in view of its use. The PARAM-CLASS is mostly for documentation purposes and does not carry runtime semantics for the D-server. All communication parameters marked with the PARAM-CLASS UNIQUE_ID are exceptions. These are used by the D-server to feed the D-PDU API correctly with addressing information. The value set of PARAM-CLASS is extendable. In the following the corresponding predefined set of values is listed alongside with the values meanings:

a) TIMING: message flow timing parameters, e.g. inter byte time or time between request and response;

b) INIT: parameters for initiation of communication, e.g. trigger address or wakeup pattern; these parameters shall not be overridden within ECU-Variant layer in any way;

c) COM: general communication parameter;

d) ERRHDL: parameter defining the behaviour of the D-server in case an error occurred, e.g. D-server could either continue communication after a timeout was detected, or stop and reactivate;

e) BUSTYPE: used to define bus type specific parameters (e.g. baud rate). most of these parameters affect the physical hardware and can only be modified by the first LOGICAL-LINK that acquired the physical resource; when a second LOGICAL-LINK is created for the same resource, these parameters that were previously set will be active for the new LOGICAL-LINK;

f) UNIQUE_ID: this type of communication parameter is used to indicate to both the ComLogicalLink and the Application that the information is used for protocol response handling from a physical or functional group of ECUs to uniquely define an ECU response;

g) TESTER_PRESENT: this type of communication parameter is used to indicate to a server (or servers) that a client is still connected to the vehicle and that certain diagnostic services and/or communication that have been previously activated are to remain active.

An attribute named CPTYPE is used by the D-server to take the right choice of behaviour if an unsupported comparam occurs.

NOTE Unsupported means that the D-server has no knowledge about this communication parameter and how it needs to be handled.

The values of CPTYPE listed below are permitted.

— STANDARD The communication parameter specified by ISO 22900-2 for a standardized protocol (or defined by this part of ISO 22901) shall be supported by the D-server implementing the corresponding protocol. Diagnostic data using a standardized protocol not supported by the D-server cannot be executed by the D-server. If the D-server detects an unsupported communication parameter of type STANDARD, it shall stop the execution of the diagnostic data and provide an error message.

— OEM-SPECIFIC The communication parameter is part of a non-standardized OEM-specific protocol; nevertheless it is required to be implemented by the runtime system. Diagnostic data using an OEM-specific protocol not supported by the runtime system cannot be executed by the runtime system. If the runtime system detects an unsupported communication parameter of type OEM-SPECIFIC, it shall stop the execution of the diagnostic data and provide an error message.

— OPTIONAL This communication parameter does not have to be supported by the D-server. If a DIAG-COMM uses an unsupported comparam of this type the comparam can be ignored and the DIAG-COMM can be executed nevertheless.

— OEM-OPTIONAL This communication parameter covers the OEM specific comparams that need not to be supported by the runtime system.

— The DISPLAY-LEVEL is used to restrict visibility (and therefore changeability) of the COMPARAMs in an application. Level 1 means that this can be changed for all users. Level 2 means that not everybody is allowed to change these communication parameter. The maximum value of DISPLAY-LEVEL is 5. The behaviour of the different applications according to the display levels can not be defined by the ODX standard. Therefore the support of DISPLAY-LEVEL is optional.

The attribute CPUSAGE has the possible values “ECU-SOFTWARE”, “APPLICATION”, “ECU-COMM” and “TESTER”. ECU-SOFTWARE parameters are communication parameters that are only used for ECU software generation and configuration. Communication parameters of this type shall be fully ignored by the D-server. APPLICATION parameters are only evaluated by the application. Communication parameters of this type shall not be passed down by the D-server to the D-PDU API, but may be used by the D-server itself. In any case, they shall be accessible via the D-server. ECU-COMM parameters are parameters needed to communicate with the ECU (e.g. timings and addresses). A parameter of the kind “ECU-COMM” is relevant both to the tester during tester communication and to the ECU software generation/calibration use case. TESTER parameters are only valid to the tester during diagnostic communication, they are not relevant to the use case of ECU software generation and calibration.

7.3.4 Inheritance of communication parameters

During runtime a diagnostic layer is linked with a PROT-STACK to import a valid set of communication parameters (COMPARAM and COMPLEX-COMPARAM elements). This unambiguous link is defined via the PROTOCOL or the LOGICAL-LINK. The COMPARAM values are inherited between the different layers. An inherited COMPARAM value is identical to that in the parent layer, i.e. it has the same value.

A communication parameter shall be overridden to change its value. For that purpose the communication parameter is referenced by a COMPARAM-REF. The SIMPLE-VALUE or COMPLEX-VALUE of the COMPARAM-REF defines the new value(s) of the referenced COMPARAM or COMPLEX-COMPARAM, respectively. The structure of a COMPLEX-VALUE shall match the structure of the COMPLEX-COMPARAM which is referred to by type and order of the contained sub elements. The PROTOCOL-SNREF can be used to define that the COMPARAM-REF should be applied only for the referenced protocol. If the PROTOCOL-SNREF is not set, the COMPARAM-REF is valid for all applicable protocols. The PROT-STACK-SNREF can be used to define that the COMPARAM-REF should be applied only for the referenced protocol stack. If the PROT-STACK-SNREF is not set, the COMPARAM-REF is valid for all applicable protocol stacks. A COMPARAM-REF detected at runtime that does not match the current PROTOCOL/PROT-STACK combination has no runtime relevance and shall be ignored.

The LINK-COMPARAM-REF at LOGICAL-LINK and PHYSICAL-VEHICLE-LINK is similar to COMPARAM-REF but without a PROTOCOL-SNREF and PROT-STACK-SNREF. The protocol and the protocol stack are specified by the LOGICAL-LINK which either contains the LINK-COMPARAM-REF or which refers to the PHYSICAL-VEHICLE-LINK containing it. Again, the D-servers ignores any LINK-COMPARAM-REF not applicable for the current combination of PROTOCOL-SNREF and PROT-STACK-SNREF defined at the LOGICAL-LINK.

See Figure 48 — UML representation of overriding of communication parameters (incomplete structure).

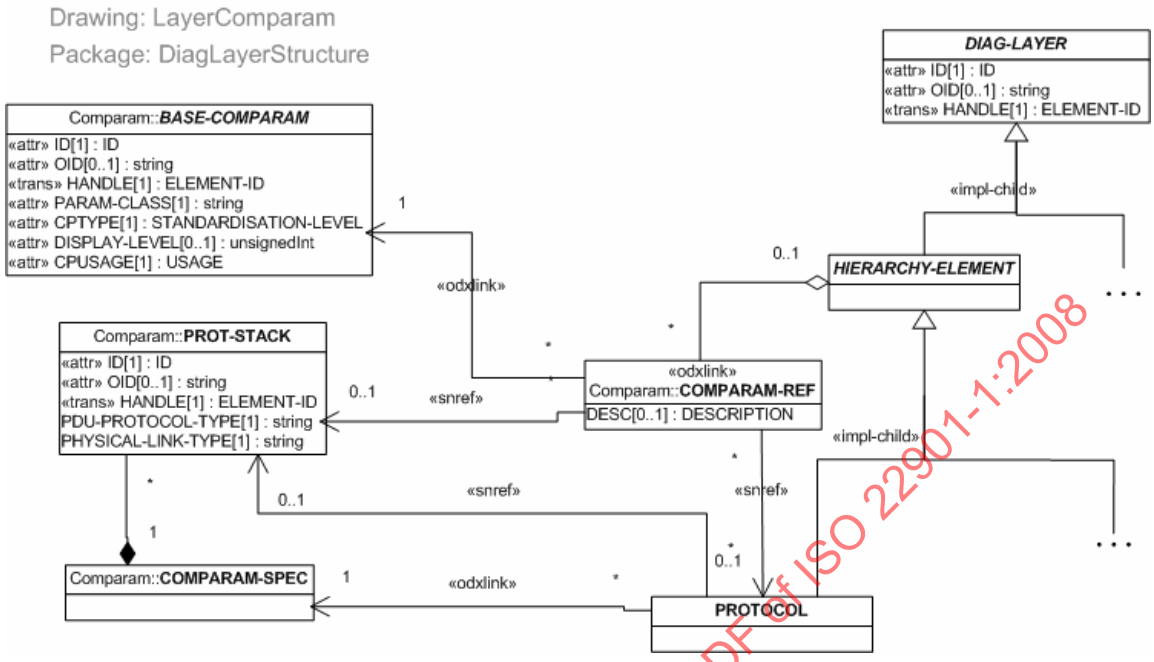


Figure 48 — UML representation of overriding of communication parameters (incomplete structure)

If a communication parameter value is overridden within the scope of a DIAG-LAYER, the changed communication parameter value is valid for all DIAG-SERVICEs that are available on this layer including DIAG-SERVICEs that were inherited from the parent layer. It is also possible to override a COMPARAM value for a single DIAG-SERVICE. The scope of this change is restricted to this specific DIAG-SERVICE. communication parameter values for a specific DIAG-SERVICE are passed on implicitly by inheritance of the DIAG-SERVICE they belong to. Therefore, they cannot be overridden within the scope of a diagnostic layer. Only overriding the whole DIAG-SERVICE can change them. If a communication parameter value is overridden in both the DIAG-SERVICE and the diagnostic layer, the DIAG-SERVICE-specific COMPARAM value takes precedence over the layer-specific one, i.e. the communication parameter value from the DIAG-SERVICE is used.

Overridden communication parameter values on a FUNCTIONAL-GROUP level are not inherited to lower layers (i.e. the inheriting base variants). Base variants take their set of communication parameter values directly from the protocol and then override with locally defined values. Since a logical link always includes one dedicated protocol, all multiple inheritance issues can unambiguously be resolved at runtime.

An overview of the precedence rules for communication parameter values is shown in Table 3 — Precedence rules for communication parameters (from lower to higher precedence).

Table 3 — Precedence rules for communication parameters

Location of value definition in ODX data model	Type of value definition
COMPARAM and COMPLEX-COMPARAM (in COMPARAM-SUBSET)	Default definition
diagnostic layer [PROTOCOL]	Override
diagnostic layer [FUNCTIONAL-GROUP]	Override (Attention: overwritten communication parameter values on FUNCTIONAL-GROUP level are not inherited to lower layers!)
diagnostic layer [BASE-VARIANT]	Override (PROTOCOL layer)
diagnostic layer [ECU-VARIANT]	Override (PROTOCOL layer, BASE-VARIANT layer)
PHYSICAL-VEHICLE-LINK	Override
LOGICAL-LINK	Override
DIAG-SERVICE	Override

In case a communication parameter value is overridden on the DIAG-SERVICE level, the D-server shall ensure that, after a DIAG-SERVICE is completed, all communication parameters have the same values that they had before its execution.

EXAMPLE Simplified inheritance example for simple communication parameters (COMPARAMs only).

As shown in Figure 49 — UML representation of inheritance of communication parameters (example), a PROTOCOL instance (PROT-A) references the PROT-STACK instance PS-B as the active PROT-STACK. As a consequence, the valid set of communication parameters consists of A, B and C. Because COMPARAM A value is overridden within the protocol layer, the valid values in the scope of PROT-A are A=4, B=5 and C=15.

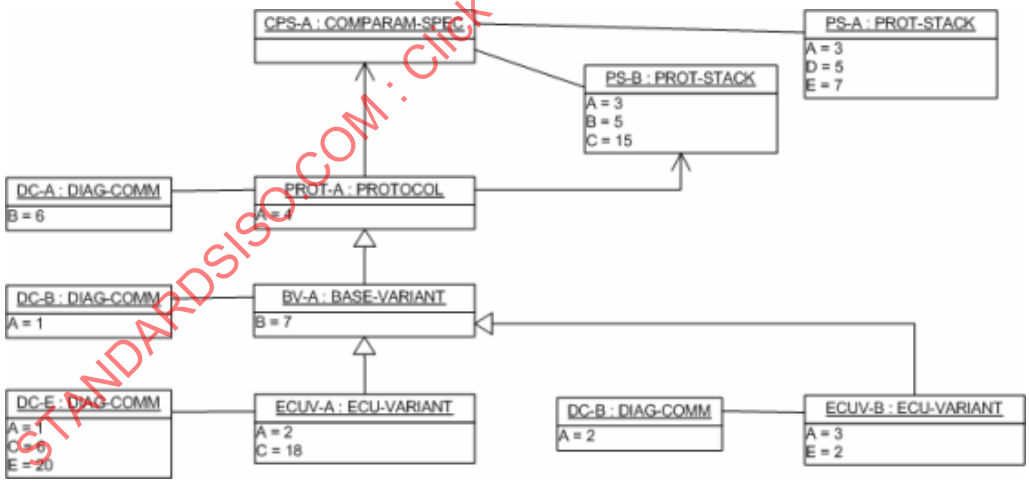


Figure 49 — UML representation of inheritance of communication parameters (example)

These values apply to all the DIAG-COMM that do not override a communication parameter value themselves. The DIAG-SERVICE with the identifier DC-A in the PROTOCOL PROT-A shown in the example overrides the COMPARAM B value. This is done within the data by a COMPARAM-REF element. The COMPARAM set for this special DIAG-SERVICE results in [A=4, B=6, C=15].

The BASE-VARIANT instance BV-A overrides the COMPARAM B value. All other COMPARAM values are derived from the protocol layer PROT-A. The COMPARAM set in the scope of BV-A is [A=4, B=7 C=15]. Two DIAG-COMMs are available in this base variant layer. The DIAG-SERVICE DC-B defined within BV-A and the

DIAG-SERVICE DC-A derived from the parent layer PROT-A. The COMPARAM set for DC-A at the base variant level is [A=4, B=6, C=15] and COMPARAM set for DC-B is [A=1, B=7, C=15].

There are two instances of an ECU-VARIANT both inherited from BV-A. For ECUV-A the COMPARAM set is [A=2, B=7, C=18]. There are three DIAG-SERVICEs available at ECUV-A. The inherited DC-A with [A=2, B=6, C=18], the inherited DC-B with [A=1, B=7, C=18] and locally defined DC-E with [A=1, B=7, C=6]. Because the COMPARAM E is not part of the current communication parameter set the COMPARAM-REF element at DC-E that tries to override the COMPARAM E value exclusively for the DIAG-SERVICE has no affect and can be ignored. The ECU-VARIANT instance ECU-B applies the COMPARAM values [A=3, B=7, C=15]. E is ignored because it is not part of the active PROT-STACK. The available DIAG-SERVICEs are the value inherited DC-A with [A=3, B=6, C=15] and the locally defined DC-B which overrides the DC-B defined at the layer above. The COMPARAM set for DC-B in the scope of ECUV-B is [A=2, B=7, C=15]. There is no other way around to change a COMPARAM value which is already overridden at the DIAG-SERVICE in a parent layer than to override the DIAG-SERVICE as a whole.

7.3.5 Datastream

7.3.5.1 Overview

This subclause (7.3.5) deals with the main topic of ODX. It specifies the diagnostic communication between a tester and an ECU. The communication process consists of two parts:

— request, and

— response.

NOTE 1 ODX only describes the PDU but not the header and checksum of the message. The service identifier is part of the PDU.

Figure 50 — Datastream example shows a request message sent by the tester to the ECU, e.g. if the tester wants to read or to write data. Normally, the ECU replies with a response message that contains the read data or the information about success of the writing operation. Figure 50 shows a simple communication between a tester and an ECU. The tester requests three bytes of data, which should be read starting at the memory address 0x204813 in the ECU's memory. The request message consists of three parameters: SID (service identifier), Address and Size and has the length of 5 bytes. The tester receives 4 bytes as the response to its request, which represent two parameters: SID and the requested three-byte value.

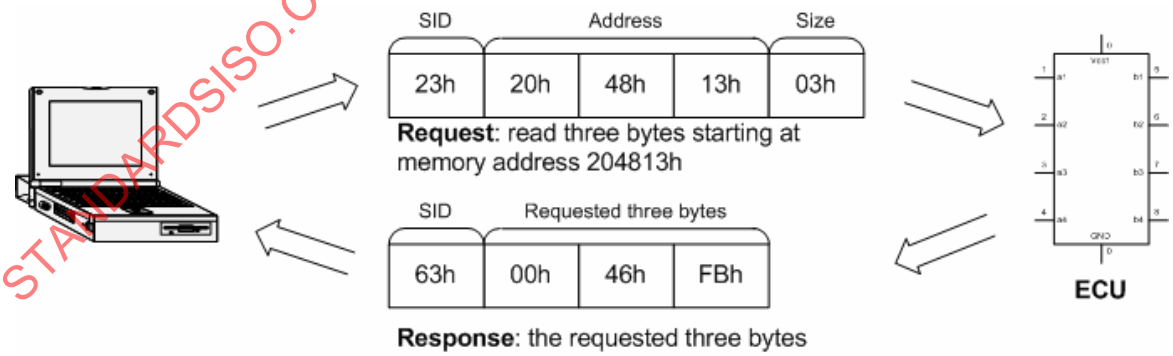


Figure 50 — Datastream example

NOTE 2 The memory addresses in ODX (type hexBinary) are mapped to A_UINT32 value in ISO 22900-3. As a result of that circumstance only memory address information up to 4 bytes length can be used.

Figure 51 — UML representation of datastream shows the overview of the packages described in this subclause. Service, Response, Request and Parameters are already mentioned above. Another possibility to communicate with an ECU consists in using diagnostic jobs. They can be seen as complex services and, in fact, they are based on them. A job uses one or more services for communication and is useful for performing

of more complex tasks than simple services can offer. Thus, DIAG-COMMs are two diagnostic communication primitives (DIAG-COMMs) used to communicate with an ECU. They are the two implementing peculiarities of the abstract class DIAG-COMM, which combines the common characteristics of jobs and services, described in the package DiagComm.

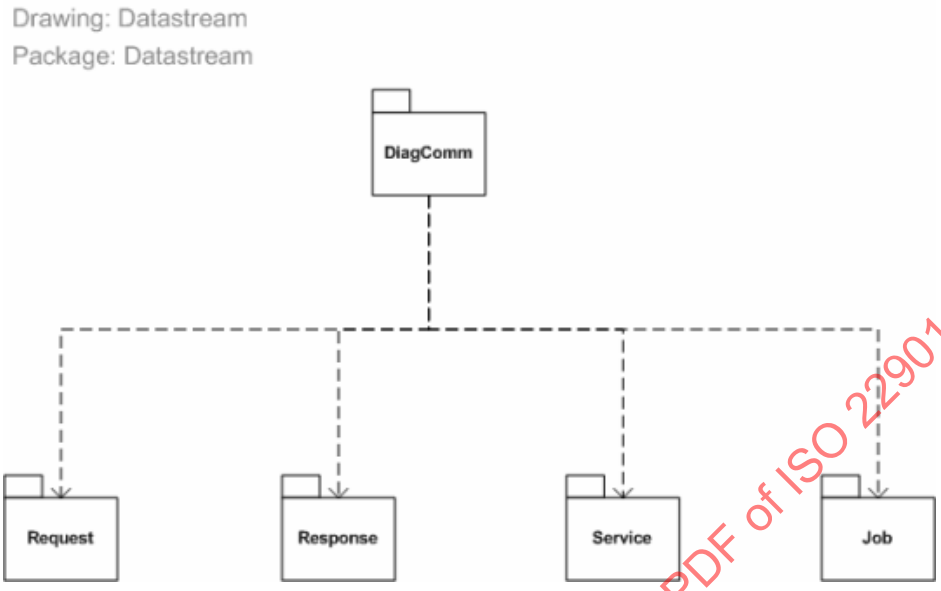


Figure 51 — UML representation of datastream

7.3.5.2 Diagnostic communication

A diagnostic communication (DIAG-COMM) provides common characteristics of diagnostic services (DIAG-SERVICEs) and jobs (SINGLE-ECU-JOBs). It defines a communication process between a tester and an ECU (or a group of ECUs) in order to enquire diagnostic information from the ECUs or/and to modify their behaviour for diagnostic purposes.

See Figure 52 — UML representation of diagnostic communication.

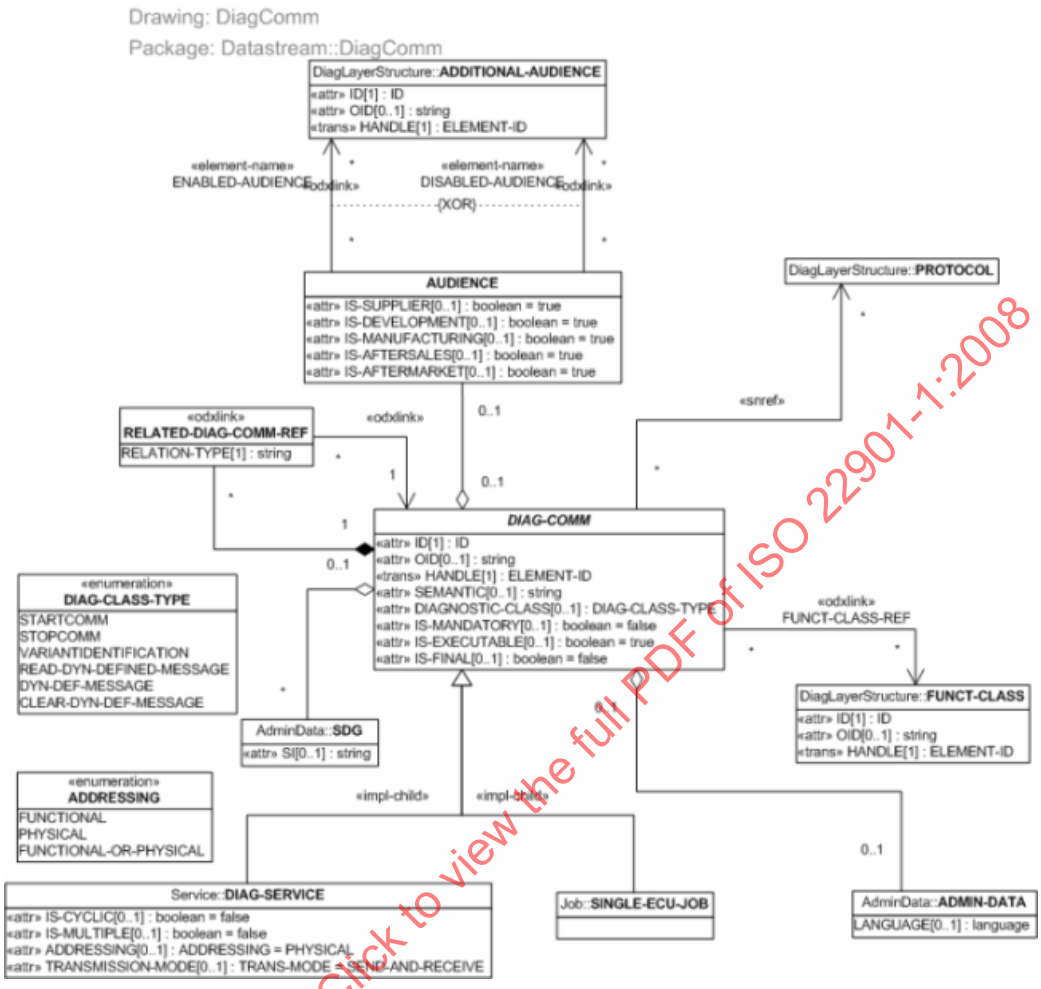


Figure 52 — UML representation of diagnostic communication

In addition to the standard members ID, HANDLE, ADMIN-DATA and SDGs, the members listed below are available for the characterization of a DIAG-COMM (a service or a job).

— AUDIENCE defines the target group or groups of users the DIAG-COMM is available for. For more details see 7.1.2.2.

— IS-MANDATORY indicates whether the DIAG-COMM can be eliminated in lower diagnostic layers by using NOT-INHERITED-DIAG-COMM. The value “true” means that the DIAG-COMM shall not be eliminated. By default each DIAG-COMM can be eliminated (IS-MANDATORY = “false”).

— IS-FINAL indicates whether the DIAG-COMM can be overridden in lower diagnostic layers by declaring a DIAG-COMM with the same SHORT-NAME. The value “true” means that the DIAG-COMM shall not be overridden. By default each DIAG-COMM can be overridden (IS-FINAL = “false”).

Some services or jobs can only be used as part of other jobs and it does not make sense to use them separately. IS-EXECUTABLE[3)] can be used in this case to tell the tool to hide this DIAG-COMM by setting IS-EXECUTABLE to “false”. The default value of this attribute is “true”.

Via RELATED-DIAG-COMM-REF, it is possible to reference one or more DIAG-COMMs that are in relationship with this DIAG-COMM. It is merely used for information purposes. RELATION-TYPE indicates the user-specific type of the relation (e.g. INPUT, OUTPUT, SESSION or SECURITY).

The DIAGNOSTIC-CLASS attribute denotes the purpose of the DIAG-COMM and may be used by the run time system or other jobs for functional identification of a service or a job, e.g. to start the communication with an ECU a service with DIAGNOSTIC-CLASS=“STARTCOMM” is used. Possible values for this attribute are: STARTCOMM, STOPCOMM, VARIANTIDENTIFICATION, DYN-DEF-MESSAGE, READ-DYN-DEF-MESSAGE and CLEAR-DYN-DEF-MESSAGE. STARTCOMM and STOPCOMM shall not exist more than one time per layer. They do not have to be present, if the ECU does not need an explicit initialisation message (see Table A.9 — Enumeration “DIAG-CLASS-TYPE”).

The member SEMANTIC is used for classifying of DIAG-COMMs. The pre-defined values can be found in Table A.1 — SEMANTIC at DIAG-COMM.

Each service or job can be assigned to a functional class using certain criteria. The assignment takes place by referencing the class with FUNCT-CLASS-REF. A service or a job may be assigned to one or more functional classes but also to no class. Such grouping of DIAG-COMM instances can be used by the application to sort them according to the defined criteria.

PROTOCOL-SNREF at DIAG-COMM is used to determine the protocol the DIAG-COMM is valid for. This information is given by referencing an arbitrary number of PROTOCOL instances via SHORT-NAME. If no protocol is referenced all protocols in the transitive closure are supported. For DIAG-COMMs defined in an ECU-SHARED-DATA layer, it makes sense to define, under which protocol the DIAG-COMM can be executed. To enable the reuse of one ECU-SHARED-DATA instance for various ECUs with different protocols supported, PROTOCOL-SNREF inside ECU-SHARED-DATA may refer to a protocol outside the transitive closure of an ECU. Such a reference is allowed inside ECU-SHARED-DATA and will be ignored by the D-server.

7.3.5.3 Service

Figure 53 — UML representation of service defines a diagnostic services (DIAG-SERVICEs) of a communication process between a tester and an ECU or a group of ECUs in order to enquire diagnostic information from the ECUs or/and to modify their behaviour for diagnostic purposes. The tester initialises a service by sending a request message. REQUEST-REF is used to reference an appropriate request object that describes the structure of the request message. If no error occurs, the ECU replies by sending a positive response message. POS-RESPONSE-REF is used to reference the corresponding response object that describes the structure of the response message. In case of an error the tester receives one or more negative response messages. The description of them is done by NEG-RESPONSEs, which are referenced by NEG-RESPONSE-REFs from the DIAG-SERVICE.

A DIAG-SERVICE neither containing a POS-RESPONSE-REF nor a REQUEST-REF is not valid.

3) A D-server will deny a client application to execute a DIAG-COMM or any DIAG-COMM used by a TABLE-ROW as runtime request parameter if the attribute IS-EXECUTABLE is set to ‘false’. Within a Java job’s source code, the value of the IS-EXECUTABLE attribute is ignored.

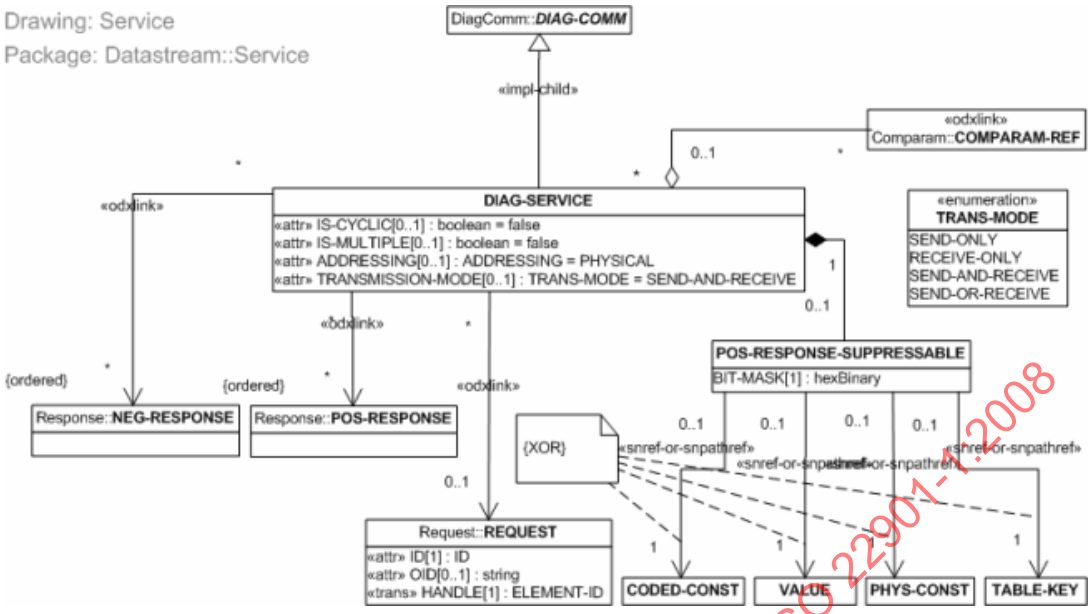


Figure 53 — UML representation of service

A DIAG-SERVICE inherits all the members defined within the DIAG-COMM class. In addition, the members IS-CYCLIC and IS-MULTIPLE belong to a DIAG-SERVICE and it refers to related services, to a request and to positive and negative responses.

IS-CYCLIC indicates whether the service is used to request data that is sent repeatedly. For example a tester can request the machine speed that is to be sent at intervals of one second. The ECU replies on such a request by repeatedly sending responses to the tester where each response contains the actual machine speed. The default value of this member is “false” which indicates that no cyclic responses are to be expected. The periodic flag shall not be combined with DIAGNOSTIC-CLASS='VARIANTIDENTIFICATION' of DIAG-COMM.

Using some services multiple messages can be expected as responses to the request. In this case IS-MULTIPLE is to be set to “true”. By default only a single response message is expected. As a stop criterion serves a timeout parameter. Use case for IS-MULTIPLE is handling protocols where e.g. DTCs are split up into multiple responses. In that case an MCDResult should contain multiple MCDResponse Objects in the MVCI Diagnostic Server.

Corresponding to the optional attribute TRANSMISSION-MODE the D-server will handle the Service in several ways. Possible values are as described below.

— SEND-ONLY: The D-server sends a request message and will not expect any kind of response. Neither positive nor negative response. The classic use case is bus simulation.

— RECEIVE-ONLY The D-server will not send a request message, but shall listen for one of the referenced positive responses. The classic use case is for listening to onboard messages.

— SEND-AND-RECEIVE: This is the regular diagnostic service. The D-server sends a request message and will listen for one of the referenced positive responses.

— SEND-OR-RECEIVE: This value is a place holder. It can be changed to another transmission mode value during runtime before the DIAG-COMM is executed. Otherwise the DIAG-COMM will be executed in SEND-AND-RECEIVE transmission mode as default. Therefore DIAG-COMM of this type shall be defined like a DIAG-COMM specified as SEND-AND-RECEIVE including request and response. The classic use case is for response-on-event handling.

As mentioned above, REQUEST-REF is used to reference the appropriate request while POS-RESPONSE-REFs reference the proper positive responses. NEG-RESPONSE-REFs reference negative responses that may be returned if an error occurs. It is possible to define more than one positive or negative response for one service. The D-server shall set up a table with expected responses that is passed down to the D-PDU API. The information in this table is e.g. built from information retrieved from CODED-CONST and MATCHING-REQUEST parameters. For details, see the D-PDU API specification.

If inherited GLOBAL-NEG-RESPONSES and locally defined GLOBAL-NEG-RESPONSES are both valid in a specific location, the locally defined GLOBAL-NEG-RESPONSEs are evaluated before the inherited ones. In case of multiple inheritance the GLOBAL-NEG-RESPONSEs are evaluated in the alphabetical order of the SHORT-NAME of the DIAG-LAYER they have been inherited from.

If the POS-RESPONSE-SUPPRESSABLE element is defined at a DIAG-SERVICE, it means the application can choose whether it expects a positive response or not. If the application wants no positive response the BIT-MASK of POS-RESPONSE-SUPPRESSABLE is applied with an OR-semantic to the internal value of the referenced CODED-CONST, VALUE, TABLE-KEY or PHYS-CONST parameter of the parameter referenced by the POS-RESPONSE-SUPPRESSABLE element. If the element POS-RESPONSE-SUPPRESSABLE is not a sub-element of a DIAG-SERVICE it means the service is handled as usual.

EXAMPLE Diagnostic service with a POS-RESPONSE-SUPPRESSABLE element. If the positive response should be suppressed the following would happen: the BIT-MASK 0x80 is applied with an OR operation to the sub-function 18 (0x12), which results in 92hex being sent instead of 0x12 as subfunction.

```
<DIAG-COMMS>
  <DIAG-SERVICE ID="ExampleServiceID" ADDRESSING="FUNCTIONAL-OR-PHYSICAL">
    <SHORT-NAME>ExampleService</SHORT-NAME>
    <AUDIENCE/>
    <REQUEST-REF ID-REF="ExampleRequestID"/>
    <POS-RESPONSE-REFS>
      <POS-RESPONSE-REF ID-REF="ExampleResponseID"/>
    </POS-RESPONSE-REFS>
    <POS-RESPONSE-SUPPRESSABLE>
      <BIT-MASK>80</BIT-MASK>
      <CODED-CONST-SNREF SHORT-NAME="ParamOfSubFunction"/>
    </POS-RESPONSE-SUPPRESSABLE>
  </DIAG-SERVICE>
</DIAG-COMMS>
<REQUESTS>
     <REQUEST ID="ExampleRequestID">
          <SHORT-NAME>ExampleRequest</SHORT-NAME>
          <PARAMS>
               <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>SIDParam</SHORT-NAME>
                    <CODED-VALUE>15</CODED-VALUE>
                    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
                         <BIT-LENGTH>8</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
               </PARAM>
               <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>ParamOfSubFunction</SHORT-NAME>
                    <CODED-VALUE>18</CODED-VALUE>
                    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
                         <BIT-LENGTH>8</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
               </PARAM>
          </PARAMS>
     </REQUEST>
</REQUESTS>
```

ADDRESSING is used to define the addressing mode used by the DIAG-SERVICE. The valid values are FUNCTIONAL, PHYSICAL or FUNCTIONAL-OR-PHYSICAL whereby the default value is PHYSICAL. FUNCTIONAL means the DIAG-COMM will be executed using a functional address. In the case of PHYSICAL only an individual ECU can be addressed using its physical address. Finally, FUNCTIONAL-OR-PHYSICAL is used when both modes are supported.

The functional address is only be used in the Level of the FUNCTIONAL-GROUP layer. When communicating functional, it is possible to open a logical link to a functional group. Now, the functionally executed service is used as defined at this FUNCTIONAL-GROUP. However, the service could be overridden on BASE-VARIANT level. The meaning of the DIAG-SERVICE overriding with respect to the functional addressing is described in Table 4 — Overriding of functional services.

Table 4 — Overriding of functional services

FUNCTIONAL-GROUP ADDRESS	BASE-VARIANT ADDRESS	Semantics
FUNCTIONAL	FUNCTIONAL	Response message is only overridden for the functional case. The service shall not be offered as an executable service on the BASE-VARIANT or ECU-VARIANT level. An overridden request message is ignored.
FUNCTIONAL	FUNCTIONAL-OR-PHYSICAL	Response message overrides for functional and physical addressing. Request only overrides for physical addressing. On BASE-VARIANT or ECU-VARIANT level the service can only be executed physically.
FUNCTIONAL	PHYSICAL	Response message overrides for physical addressing only. For functional addressing, the response of the functional group is used. Request only overrides for physical addressing. On BASE-VARIANT or ECU-VARIANT level the service can only be executed physically.
FUNCTIONAL-OR-PHYSICAL	FUNCTIONAL-OR-PHYSICAL	Response message overrides for functional and physical addressing. Request only overrides for physical addressing. On BASE-VARIANT or ECU-VARIANT level the service can only be executed physically.
FUNCTIONAL-OR-PHYSICAL	PHYSICAL	Response message overrides for physical addressing only. For functional addressing, the response of the functional group is used. Request only overrides for physical addressing. On BASE-VARIANT or ECU-VARIANT level the service can only be executed physically.

A diagnostic service can redefine one ore more communication parameters by referencing an existing COMPARAM. A COMPARAM-REF contains a reference to a COMPARAM defined in a COMPARAM-SPEC and a new VALUE for this parameter. In addition, there is a possibility to add a description to such a redefinition by means of DESC.

Use of Base-Variant overridden response messages can only be used after base variants have been identified.

7.3.5.4 Parameter

A response or a request consists of one or more parameters (PARAMs). There are several types of parameters:

— VALUE;

— RESERVED;

— CODED-CONST;

— PHYS-CONST;

— LENGTH-KEY;

— MATCHING-REQUEST-PARAM;

— TABLE-KEY;

— TABLE-STRUCT;

— TABLE-ENTRY;

— DYNAMIC;

— SYSTEM;

— NRC-CONST.

All of them can be used directly or indirectly in a response and in a request, except that MATCHING-REQUEST-PARAM, DYNAMIC and NRC-CONST may only be used in a response.

Matching of result parameters can be done in one of the following ways:

a) MATCHING-REQUEST-PARAM (repetition of data from request);

b) CODED-CONST (matching with fixed coded-value);

c) NRC-CONST (matching with one of the fixed coded-values);

d) PHYS-CONST (matching with fixed physical value).

See Figure 54 — UML representation of parameter.

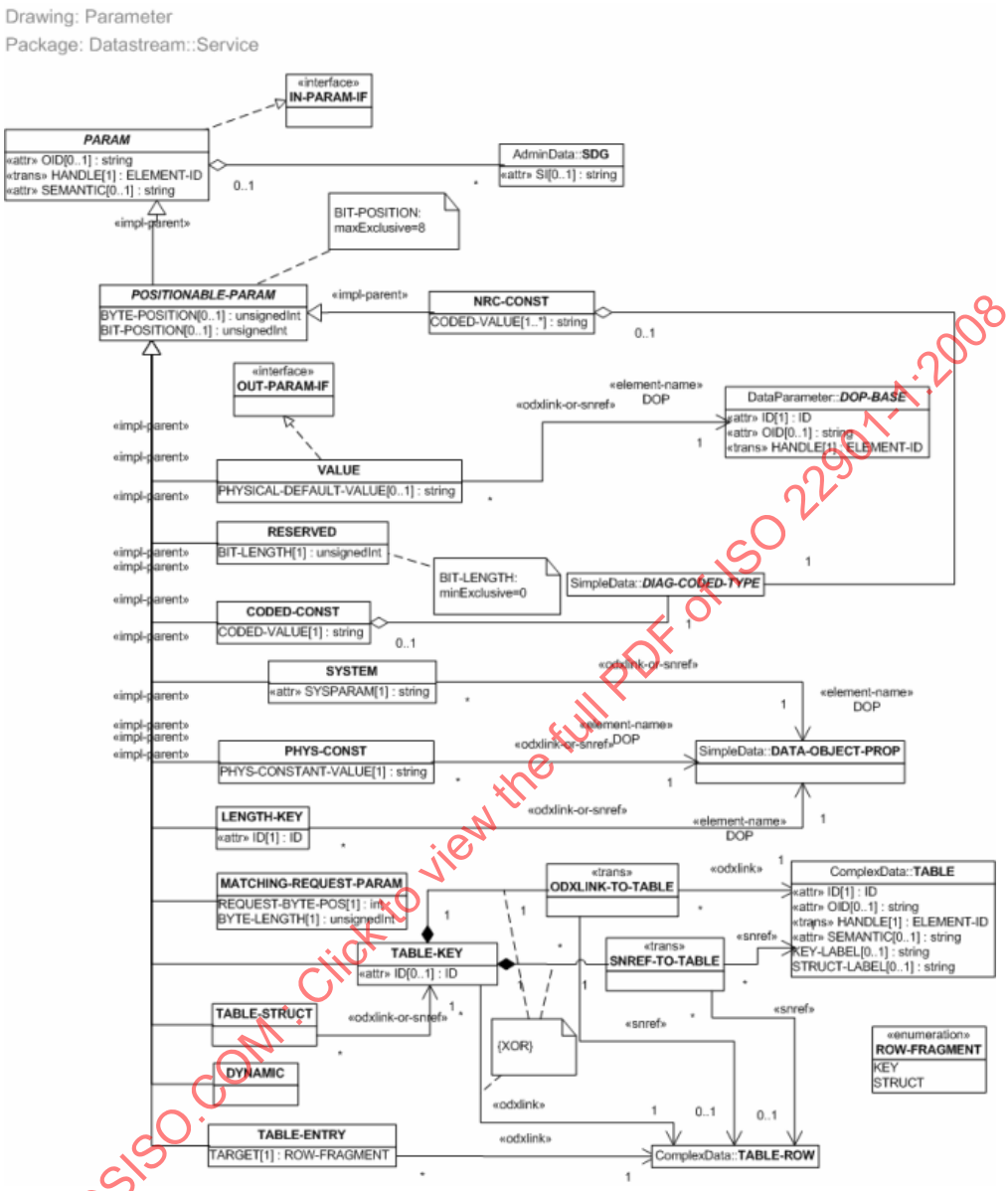


Figure 54 — UML representation of parameter

A VALUE parameter is a frequently used parameter type. It references a DOP, which converts a concrete value from coded into physical representation and vice versa. The member PHYSICAL-DEFAULT-VALUE can be used by this parameter to define the default value to be sent to the ECU in case of a request, if the user/application specifies no value. In a response message PHYSICAL-DEFAULT-VALUE is used for verification of the received value. In this case, the received coded value shall be converted into the physical representation. After that the specified and the received values can be compared.

A parameter of type RESERVED is used when the parameter should be ignored by the D-server. Such parameters are not interpreted and are not shown on the user's display. If the last or the first parameter of the PDU (request or response) should not be interpreted, it shall not be omitted. It shall be defined in the ODX instance as RESERVED in order to compute the PDU length and to compare it with the length of the received PDU in case of a response. Every bit in a request covered by a RESERVED parameter shall be filled with 0.

CODED-CONST parameters are used in a request, if the user shall not change its value. For example, the service identifier is always fixed. In this case DIAG-CODED-TYPE provides information how to put the given

value into the PDU. In a response it can be used for verification of the received value without converting it into the physical representation. If a physical value is required for this parameter, a COMPU-METHOD of CATEGORY = IDENTICAL is assumed (see 7.3.6.6.2).

PHYS-CONST is an alternative for CODED-CONST with the difference that the value is given in the physical representation. In a request it shall be converted into the coded value using the referenced DATA-OBJECT-PROP. In the response, the received coded value shall be converted into the physical representation. After that the specified and the calculated values can be compared.

NRC-CONST is used only in negative responses (see also Figure 57 — UML representation of Response). It describes a set of negative response codes as coded values. The response shall contain one of the given coded values in the encoding defined by the DIAG-CODED-TYPE. It differs from a CODED-CONST only in that it describes a set of values and shall not be used in a REQUEST or in a POS-RESPONSE. The values are used in matching a negative response. For example you might define a VALUE parameter that uses a TEXTTABLE COMPU-METHOD to map all possible negative response codes to descriptive texts. An additional NRC-CONST parameter can make the NEG-RESPONSE match if one of a specific subset of these codes is actually returned. The NRC-CONST is located at the same PDU position as the VALUE parameter. As a VALUE parameter is never used to match a RESPONSE, the parameter (and its associated TEXTTABLE) does not suffice to select the response.

In some cases the parameter's length can be defined by another one in the same message. For this purpose the type LENGTH-KEY exists. Such a parameter has the member ID and is referenced by the DIAG-CODED-TYPE of the DATA-OBJECT-PROP used by the parameter, whose length is determined by the LENGTH-KEY parameter. The LENGTH-KEY parameter and the parameter using the DATA-OBJECT-PROP that refers to this LENGTH-KEY parameter should be defined within the same wrapper (STRUCTURE, ENV-DATA, RESPONSE or REQUEST). 7.4.8 gives an example for use of LENGTH-KEY parameters.

The verification can also take place by matching the received value with a value in the request message using MATCHING-REQUEST-PARAM. REQUEST-BYTE-POS and BYTE-LENGTH are used then to define the position and the length of the value within the request PDU. The counting of REQUEST-BYTE-POS starts with zero at the first byte of the PDU. In this case the request PDU shall be stored by the D-server for the comparison. Furthermore, the received value can be compared only with byte-aligned values within the request PDU because no bit wise value extraction takes place from the request PDU. MATCHING-REQUEST-PARAM may only be used in a response.

The parameters of types TABLE-KEY, TABLE-STRUCT and TABLE-ENTRY are used for reading and writing data by data identifiers. Parameter type TABLE-ENTRY can only exist at a STRUCTURE referenced by TABLE-ROW. TABLE-KEY and TABLE-STRUCT parameters should only be used within the same PARAMS wrapper (of a STRUCTURE, an ENV-DATA, a RESPONSE or a REQUEST). For more information see 7.3.6.11.

With some protocols (e.g. ISO 14230 or ISO 14229-1) it is possible to dynamically define data records at run time, which contain values from other records identified by a local identifier, a common identifier or an address in the ECU memory, etc. A DYNAMIC parameter is used in this case to indicate the D-server that the parameter was defined dynamically. If the D-server detects a DYNAMIC parameter, it shall interpret the parameter according to the information stored in the memory. See 7.4.2 to get an example for use of DYNAMIC parameters.

The parameter of type SYSTEM is used to cause the D-server to calculate a value that depends on the run time information (e.g. the current system time) or on the special D-server information (e.g. the tester ID). The member SYSPARAM defines this value type. All SYSPARAMS listed below shall be supported by any D-server. This value set of SYSPARAM is extendable. If a user-defined SYSPARAM is used, which is not supported by the D-server, an error shall be reported. The value received by the D-server is a physical value and shall be coded using the referenced DOP. Within a response the handling of parameters of type SYSTEM is similar to the handling of parameters of type VALUE.

Table 5 — Coding of system parameter specifies the data types and coding which applies for SYSPARAMs.

Table 5 — Coding of system parameter

SYSPARAM	data type (physical)	coding	example
TIMEZONE	A_INT32	Count of minutes to UTC	+60 (Berlin)
YEAR	A_UINT32	YYYY	2008
MONTH	A_UINT32	MM	03 (march)
DAY	A_UINT32	DD	01 (first day of month)
HOUR	A_UINT32	hh	22 (10 pm)
MINUTE	A_UINT32	mm	00 (full hour)
SECOND	A_UINT32	ss	00 (full minute)
TESTERID	A_BYTEFIELD		"00F056"
USERID	A_BYTEFIELD		"043FF0"
CENTURY	A_UINT32	CC	20 (in year 2008)
WEEK	A_UINT32		04 (Fourth week of the year)
TIMESTAMP	A_BYTEFIELD	The number of elapsed milliseconds since 1970-01-01 00:00:00 GMT coded as 64bit-Integer (most significant byte first)	"0x0000011d73eb0b18" for the date 2008-11-06 22:27:43 GMT

BYTE- and BIT-POSITION specify the start position of the parameter in the PDU, where the counting starts with zero. This information is used for assembling of the request message and for extraction of the parameter from the response PDU. Sometimes, the position of a parameter cannot be determined until runtime, so the BYTE- and the BIT-POSITION shall be omitted in the ODX instance. The parameter starts then at the byte edge next to the end of the last parameter. The D-server shall pack all PARAMs without a BYTE-POSITION in the exact order in which they are specified within the ODX file. The same applies to data extraction: All unpositioned parameters are extracted in the order in which they are specified in ODX. If the unpositioned parameters follows a complex DOP (see 7.3.6), e.g. STRUCTURE, it follows behind the total size of the complex DOP (e.g. given by the "longest" parameter in any order or by BYTE-SIZE). A PARAM which is referenced by another PARAM (e.g. LENGTH-KEY and TABLE-KEY) shall be specified before the referencing PARAM (in order of appearance in the XML document) or shall be positioned.

For more information about data extraction see 7.3.6.2. BIT-POSITION shall not have a value greater than 7.

A data object of DIAG-CODED-TYPE A_BYTEFIELD, A_ASCIISTRING, A_UTF8STRING, and A_UNICODE2STRING as well as complex data objects shall cover whole bytes, i.e. the bit position shall be zero and the length shall be a multiple of 8 (16 in case of A_UNICODE2STRING). This is valid even if the bit-length of the data object is dynamically defined by MIN-MAX-LENGTH type, LEADING-LENGTH-INFO or PARAM-LENGTH-INFO-TYPE.

Information about the content of a parameter is defined by the member SEMANTIC. The values are listed in Table A.2 — SEMANTIC at PARAM. The list can also be extended if needed. This member can be used by the application to handle the parameter in a specific way or for searching and sorting purposes.

7.3.5.5 Request

The request message (REQUEST) is an integral part of each diagnostic service. It is sent by the tester to an ECU or a group of ECUs in order to get some diagnostic data or to adjust the ECU's settings. Figure 55 — UML representation of request describes the structure of such a request message. It consists of one or more request parameters (PARAMs) that define the content of the message. The ECU interprets these parameters and replies to the tester by sending a response message, which consists of response parameters in an analogous manner.

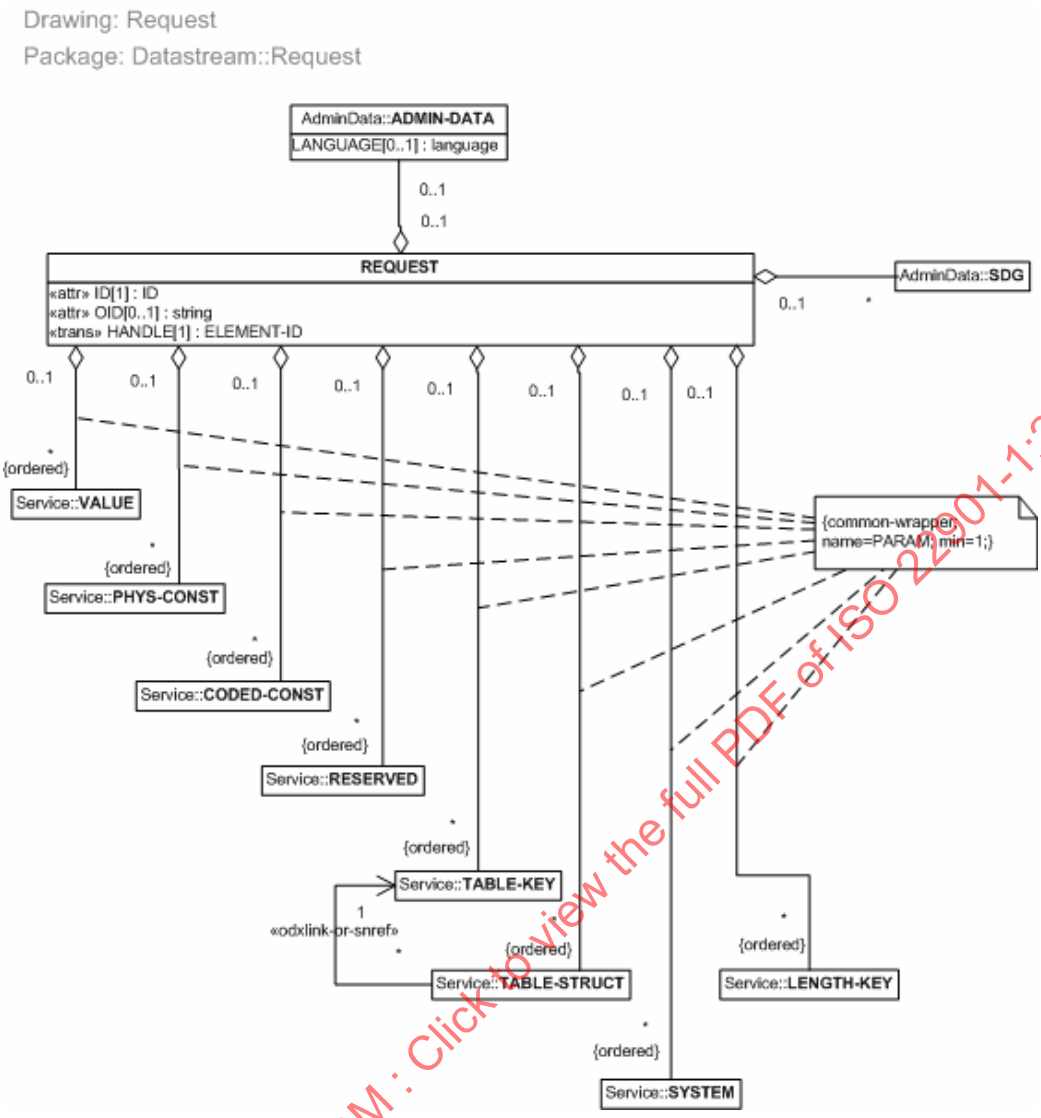


Figure 55 — UML representation of request[4)]

Each request owns the member ID that unambiguously identifies the request. One or more PARAM objects give the content of the request message. The PARAMs shall not overlap in the request byte stream. In general, all bytes and bits of the request byte stream should be covered by at least one of the request parameters. Be aware that the payload of the request is initialized by bit value 0. In consequence, all gaps of the request that are not explicitly defined by parameters remain unchanged (with the bit value 0).

Generally speaking, the request message is assembled as follows. Firstly, each parameter is converted to its coded representation if needed. The coded values are then put into the PDU at the specified BYTE- and BIT-POSITION. Bytes are counted from zero starting with the first byte of the message, i.e. byte 0 is the first byte after the message header. Bits are counted from zero starting with the least significant bit (LSB). If the BIT-POSITION is omitted, the coding starts at the bit position 0 of the specified byte.

EXAMPLE ReadDataByAddress request message defined in Keyword Protocol 2000:

```
<REQUEST ID = "REQ_ReadDataByAddress">
    <SHORT-NAME>REQ_ReadDataByAddress</SHORT-NAME>
    <LONG-NAME>Read Data By Address Request</LONG-NAME>
```

4) The order of PARAMs inside the wrapper is globally significant regardless of the PARAM class.

```
    <PARAMS>
        <PARAM SEMANTIC = "SERVICE-ID" xsi:type = "CODED-CONST">
            <SHORT-NAME>ServiceId</SHORT-NAME>
            <BYTE-POSITION>0</BYTE-POSITION>
            <CODED-VALUE>35</CODED-VALUE>
            <DIAG-CODED-TYPE BASE-DATA-TYPE = "A_UINT32" xsi:type = "STANDARD-LENGTH-TYPE">
                <BIT-LENGTH>8</BIT-LENGTH>
            </DIAG-CODED-TYPE>
        </PARAM>
        <PARAM xsi:type = "VALUE">
            <SHORT-NAME>Address</SHORT-NAME>
            <LONG-NAME>Address in the ECU memory</LONG-NAME>
            <BYTE-POSITION>1</BYTE-POSITION>
            <BIT-POSITION>0</BIT-POSITION>
            <DOP-REF ID-REF = "DOP_3ByteIdentical"/>
        </PARAM>
        <PARAM xsi:type = "VALUE">
            <SHORT-NAME>Size</SHORT-NAME>
            <LONG-NAME>Size of the value</LONG-NAME>
            <BYTE-POSITION>4</BYTE-POSITION>
            <PHYSICAL-DEFAULT-VALUE>1</PHYSICAL-DEFAULT-VALUE>
            <DOP-REF ID-REF = "DOP_1ByteIdentical"/>
        </PARAM>
    </PARAMS>
</REQUEST>
```

In this example, the first PARAM (the service ID) has a constant internal value and cannot be modified by the user. The user/application shall specify the second PARAM because neither a constant nor a default value is given, while the third PARAM can be specified, otherwise the default value “1” is used. The second parameter (“Address”) references the DOP with ID = “DOP_3ByteIdentical”, while the third one (“Size”) references the DOP with ID=“DOP_1ByteIdentical”. Both DOPs convert the physical values given by the user/application or the default value into their coded representation.

7.3.5.6 Response

A response message (RESPONSE) is the ECU's part of a diagnostic service. It is returned by the ECU in response to a tester's request message. Figure 56 — UML representation of response describes the class RESPONSE of such a response message. Three types of responses are defined in ODX: positive, negative and global negative responses. Normally a positive response message (POS-RESPONSE) is sent by an ECU. If an exceptional situation occurs within the ECU, a negative response message can be sent back.

Negative responses might be defined specific to a service or to a diagnostic layer instance. In the first case the negative response shall be defined by referencing a NEG-RESPONSE from the service via NEG-RESPONSE-REF. In the second case, a global negative response (GLOBAL-NEG-RESPONSE) shall be defined. This response shall automatically be used by a D-server if no negative response, referenced by the actual service, matches.

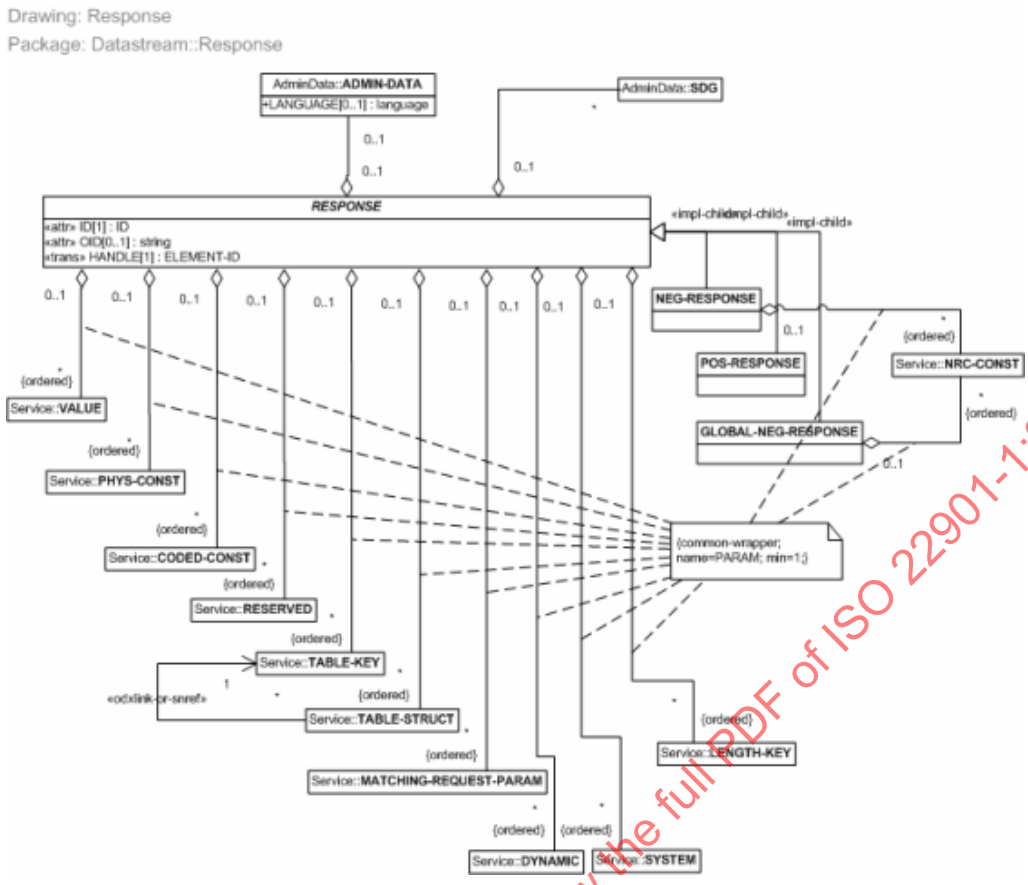


Figure 56 — UML representation of response[5)]

All types of responses have the same structure. A response is identified by the ID attribute and consists of several response parameters (PARAMs) that represent all data to be extracted from the response message.

The interpretation of the response consists of the interpretation of the parameters defined by the RESPONSE object. See 7.3.5.4 for more information about the parameters.

EXAMPLE ReadDataByAddress response message defined in Keyword Protocol 2000:

```
<POS-RESPONSE ID = "RESP_ReadDataByAddress">
    <SHORT-NAME>RESP_ReadDataByAddress</SHORT-NAME>
    <LONG-NAME>Read Data By Address Response</LONG-NAME>
    <PARAMS>
        <PARAM SEMANTIC = "SERVICE-ID" xsi:type = "CODED-CONST">
            <SHORT-NAME>RDBA</SHORT-NAME>
            <LONG-NAME>read Data By Address</LONG-NAME>
            <BYTE-POSITION>0</BYTE-POSITION>
            <CODED-VALUE>99</CODED-VALUE>
            <DIAG-CODED-TYPE BASE-DATA-TYPE = "A_UINT32" xsi:type = "STANDARD-LENGTH-TYPE">
                <BIT-LENGTH>8</BIT-LENGTH>
            </DIAG-CODED-TYPE>
        </PARAM>
        <PARAM SEMANTIC = "DATA" xsi:type = "VALUE">
            <SHORT-NAME>ReadData</SHORT-NAME>
            <LONG-NAME>Read data</LONG-NAME>
```

5) The order of PARAMs inside the wrapper is globally significant regardless of the PARAM class.

```
            <BYTE-POSITION>1</BYTE-POSITION>
            <DOP-REF ID-REF = "DOP-ByteArray"/>
        </PARAM>
    </PARAMS>
</POS-RESPONSE>
```

In the above example, the first PARAM is the service identifier. Its value shall match the value given by CODED-VALUE and DIAG-CODED-TYPE. The second PARAM references a DOP with ID= “DOP-ByteArray”, which interprets the received byte array.

7.3.5.7 Single ECU job

Jobs (SINGLE-ECU-JOBs) are a second kind of diagnostic communication primitive (DIAG-COMM) next to services (DIAG-SERVICE, see above). They are seen as complex services that are not provided natively by the ECU, but shall be implemented manually on top of the ECU’s native services. Jobs solely interact with the D-server. They have no means to communicate directly with e.g. a diagnostic application or a user interface for interaction. Thus, they are macros for recurring complex tasks (e.g. reading of diagnostic trouble codes and the corresponding environment data, seed&key algorithms or ECU memory programming). Jobs are not to be used to define complete diagnostic test sequences.

The D-server supports two kinds of jobs, which are modelled separately within ODX. One kind of job contains calls to services of one and only one dedicated ECU (SINGLE-ECU-JOB). These jobs are explained in this subclause and illustrated in Figure 57 — UML representation of job (SINGLE-ECU-JOB). The service calls within the other kind (MULTIPLE-ECU-JOB, see 7.3.11) span multiple ECUs and are explained in a separate subclause, because they cannot be assigned to one DIAG-LAYER alone.

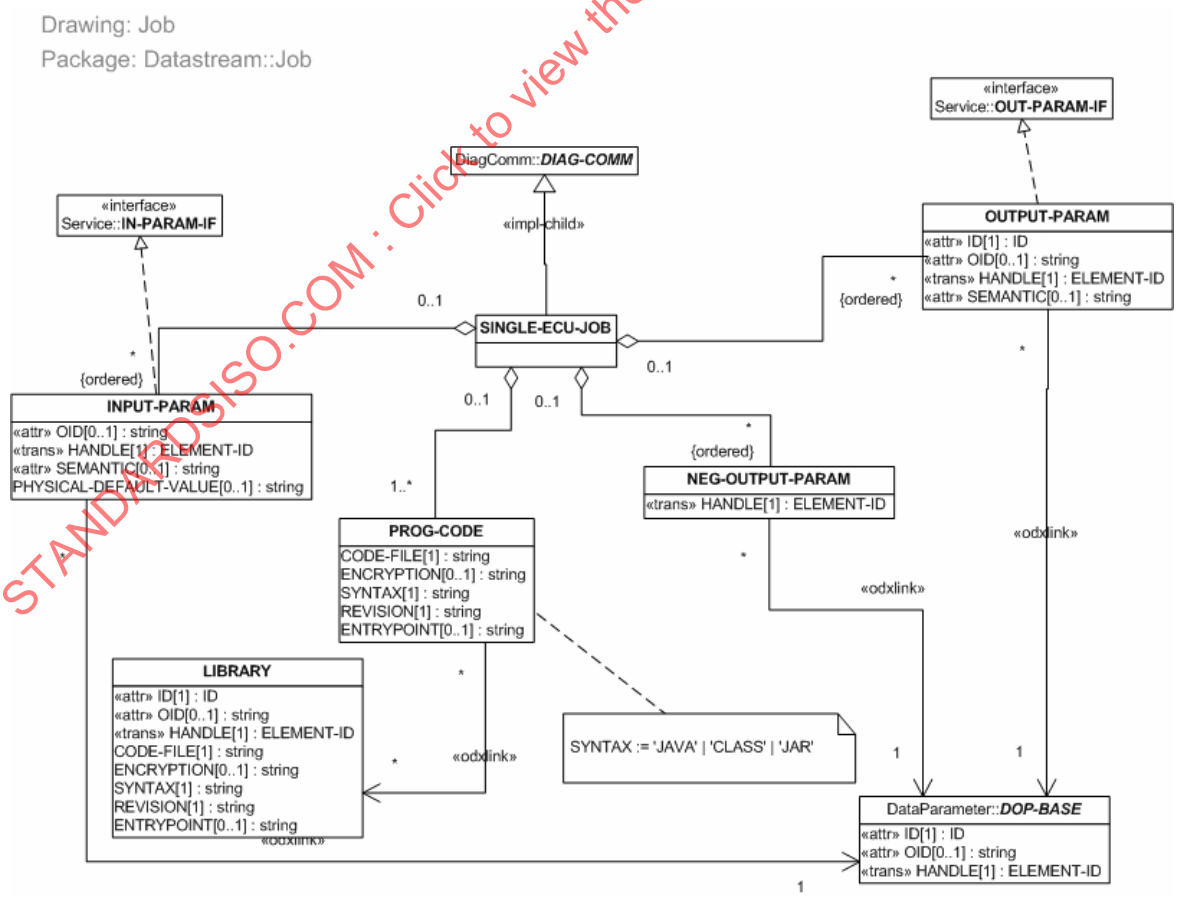


Figure 57 — UML representation of job (SINGLE-ECU-JOB)

A SINGLE-ECU-JOB is derived from DIAG-COMM, the generic class for all diagnostic communication primitives. In analogy to a DIAG-SERVICE, a SINGLE-ECU-JOB has input and output data, which are modelled as a set of input parameters (INPUT-PARAM) and output parameters (OUTPUT-PARAM). In case of errors, a set of negative output parameters (NEG-OUTPUT-PARAM) may be returned by the job instead of a set of regular output parameters.

Job INPUT- and OUTPUT PARAMs have references to a DOP-BASE. However DOPs are applied differently from services: The conversion method is ignored, meaning an input parameter is passed to the job as-is and always as a physical value. PHYSICAL-DEFAULT-VALUE shall only be used, if INPUT-PARAM references a DATA-OBJECT-PROP. Output parameters are returned by the job as physical values. The physical values returned by the job shall comply with the PHYSICAL-TYPE specification of the associated DOP-BASE.

Special cases are input parameters that have a TEXTTABLE defined as COMPU-METHOD (see 7.3.6.6.7). Here, the valid texts of the TEXTTABLE can be listed within an application. Upon selection, the corresponding text (the physical value) is passed to the job. Hence, the COMPU-METHOD is only used for comfort reasons not for actual conversion of the input parameter from a physical into a coded type. Consequently, the DIAG-CODED-TYPE of the DOP associated by a job parameter (regardless of whether it is an INPUT- or OUTPUT-PARAM) is ignored.

The reasons for this solution are as follows:

a) the job parameter specification can reuse DOPs of contained services in cases where the service parameters are part of the job's input or output as well;

b) the D-server does not support conversion of input- and output parameters of jobs.

The SEMANTIC attribute's value domain of the INPUT-PARAM and OUTPUT-PARAM is defined in analogy to that of PARAMs.

The core aspect of a job is the program code (PROG-CODE). A PROG-CODE instance contains all data needed to reference a specific piece of code that is to be executed as the job. Within the PROG-CODE instance, a reference to the file containing the code is specified (CODE-FILE). The syntax of the code is given in SYNTAX (either JAVA for Java source code, or CLASS for Java byte code or JAR for .jar-files) and the code revision number in REVISION. As an option, an encryption algorithm may be specified in ENCRYPTION as well as an ENTRYPOINT into a binary code or .jar-file.

It is recommended that within one job[6] the contained PROG-CODE elements are disjoint regarding their value for the SYNTAX attribute. JAVA, CLASS and JAR shall exclude each other.

For the naming of a java job used in PROG-CODE there are some guidelines to be observed. To assure the unambiguousness of the name of the used java classes the namespace of java package will be used. The package structures are supplier specific and may start for example with company's internet domain.

The following rules exist for the different SYNTAX values:

— in the case of SYNTAX = "JAVA", the CODE-FILE value shall be specified using the slash character as package separator (e.g. "com/carmanufacturer/jobs/EcuJob.java");

— in the case of SYNTAX = "CLASS", the CODE-FILE shall be specified using the dot character as package separator (e.g. "com. carmanufacturer.jobs.EcuJob");

— in the case of SYNTAX = "JAR", the CODE-FILE shall contain the name of the Jar-file and the ENTRY-POINT specifies the name of the job in dot notation (e.g. CODE-FILE = "jobs.jar" and ENTRY-POINT = "com.carmanufacturer.jobs.EcuJob").

6) SINGLE-ECU-JOB or MULTIPLE-ECU-JOB.

However, if jobs in another syntax (e.g. a Windows DLL) need to be supported, a corresponding value for SYNTAX and ENTRY-POINT can be used. A regular D-server implementation is not able to execute such a proprietary job. Similar information applies to the ENCRYPTION field.

EXAMPLE 1 In the following, an example for a SINGLE-ECU-JOB instance is given. The data is meant for a job able to return a list of all diagnostic trouble codes for a KW2000 ECU. An input parameter provides the option of retrieving the diagnostic trouble codes together with the environment data. Thus, the job contains calls to two KW2000 services of the ECU. The output parameter has a reference to a DOP. In this case, it is a COMPLEX-DOP describing the data structure for the returned data of the ECU's fault memory.

```
<SINGLE-ECU-JOB ID = "JOB_Read_DTC_With_Env_Data" SEMANTIC = "FAULTREAD">
    <SHORT-NAME>JOB_Read_DTC_With_Env_Data</SHORT-NAME>
    <AUDIENCE/>
    <PROG-CODES>
        <PROG-CODE>
            <CODE-FILE>SEJ_ReadDTCWithEnvData.java</CODE-FILE>
            <SYNTAX>JAVA</SYNTAX>
            <REVISION>1.1.2</REVISION>
        </PROG-CODE>
    </PROG-CODES>
    <INPUT-PARAMS>
        <INPUT-PARAM>
            <SHORT-NAME>WITH_ENV_DATA</SHORT-NAME>
            <PHYSICAL-DEFAULT-VALUE>true</PHYSICAL-DEFAULT-VALUE>
            <DOP-BASE-REF ID-REF = "ID_73474"/>
        </INPUT-PARAM>
    </INPUT-PARAMS>
    <OUTPUT-PARAMS>
        <OUTPUT-PARAM ID = "ID_73472">
            <SHORT-NAME>DTC_LIST</SHORT-NAME>
            <DOP-BASE-REF ID-REF = "ID_73473"/>
        </OUTPUT-PARAM>
    </OUTPUT-PARAMS>
</SINGLE-ECU-JOB>
```

EXAMPLE 2 In a second example, the data of a job is shown that calls a single diagnostic service. This service is the ReadDataByAddress service already used as an example in 7.3.5.3 onwards (there it is named ReadMemoryByAddress). As an essential difference between the job's parameters and the parameters of the service's request that is called in its body, constant parameters do not have to be included. Constant parameters of the service will be filled with their constant value automatically when the job calls the service via the D-server. The other (variable) parameters reappear in the job's parameter profile and the service's DOPs are reused for the job's parameters.

```
<SINGLE-ECU-JOB ID = "JOB_Read_Data_By_Address" SEMANTIC = "COMMUNICATION">
    <SHORT-NAME>JOB_Read_Data_By_Address</SHORT-NAME>
    <AUDIENCE/>
    <PROG-CODES>
        <PROG-CODE>
            <CODE-FILE>SEJ_ReadDataByAddress</CODE-FILE>
            <SYNTAX>CLASS</SYNTAX>
            <REVISION>2.3.0</REVISION>
        </PROG-CODE>
    </PROG-CODES>
    <INPUT-PARAMS>
        <INPUT-PARAM>
            <SHORT-NAME>Address</SHORT-NAME>
            <DOP-BASE-REF ID-REF = "DOP_3ByteIdentical"/>
        </INPUT-PARAM>
        <INPUT-PARAM>
            <SHORT-NAME>Size</SHORT-NAME>
```

```
            <DOP-BASE-REF ID-REF = "DOP_1ByteIdentical"/>
        </INPUT-PARAM>
    </INPUT-PARAMS>
    <OUTPUT-PARAMS>
        <OUTPUT-PARAM ID = "ID_734712">
            <SHORT-NAME>Returned_Value</SHORT-NAME>
            <DOP-BASE-REF ID-REF = "DOP_ByteArray"/>
        </OUTPUT-PARAM>
    </OUTPUT-PARAMS>
</SINGLE-ECU-JOB>
```

7.3.5.8 Library

Libraries (LIBRARY) are used to specify code that is not directly executed, but is used by executable code defined within a PROG-CODE instance. For example, a PROG-CODE element containing a Java job might reference a LIBRARY element that describes an additional JAR-file, which is included by import statement in the job's Java code. LIBRARYs are attached to DIAG-LAYERs to allow for multiple references without redundancies.

As described in the above paragraph, LIBRARY elements can provide additional information to a D-server about constraints when executing a PROG-CODE instance at runtime. For example, LIBRARY elements can define a set of program libraries to be used by the PROG-CODE, and define the exact revisions and code-files of these libraries that are to be used. This enables a PROG-CODE to make explicit which libraries and which versions of these libraries are needed for a successful execution. However, it is still up to the PROG-CODE provider and to the data and application designers whether code files described by a LIBRARY are actually part of an ODX data distribution, or are deployed in a different way. For example, an ODX package might include a JAR-file that is described by a LIBRARY element, or LIBRARY elements might point to specific versions of system libraries that a PROG-CODE code files depends on for successful execution. A D-server can use the information provided through LIBRARY definitions to enforce execution constraints on PROG-CODE code files. For example, it might refuse to execute a PROG-CODE if not all of its LIBRARY prerequisites are met on a system. Also, LIBRARY elements can provide information to a D-server where to look for specific library files that need to be made accessible to a PROG-CODE during execution.

Figure 58 — LIBRARY collection at DIAG-LAYERA can be referenced by PROG-CODE instances via odxlink, and can be used within SINGLE-ECU-JOBs, MULTIPLE-ECU-JOBs or from COMPU-METHODs of type COMPUCODE.

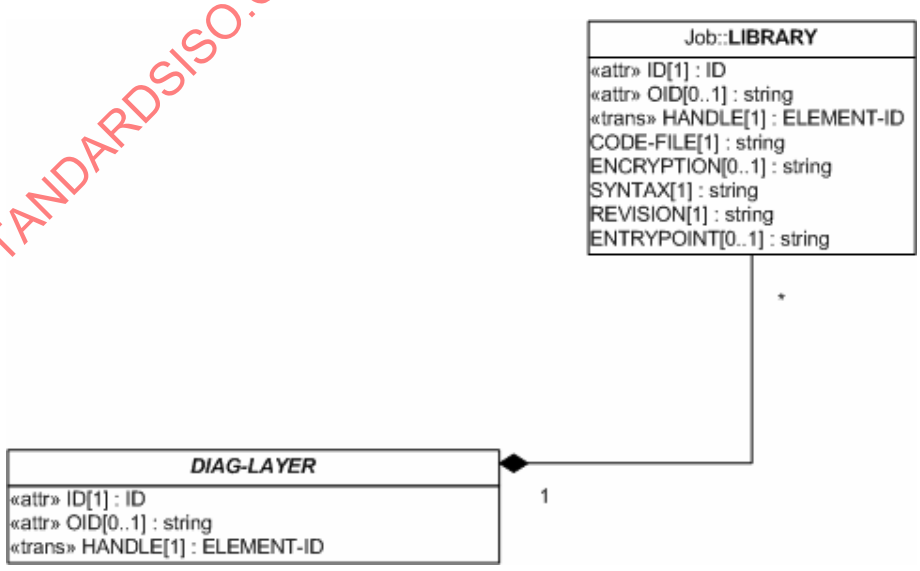


Figure 58 — LIBRARY collection at DIAG-LAYER

A LIBRARY instance covers similar properties and attributes like PROG-CODE (see also 7.3.5.7 and 7.3.6.6.9):

— CODE-FILE: references the file containing the code;

— ENCRYPTION: optional, specifies an encryption algorithm;

— SYNTAX: additional to the formats that apply as for PROG-CODE (“JAVA”, “CLASS” and “JAR”), other suffixes (e.g. .so or .dll) are valid for a LIBRARY; for all Java data formats like “JAVA”, “CLASS” and “JAR” the same processing mechanism applies as for PROG-CODE, all non-Java data formats shall only be provided to an application without any further processing by the D-server;

— REVISION: specifies the code file revision number;

— ENTRYPOINT: optional, semantic depends on the kind of code file that is described by the LIBRARY.

See Figure 59 — LIBRARY referenced by PROG-CODE.

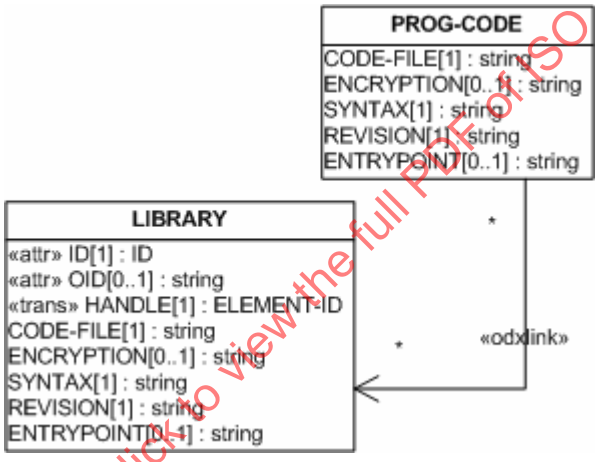


Figure 59 — LIBRARY referenced by PROG-CODE

EXAMPLE A LIBRARY element defined in XML, referencing a java archive (JAR) file to be used for security access tasks:

```
<LIBRARY>
  <SHORT-NAME>Security_Access</SHORT-NAME>
  <LONG-NAME>Security access for telematics ECUs.</LONG-NAME>
  <DESC>
    <p>Contains classes with methods for calculation of security keys for telematic ECUs.</p>
  </DESC>
  <CODE-FILE>SecurityAccess.jar</CODE-FILE>
  <ENCRYPTION>myEncryptionMethod</ENCRYPTION>
  <SYNTAX>JAR</SYNTAX>
  <REVISION>1.0.1</REVISION>
  <ENTRYPOINT>SecurityAccessNavigation</ENTRYPOINT>
</LIBRARY>
```

7.3.6 Data parameter

7.3.6.1 Overview

Data parameters play an important role within diagnostic communication. Diagnostic services send request data to the ECU and get response data back from it. The task of the diagnostic D-server is not only to perform the communication between tester and ECU but also to transform the data from the physical representation format to the coded byte format that is actually transported via the communication layer and physical layer. The same procedure holds for the reverse way where the coded byte format is converted to the physical representation. The properties of data parameters describe this process of data extraction, interpretation and conversion. The major objects are the DATA-OBJECT-PROPs (or DOPs), which consist of data types for coded and physical format, constraints, computation methods and reference physical units.

Figure 60 — Data parameters and diagnostic communication (ISO 14230 example) illustrates above described behaviour.

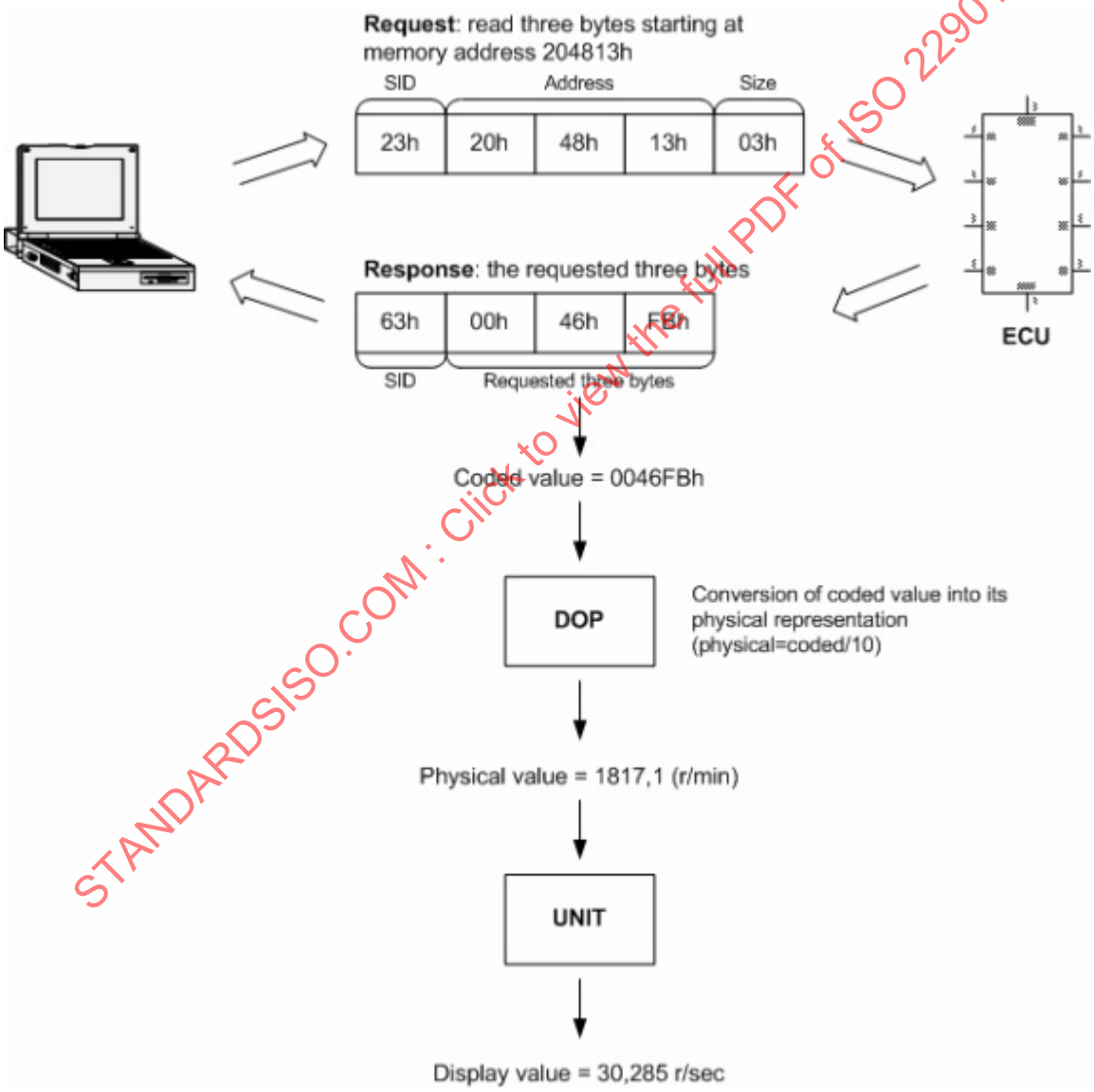


Figure 60 — Data parameters and diagnostic communication (ISO 14230 example)

Figure 61 — UML representation of data parameter packages illustrates data parameters, which are a substantial part of the whole ODX data model with a lot of classes and attributes.

The following packages keep the model of data parameters:

a) SimpleData consisting of simple data structures and using the packages:

   — conversion for the description of transforming data from the physical presentation to the internal format and vice versa: there are 8 different computation method types defined;

   — units containing the modelling of physical units especially the methods to convert units to each other with consideration of specific unit groups;

b) ComplexData defines composite and diagnostic specific data structures with complex substructures that end up in simple data structures.

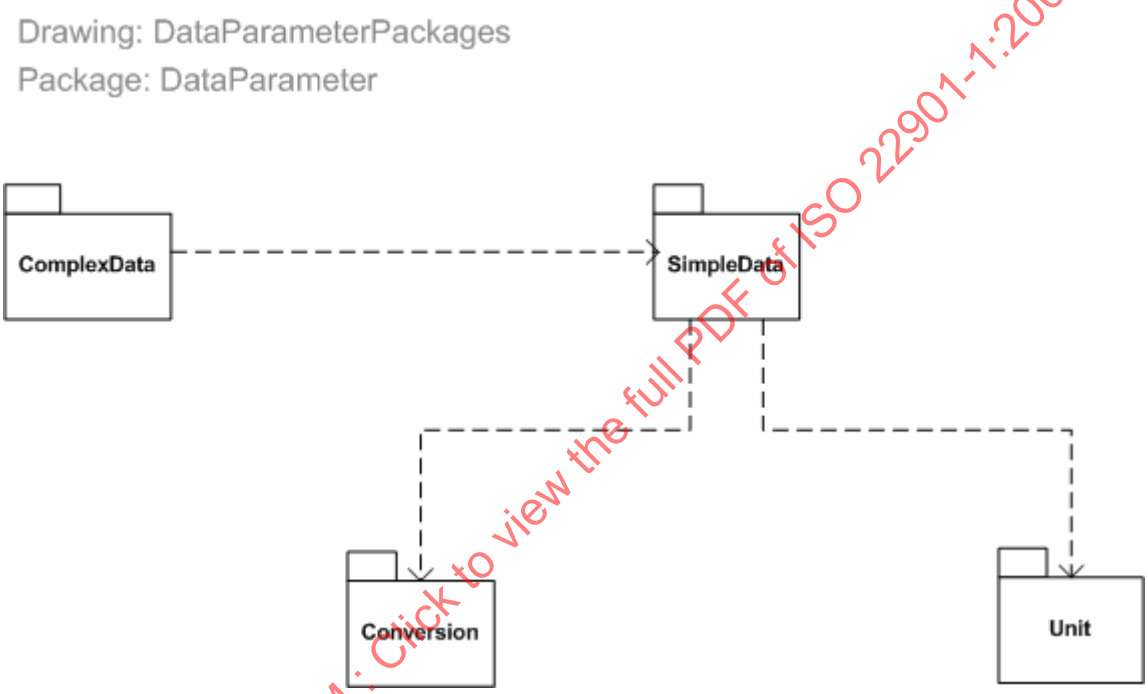


Figure 61 — UML representation of data parameter packages

Figure 62 — UML representation of data parameter overview illustrates the DIAG-LAYER which can include a data dictionary, which is the location for all diagnostic data parameters. The data dictionary is modelled as class DIAG-DATA-DICTIONARY-SPEC that consists mainly of an aggregation of subclasses defining the different types of data parameters besides administration data and unit definitions.

The following classes of diagnostic data objects are used for request and/or response parameters:

— DATA-OBJECT-PROP (SimpleData): represents a single value, consists of data types for coded and physical format, constraints, computation methods, and references physical units;

— DTC-DOP (SimpleData): represents a single diagnostic trouble code (DTC), consists of data types for coded and physical format, computation methods, and collection of possible DTC values;

— STRUCTURE (ComplexData): container for data parameters of any class;

— MUX (ComplexData): multiplexed data parameters;

— ENV-DATA (ComplexData): container for environment data parameters belonging to one DTC;

— ENV-DATA-DESC (ComplexData): multiplexed environment data parameters data belonging to DTC information;

— FIELD (STATIC-FIELD, DYNAMIC-ENDMARKER-FIELD, DYNAMIC-LENGTH-FIELD, END-OF-PDU-FIELD) (ComplexData): container for data parameters that are repeated within the PDU;

— TABLE: multiplexed data parameters with table structure.

All used data objects are identified by an ID and have a HANDLE (of type ELEMENT-ID). The general class DOP-BASE is specialized by its subclasses, which are the only ones who can be instantiated (<<impl-child>> inheritance). The most frequently used objects are DATA-OBJECT-PROPs. Parameter using the classes DATA-OBJECT-PROPs and DTC-DOPs are the final classes within an interpretation chain. Exclusively via these objects D-server is able to deliver values of measured ECU variables or other ECU information to the tester. Data objects using objects of the other classes COMPLEX-DOPs and TABLE describe the packaging rules of special responses from the ECU like repeated structures, multiplex structures or responses with dynamic length. But the description of each COMPLEX-DOP finally ends up with references to a parameter using DATA-OBJECT-PROPs or DTC-DOPs.

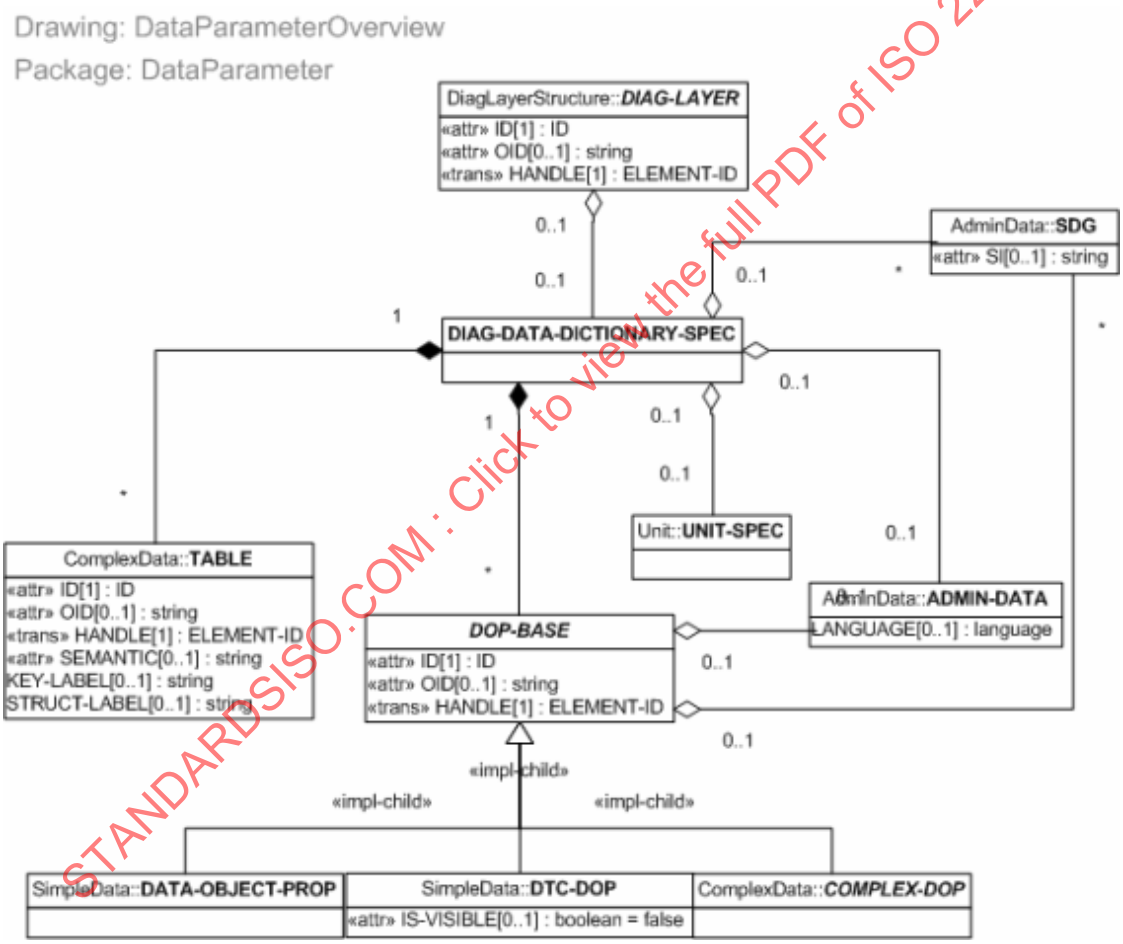


Figure 62 — UML representation of data parameter overview

The Conversion package includes all types of computational methods (COMPU-METHODs), which can be used for conversion between physical and coded representation of values. There are eight different types of COMPU-METHODs that are described in 7.3.6.6. Where possible, COMPU-METHODs use a generic approach to define the mathematical formula used for conversion. The integration of external computer programs used for conversion is also specified in this subclause.

The package Unit offers the possibility to define units, unit groups and methods for converting units. The class UNIT defines the physical units (e.g. m, km, km/h, s) with their physical dimensions. UNITs are referenced by DATA-OBJECT-PROPs and indicate to the D-server that the values from the ECU shall be delivered to the tester inclusive a physical unit (e.g. 100 km/h, 10,3467 s). There is also a concept to define equivalence classes of units used in different countries. For example “mph” is used in the USA where “km/h” is used in Europe. Though “m/s” also represents speed, it is normally used for other values and thus will not belong to this class.

Table 6 — Positioning of PARAM and DOP types shows the rules for the use of PARAMs and DOPs. Generally speaking, FIELDs shall not be contained in FIELDs.

Table 6 — Positioning of PARAM and DOP types

Context	Class	Allowed PARAM types	Allowed Reference Targets
DIAG-SERVICE	REQUEST	VALUE	DATA-OBJECT-PROP, DTC-DOP, STRUCTURE, END-OF-PDU-FIELD
		CODED-CONST, RESERVED	-
		PHYS-CONST, SYSTEM, LENGTH-KEY	DATA-OBJECT-PROP
		TABLE-KEY	TABLE, TABLE-ROW
		TABLE-STRUCT	-
DIAG-SERVICE	RESPONSE	VALUE	DOP-BASE
		CODED-CONST, RESERVED, MATCHING-REQUEST-PARAM, DYNAMIC, NRC-CONST	-
		PHYS-CONST, SYSTEM, LENGTH-KEY	DATA-OBJECT-PROP
		TABLE-KEY	TABLE, TABLE-ROW
		TABLE-STRUCT	-
REQUEST	STRUCTURE	See "Allowed PARAM types" and corresponding "Allowed Reference Targets" at class REQUEST	
RESPONSE	STRUCTURE, ENV-DATA	See "Allowed PARAM types" and corresponding "Allowed Reference Targets" at class RESPONSE	
TABLE-ROW	STRUCTURE	VALUE[7)]	DOP-BASE
		CODED-CONST, RESERVED	-
		PHYS-CONST, SYSTEM	DATA-OBJECT-PROP
		TABLE-KEY	TABLE, TABLE-ROW
		TABLE-STRUCT	-
		TABLE-ENTRY	TABLE-ROW

7) If the TABLE-ROW is used inside a REQUEST, then the allowed reference target are restricted to DATA-OBJECT-PROP, DTC-DOP, STRUCTURE, and END-OF-PDU-FIELD.

7.3.6.2 Simple data - DATA-OBJECT-PROP

7.3.6.2.1 Figure 63 — UML representation of DATA-OBJECT-PROP illustrates the request message which is sent by the tester to the ECU and the response message sent by the ECU to the tester. The response message can be interpreted by request parameters and response parameters, respectively, using simple and complex DOPs as described in 7.3.6.1. Using the object of class DATA-OBJECT-PROP in a request, it describes how to transform the physical representation value into its internal form using a computational method (COMPU-METHOD) and how to encode and to pack the internal value as a single data item into the byte stream of the request message. The class DATA-OBJECT-PROP describes how to extract and decode a single data item from the byte stream of the response message and how to transform the internal value into its physical representation using a computational method (COMPU-METHOD).

After the conversion a UNIT-REF can be used to reference a UNIT in order to provide additional visualization information for the value.

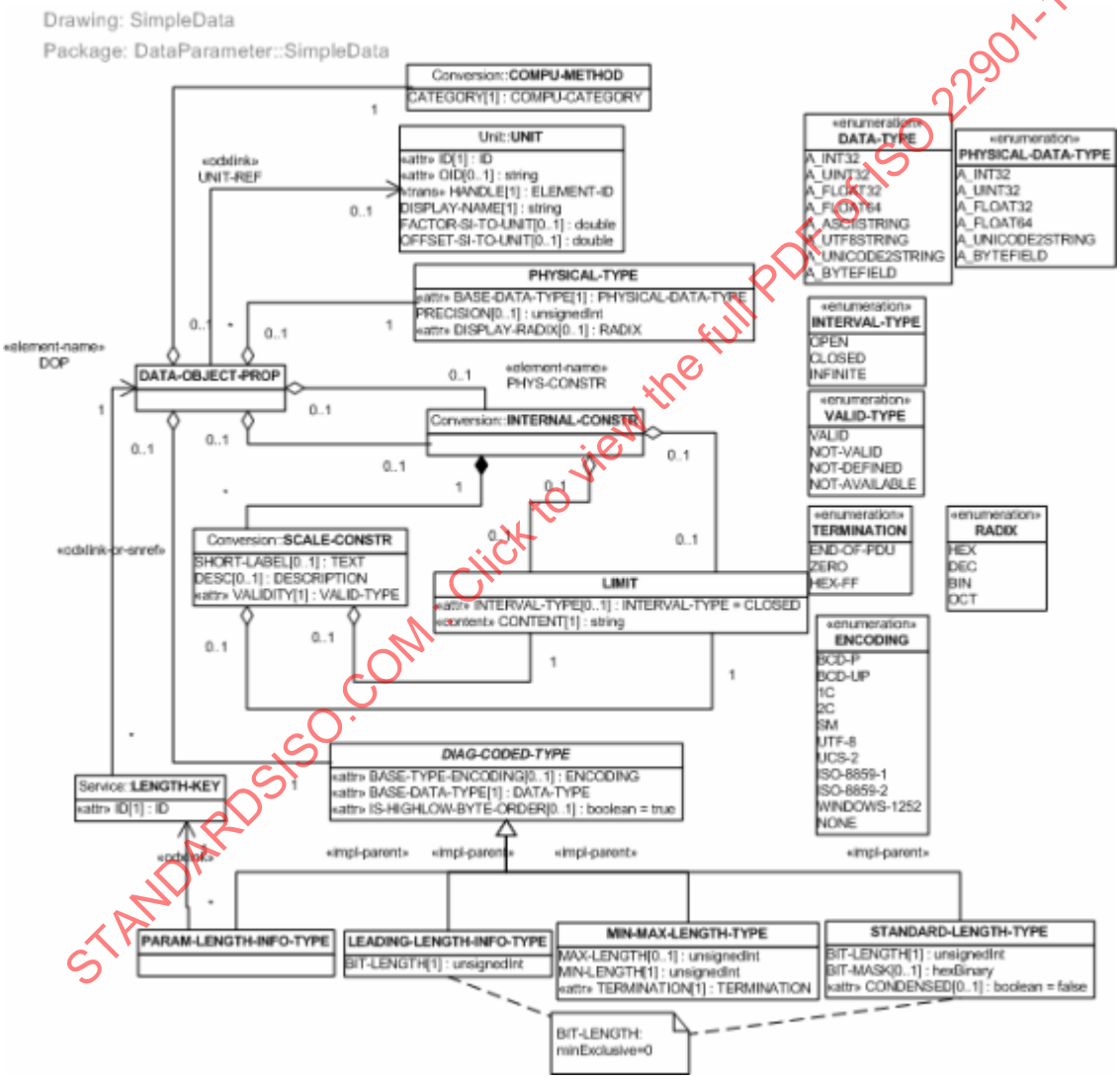


Figure 63 — UML representation of DATA-OBJECT-PROP

A DATA-OBJECT-PROP is identified by its SHORT-NAME and ID. It consists of a COMPU-METHOD, a DIAG-CODED-TYPE and a PHYSICAL-TYPE. As an option, INTERNAL-CONSTR and PHYS-CONSTR can be given for a DATA-OBJECT-PROP and a UNIT may be referenced.

The implicit valid range of a data type is defined by BASE-DATA-TYPE, restricted by ENCODING and size of the parameter, which is defined either by the number of 1 in a condensed BIT-MASK or the BIT-LENGTH. The

explicit valid range is defined by UPPER-LIMIT and LOWER-LIMIT minus all SCALE-CONSTR with VALIDITY!=VALID. If UPPER-LIMIT or LOWER-LIMIT is missing, the explicit valid range is not restricted in that direction. The valid range is defined by the intersection of the implicit and the explicit valid range.

SCALE-CONSTR are not allowed to overlap each other, and shall be defined within INTERNAL-CONSTR limits.

DIAG-CODED-TYPE is responsible for extraction of the coded value out of the byte stream and for packing of a value into the stream (see subclauses “Data extraction” and “Data packing” below). The coded value type inside the message is described by the member BASE-DATA-TYPE of the DIAG-CODED-TYPE. The extracted and decoded data item is stored within the D-server as an internal value. The internal value type is connected with the coded value type. Because the decoded value of type A_ASCIISTRING cannot be longer stored as an ASCII string – the character set becomes larger during decoding – the internal value type of any string type is A_UNICODE2STRING (see also 7.3.6.6.2 and 7.3.6.6.9).

Table 7 — Predefined ODX base data types specifies the ODX BASE-DATA-TYPE.

Table 7 — Predefined ODX base data types

BASE-DATA-TYPE	Description
A_INT32	32 bit signed integer (range: -2^31 .. 2^31-1)
A_UINT32	32 bit unsigned integer (range: 0 .. 2^32-1)
A_FLOAT32	32 bit float (IEEE 754 single precision)
A_FLOAT64	64 bit float (IEEE 754 double precision)
A_ASCIISTRING	ISO-LATIN-1 (ISO/IEC 8859-1) coded character field with maximum length: 2^31-1 bytes
A_UTF8STRING	UTF-8 coded character field with maximum length: 2^31-1 bytes
A_UNICODE2STRING	UNICODE-2 (ISO/IEC 10646 UCS-2) coded character field with maximum length: 2^31-1 characters
A_BYTEFIELD	Byte field with maximum length: 2^32-1

The internal value and its type are used as input for the computational method (COMPU-METHOD) to transform the value into its physical representation (physical value). Consequential the COMPU-METHOD within a DOP shall support the internal and physical value type of it. The detailed description of COMPU-METHODs occurs in 7.3.6.6. There are four types of the class DIAG-CODED-TYPE, as described below.

a) **LEADING-LENGTH-INFO-TYPE**: the length of the parameter is to be extracted from the PDU. BIT-LENGTH specifies the bit count at the beginning of this parameter and shall be greater than zero. These bits contain the length of the parameter in bytes. The bits of length information are not part of the coded value. That means the data extraction of the coded value starts at the byte edge to this length information. The attribute IS-HIGHLOW-BYTE-ORDER of the DIAG-CODED-TYPE is applied to the extraction of the coded value (except A_BYTEFIELD, A_ASCIISTRING, and A_UTF8STRING) and the leading length parameter.

b) **PARAM-LENGTH-INFO-TYPE**: the length of the parameter is defined by another parameter referenced by this kind of DIAG-CODED-TYPE. The physical value of the referenced parameter defines the length of the referencing parameter in bits.

c) **MIN-MAX-LENGTH-TYPE**: the minimum and the maximum length of a parameter in a message are specified by MIN-LENGTH and MAX-LENGTH. Both values (MIN-LENGTH and MAX-LENGTH) are to be given in number of bytes. Consequently, the coded parameter shall at minimum have the length of MIN-LENGTH and at maximum the length of MAX-LENGTH bytes, independent of TERMINATION. TERMINATION specifies a possible pre-mature end of a parameter in the message, that is, the end of the parameter is reached before MAX-LENGTH bytes have been read. The following values for TERMINATION are possible:

— ZERO: the byte stream to be extracted is terminated by the first of the following conditions:

  — finding termination character 0x00 (A_ASCIISTRING and A_UTF8STRING) or 0x0000 (A_UNICODE2STRING) after MIN-LENGTH bytes;

  — reaching of MAX-LENGTH bytes;

  — reaching end of PDU.

— HEX-FF: the byte stream to be extracted is terminated by the first of the following conditions:

  — finding termination character 0xFF (A_ASCIISTRING and A_UTF8STRING) or 0xFFFF (A_UNICODE2STRING) after MIN-LENGTH bytes;

  — reaching of MAX-LENGTH bytes;

  — reaching end of PDU.

— END-OF-PDU: the byte stream to be extracted is terminated by the first of the following conditions:

  — reaching of MAX-LENGTH bytes;

  — reaching end of PDU.

— If the end of the PDU is reached before MIN-LENGTH bytes have been read, an error shall be returned at runtime.

If the actual payload of the parameter is shorter than MAX-LENGTH, the termination value shall be coded into the PDU in case of TERMINATION ZERO or HEX-FF. If the actual payload length of the parameter is equal to the MAX-LENGTH, the termination value is omitted (it is not coded into the request and the coded value of the corresponding response parameter does not contain the termination byte). Only the payload will be encoded and converted. The termination value is not used for conversion, but it is considered if the next parameter has an undefined BYTE-POSITION, i.e. the next parameter follows the termination value.

In the case of A_UNICODE2STRING it is necessary to test the termination value 0x0000 and 0xFFFF as character codes rather the sequence of 2 bytes with values 0x00 or 0xFF. For example, the unicode string “Āa” has the byte sequence 0x01 0x00 0x00 0x61 assuming a byte order HIGH-LOW. The both values 0x00 in the middle may not be interpreted as a ZERO termination.

The termination characters 0xFF and 0xFFFF are not valid UTF-8 and UCS-2 characters, respectively. Therefore, they are suitable as termination characters.

In the case of A_UNICODE2STRING, the values of MIN-LENGTH, MAX-LENGTH shall be even.

In the case of A_UNICODE2STRING, the length of actual payload shall be even. Otherwise, the D-server shall report an error.

d) **STANDARD-LENGTH-TYPE**: the parameter always occupies the specified length of BIT-LENGTH bits inside the message. If a provided value is shorter than the specified length, an error shall be signalled. The value of BIT-LENGTH shall be greater than zero.

As an option, an attribute BIT-MASK can be specified, if not all BIT-LENGTH bits inside the request message should be modified by the parameter value or some bits inside the response message should be masked out. The length of the bit mask shall be equal to the BIT-LENGTH. If the parameter does not occupy a continuous bit stream inside the message, the optional attribute CONDENSED = true can be combined with BIT-MASK.

In this case, the unset bits inside the bit mask do not only specify the unaffected bits inside the message, but additional the bits to remove from the coded value while the extraction step by shifting the higher bits to the lower bits. The following example compares the application of BIT-MASK with and without the CONDENSED attribute:

Response message:	0101**11100111**10100101**10100101**1010		
Assume BIT-LENGTH = 24:			
Bit mask:	**11111111**0000000**11111111**	=	0xFF00FF
Apply binary AND:	111001110000000010100101	=	0xE700A5
Extracted coded value in case of			
CONDENSED = false:	111001110000000010100101	=	0xE700A5
CONDENSED = true:	11100111 10100101	=	0xE7A5

7.3.6.2.2 Regardless of the type of length specification, the following additional restrictions shall be considered depending on the BASE-DATA-TYPE:

— A_INT32 and A_UINT32: length shall be between 1 bit and 32 bit;

— A_FLOAT32: length shall be exactly 32 bit;

— A_FLOAT64: length shall be exactly 64 bit;

— A_BYTEFIELD, A_ASCIISTRING, and A_UTF8STRING: length shall be a multiple of 8 bits (i.e. whole bytes);

— A_UNICODE2STRING: length shall be a multiple of 16 bits (i.e. whole words);

— the D-server shall report an error in the case of the violation of these restrictions;

— LEADING-LENGTH-INFO-TYPE and MIN-MAX-LENGTH-TYPE are allowed for the internal data types A_BYTEFIELD, A_ASCIISTRING, A_UNICODE2STRING, and A_UTF8STRING only.

Examples for type MIN-MAX-LENGTH-TYPE of class DIAG-CODED-TYPE are given below.

EXAMPLE 1 MIN-MAX-LENGTH-TYPE: Ignore termination byte before MIN-LENGTH.

TERMINATION bytes within the first MIN-LENGTH bytes are inserted into the payload.

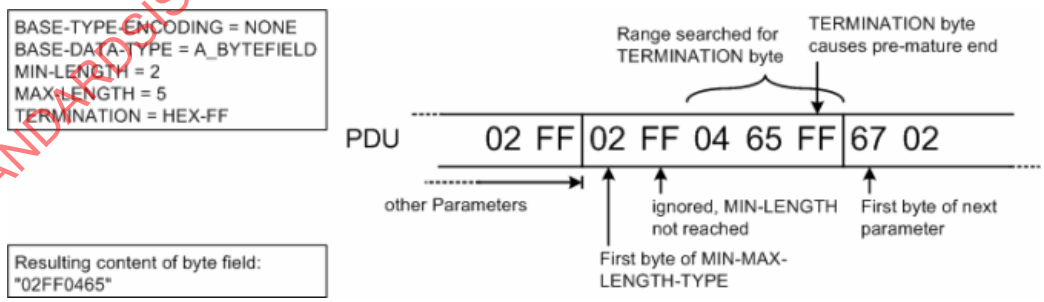


Figure 64 — MIN-MAX-LENGTH-TYPE: Ignore termination byte before MIN-LENGTH

EXAMPLE 2 MIN-MAX-LENGTH-TYPE: End at MAX-LENGTH.

After reading MAX-LENGTH bytes, no additional TERMINATION byte is expected or read.

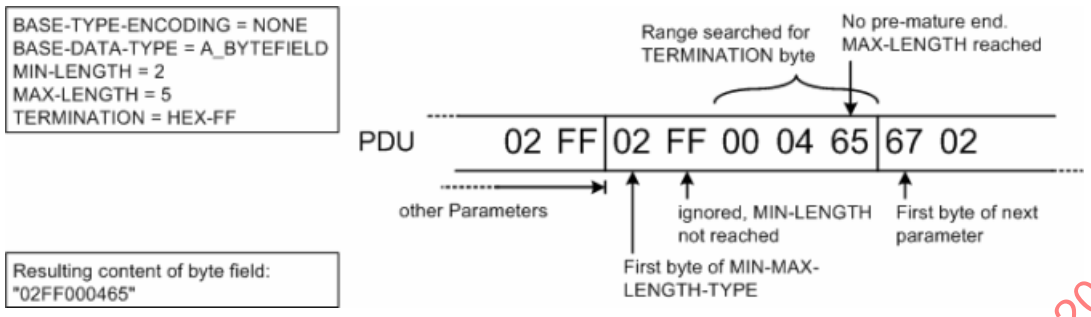


Figure 65 — MIN-MAX-LENGTH-TYPE: End at MAX-LENGTH

EXAMPLE 3 MIN-MAX-LENGTH-TYPE: End before MAX-LENGTH.

The next parameter starts immediately after the TERMINATION byte. The TERMINATION byte is not part of the payload.

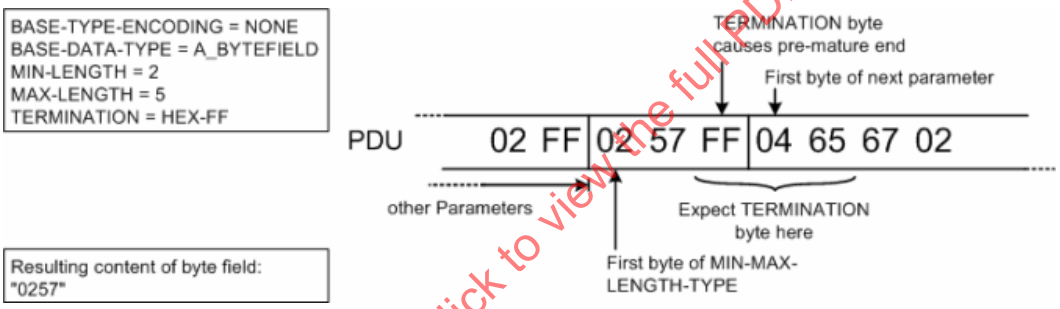


Figure 66 — MIN-MAX-LENGTH-TYPE: End before MAX-LENGTH

EXAMPLE 4 'EndOfPDU': Termination by END-OF-PDU?

Everything up to the end of the PDU is added to the payload.

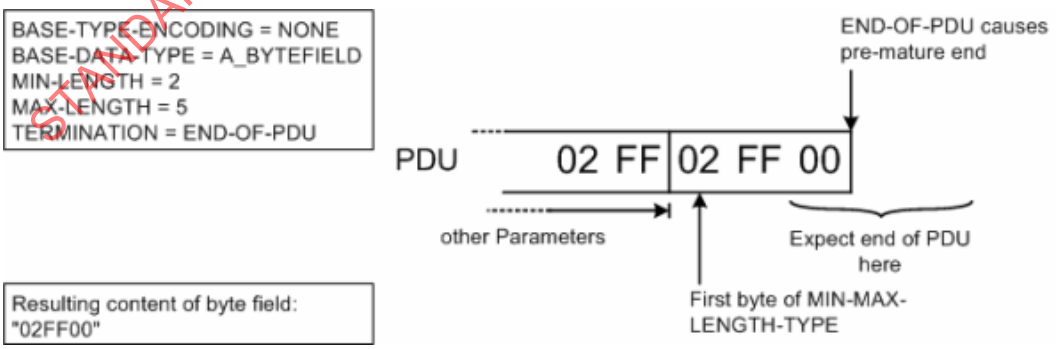


Figure 67 — MIN-MAX-LENGTH-TYPE: End at end of PDU

EXAMPLE 5 MIN-MAX-LENGTH-TYPE: Error reaching end of PDU before MIN-LENGTH.

At least MIN-LENGTH bytes shall be read before termination or an error occurs.

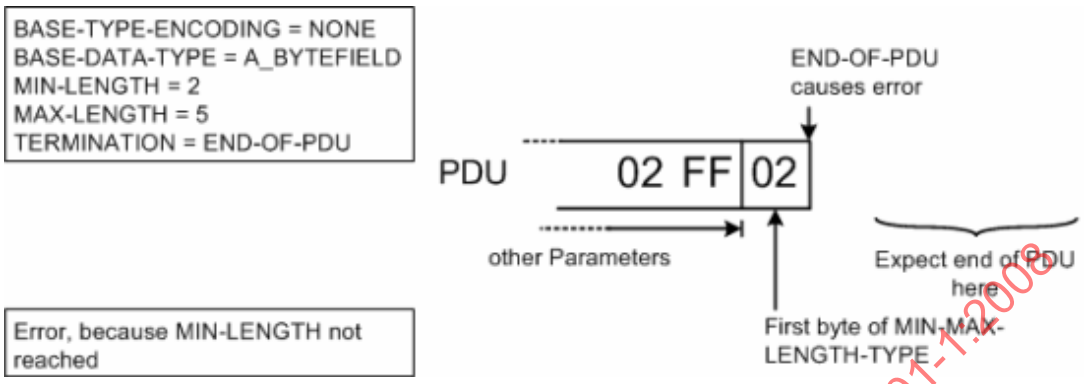


Figure 68 — MIN-MAX-LENGTH-TYPE: Error reaching end of PDU before MIN-LENGTH

7.3.6.2.3 The member ENCODING at DIAG-CODED-TYPE gives the method used for encoding of the value in the PDU. It has the following value domain:

— BCD-P,

— BCD-UP,

— 1C,

— 2C,

— SM,

— UTF-8,

— UCS-2,

— ISO-8859-1,

— ISO-8859-2 and

— WINDOWS-1252.

Table 8 — BASE-DATA-TYPE encodings provides an overview over possible combinations of BASE-DATA-TYPE and BASE-TYPE-ENCODING values.

Table 8 — BASE-DATA-TYPE encodings

BASE-DATA-TYPE	Possible BASE-TYPE-ENCODINGs
A_INT32	SM (sign-magnitude)[8], 1C (one's complement), 2C (two's complement)(=default encoding)[9]
A_UINT32	BCD-P(packed BCD)[10], BCD-UP[11], NONE (=default encoding)
A_FLOAT32	IEEE 754 (=default encoding)
A_FLOAT64	IEEE 754 (=default encoding)
A_ASCIISTRING	ISO-8859-1 (=default coding), ISO-8859-2, WINDOWS-1252
A_UTF8STRING	UTF-8 (=default encoding)
A_UNICODE2STRING	UCS-2 (=default encoding)
A_BYTEFIELD	BCD-P(packed BCD)[12) 13], BCD-UP[14], NONE (=default encoding)

If the D-server is not able to encode or decode the value according to the ENCODING specified, it shall provide an appropriate error message.

Binary-coded decimal encoding of BASE-DATA-TYPE = A_BYTEFIELD.

Binary-coded decimal (BCD) is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations.

In case of a BCD encoded BYTEFIELD in the PDU the D-server does not need to convert the coded into the internal value, it only needs to check whether the value contains forbidden digits like 0xA to 0xF or not. If a forbidden digit is used, the D-server shall provide an appropriate error message. In a common use case there will be no further calculation from the internal into the physical value, the byte sequence can be passed as identical value directly to the application. Contrary to data types A_UINT32 or A_INT32 it is possible to cover values of more than 4 byte with data type A_BYTEFIELD.

Figure 69 — Example for binary-coded decimal encoding of BASE-DATA-TYPE = A_BYTEFIELD illustrates an example which shows a BCD encoded PDU containing a parameter with BASE-DATA-TYPE=A_BYTEFIELD and how it is converted from its coded over the internal into the physical value.

8) The encoding sign-magnitude means the first bit represents the sign (e.g. "1" equals "-"). In the following bits the magnitude of the value is coded.

9) If the BASE-TYPE-ENCODING is not defined explicitly, the "default encoding" shall be applied.

10) A BCD digit is packed in 4 bits (nibble) (e.g. 47 = 47h).

11) A BCD digit is mapped to one byte (e.g. 17 = 0107h).

12) An use case is, for example, to transmit a 5 byte field of fixed length containing only numbers (e.g. a part number) and display the value as it has been transmitted.

13) The D-server will actually not encode or decode the internal value during packing and extracting, but only check that each byte of the byte field is interpretable as a byte with BCD encoding.

14) The D-server will actually not encode or decode the internal value during packing and extracting, but only check that each byte of the byte field is interpretable as a byte with BCD-UP encoding.

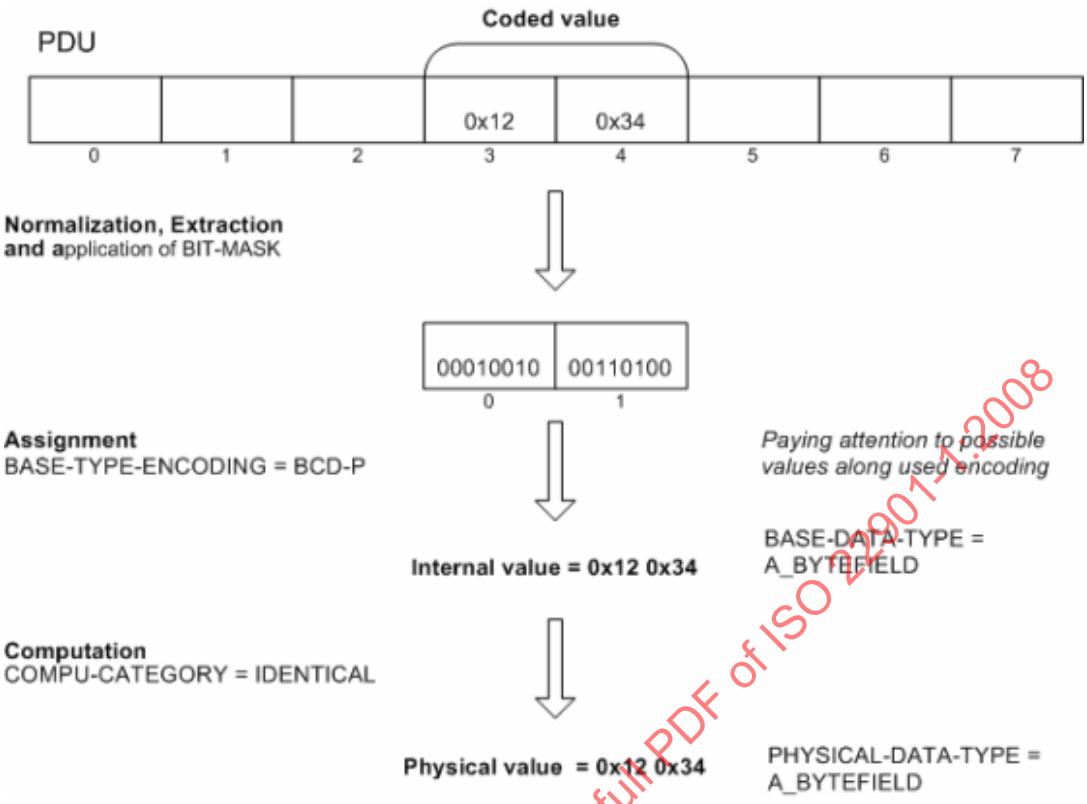


Figure 69 — Example for binary-coded decimal encoding of BASE-DATA-TYPE = A_BYTEFIELD

PHYSICAL-TYPE specifies the physical representation of the value. The member BASE-DATA-TYPE gives the type and encoding of the physical value again. Physical values always have the default encoding of their respective type as defined in Table 8 — BASE-DATA-TYPE encodings. The optional member DISPLAY-RADIX is used for visualization purposes, i.e. it specifies whether the physical value is to be displayed in a hexadecimal, decimal, binary or octal representation. The corresponding attribute values are HEX, DEC, BIN or OCT. It is only applicable to the physical data type A_UINT32. There is also a possibility to specify the number of digits to be displayed after the decimal point by the member PRECISION. It can be given for A_FLOAT32 and A_FLOAT64 data types and is used for the visualisation of the physical value at the tester side. For example, to display the value 1.234 the PRECISION is to be set to 3 while 0.00010 requires a PRECISION of 5.

The calculated physical value can be directly displayed on the tester. Usually units are used to augment this displayed value with additional information like m/s or litre. Also the conversion of values from one unit to another like “km/h” to “m/s” is possible. The declaration of the unit to be used is given via the reference UNIT-REF. See 7.3.6.7 for more information about units.

The value coding of numbers in the ODX data is subject to restrictions described by specification “XML schema Part 2”. For example, “100” and “1.0E2” are two different literals which both denote the same float value. A value of type A_BYTEFIELD shall be defined hexadecimal in ODX data with an even number of digits. Value of type A_UINT32 and A_INT32 shall be defined decimal in ODX data.

The validity of the internal value can be restricted to a given interval via the INTERNAL-CONSTR and its members LOWER-LIMIT and UPPER-LIMIT. In addition, SCALE-CONSTRS can be used to define valid, non-valid, non-defined or non-available sub-intervals within the interval spanned by INTERNAL-CONSTR. The limits of the sub-intervals are defined with LOWER-LIMIT and UPPER-LIMIT in the same way as in INTERNAL-CONSTR. The attribute VALIDITY of each SCALE-CONSTR can take the values VALID, NOT-VALID, NOT-DEFINED or NOT-AVAILABLE, respectively (Table 9 — Values of VALIDITY). As an option, a SHORT-LABEL can be used to add an identifier to a SCALE-CONSTR.

The following restrictions regarding SCALE-CONSTR apply:

a) each range of SCALE-CONSTR shall not overlap any other range: two closed range ends with the same limit are prohibited;

b) each range of SCALE-CONSTR shall fit into the INTERNAL-CONSTR range;

c) LOWER-LIMIT/CONTENT shall be less or equal to UPPER-LIMIT/CONTENT: if the CONTENT values are equal, the INTERVAL-TYPE of both shall be set to CLOSED.

Table 9 — Values of VALIDITY

VALIDITY	Description
VALID	Valid area. Current value is within a valid range and can be presented to user.
NOT-VALID	Invalid area. The ECU cannot process the requested data.
NOT-DEFINED	Indicates an area, which is marked in a specification (e.g. as reserved). Shall usually not be set by the ECU but is used by a tester to verify correct ECU behaviour. Also prevents use of areas during editing process.
NOT-AVAILABLE	Currently invalid area. The value usually is presented by the ECU but can currently not be performed due to e.g. initialisation or temporary problems. This behaviour appears during runtime and cannot be handled while data is edited.

7.3.6.2.4 Some information like “value is not valid” might not always be sufficient. Therefore the SHORT-LABEL can be used to provide further information like error messages. The attribute INTERVAL-TYPE of LOWER-LIMIT and UPPER-LIMIT defines the type of the interval at the corresponding end. Possible values are OPEN if the edge value is not to be included, CLOSED if it is to be included, or INFINITE if the value defined by LOWER-LIMIT is -∞ or if it is +∞ in the case of UPPER-LIMIT. The value specified by the limit is ignored if INTERVAL-TYPE=“INFINITE”. In case of no limit given, the interval is not constrained in that direction, i.e. the limit is implicitly set to -∞ if LOWER-LIMIT is missing or to +∞ if no UPPER-LIMIT is given.

IMPORTANT — The LOWER-LIMIT and UPPER-LIMIT of INTERNAL-CONSTR define the valid interval of values. All values outside of this interval are not valid and thus are not to be processed by the ECU. Therefore, the declaration of an INTERNAL-CONSTR is equivalent to the declaration of a SCALE-CONSTR with VALIDITY=“VALID”.[15] If neither LOWER-LIMIT nor UPPER-LIMIT is defined in INTERNAL-CONSTR directly, at least one SCALE-CONSTR with VALIDITY other than “VALID” shall exist.

The implicit valid range of a data type is defined by BASE-DATA-TYPE, restricted by ENCODING and size of parameter which is defined either by the number of 1 in a condensed BIT-MASK or the BIT-LENGTH.

The explicit valid range is defined by UPPER-LIMIT and LOWER-LIMIT minus all SCALE-CONSTR with VALIDITY != VALID. If UPPER-LIMIT or LOWER-LIMIT is missing the explicit valid range is not restricted in that direction.

The valid range is defined by the intersection of the implicit and the explicit valid range.

INTERNAL-CONSTR can only be used in conjunction with a DATA-OBJECT-PROP which has a numerical type of the internal value.

15) In ODX there is no need for SCALE-CONSTRS with VALIDITY="VALID" because the valid interval is defined by the INTERNAL-CONSTR. If a SCALE-CONSTR with VALIDITY="VALID" is found within the ODX data (e.g. as the result of a data import from another use case) then it should be ignored during the processing of the INTERNAL-CONSTR.

a) If the DATA-OBJECT-PROP is used for calculation of physical value from the internal one, the internal value is matched against the valid range directly:

   1) match internal value against the valid range defined above of the DATA-OBJECT-PROP;

   2) if match was successful (i.e. the internal value is valid), calculate the physical value from the internal value with the COMPU-METHOD of the DATA-OBJECT-PROP:

      i) if the value is outside the valid range, the D-server reports an error message;

      ii) if the value falls into an explicitly specified invalid scale, the information of this SCALE-CONSTR is used to generate the error message.

b) If the DATA-OBJECT-PROP is used to transform the physical into the internal value, the physical value given by the user is first converted into the internal representation and is then matched against the valid range:

   1) calculate the internal value from the physical value with the COMPU-METHOD of the DATA-OBJECT-PROP;

   2) match calculated internal value against the valid range defined above of the DATA-OBJECT-PROP:

      i) if the value is outside the valid range, the D-server does not accept the physical value as an input value and reports an error message;

      ii) if the value falls into an explicitly specified invalid scale, the information of this SCALE-CONSTR is used to generate the error message.

For the examples below, the following specification of DIAG-CODED-TYPE is assumed:

```
<DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_INT32"
  CONDENSED="true" BASE-TYPE-ENCODING="SM">
  <BIT-LENGTH>12</BIT-LENGTH>
  <BIT-MASK>E07</BIT-MASK>
</DIAG-CODED-TYPE>
```

The parameter occupies inside the message a length of 12 bit, but the condensed bit mask restricts the length of the signed integer value to actually 6 bit. Because it is encoded as SM, the implicit valid range is [-31, +31].

Figure 70 — Use of constraints: no explicit restrictions (no INTERNAL-CONSTR) shows an example.

The explicit valid range is (-∞, +∞) and, therefore, the intersection between the explicit and implicit valid ranges is [-31, +31].

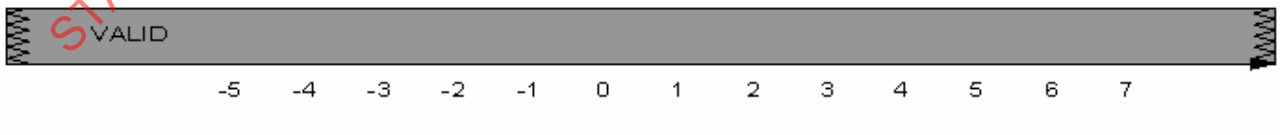


Figure 70 — Use of constraints: no explicit restrictions (no INTERNAL-CONSTR)

Figure 71 — Use of constraints: one valid interval shoes an example.

```
<INTERNAL-CONSTR>
    <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">-3</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE = "CLOSED">5</UPPER-LIMIT>
</INTERNAL-CONSTR>
```

The explicit valid range is [-3, +5] and, therefore, the intersection between the explicit and implicit valid ranges is [-3, +5].

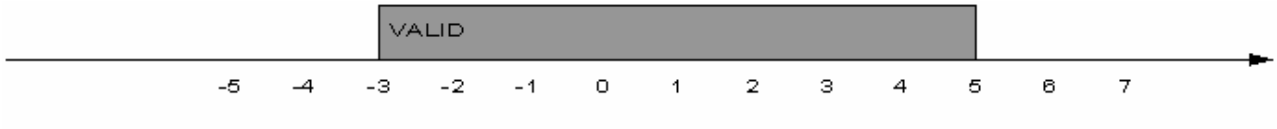


Figure 71 — Use of constraints: one valid interval

Figure 72 — Use of constraints: no INTERNAL-CONSTR limits shows an example.

```
<INTERNAL-CONSTR>
    <SCALE-CONSTRS>
        <SCALE-CONSTR VALIDITY = "NOT-DEFINED">
            <LOWER-LIMIT INTERVAL-TYPE = "OPEN">4</LOWER-LIMIT>
            <UPPER-LIMIT INTERVAL-TYPE = "CLOSED">5</UPPER-LIMIT>
        </SCALE-CONSTR>
        <SCALE-CONSTR VALIDITY = "NOT-VALID">
            <LOWER-LIMIT INTERVAL-TYPE = "OPEN">5</LOWER-LIMIT>
            <UPPER-LIMIT INTERVAL-TYPE = "CLOSED">6</UPPER-LIMIT>
        </SCALE-CONSTR>
    </SCALE-CONSTRS>
</INTERNAL-CONSTR>
```

With no limits for the INTERNAL-CONSTR, the explicit validity of the internal value is defined in the interval (-∞, +∞). Within this interval the interval (4, 5] is declared via SCALE-CONSTRS as non-defined and the interval (5, 6] as not valid, which is illustrated in the following figure.

The explicit valid ranges are (-∞, +4] and (+6, +∞), therefore, the intersections between the explicit and implicit valid ranges are [-31, +4] and (+6, +31]. Because the internal type is an integer type, the second range is equivalent to [+7, +31].

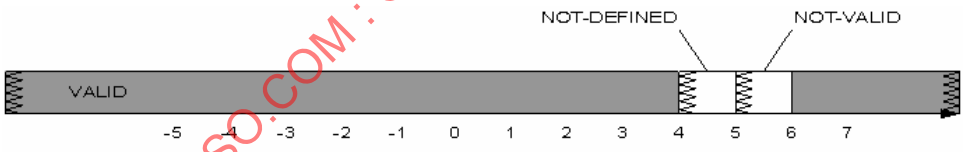


Figure 72 — Use of constraints: no INTERNAL-CONSTR limits

EXAMPLE 1 Use of constraints: Clipping:

```
<INTERNAL-CONSTR>
    <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">0</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE = "INFINITE"/>
    <SCALE-CONSTRS>
        <SCALE-CONSTR VALIDITY = "NOT-DEFINED">
            <LOWER-LIMIT INTERVAL-TYPE = "OPEN">4</LOWER-LIMIT>
            <UPPER-LIMIT INTERVAL-TYPE = "CLOSED">5</UPPER-LIMIT>
        </SCALE-CONSTR>
        <SCALE-CONSTR VALIDITY = "NOT-AVAILABLE">
            <LOWER-LIMIT INTERVAL-TYPE = "OPEN">5</LOWER-LIMIT>
            <UPPER-LIMIT INTERVAL-TYPE = "CLOSED">6</UPPER-LIMIT>
        </SCALE-CONSTR>
    </SCALE-CONSTRS>
</INTERNAL-CONSTR>
```

This XML code defines the explicit validity of the internal value in the interval [0, +∞) by defining limits for the INTERNAL-CONSTR. Within this interval the interval (4, 5] is declared via SCALE-CONSTRS as non-defined and the interval (5, 6] as non-available, which is illustrated in the following figure.

The explicit valid ranges are [0, +4] and (+6, +∞), therefore, the intersections between the explicit and implicit valid ranges are [0, +4] and (+6, +31]. Because the internal type is an integer type, the second ranges is equivalent to [+7, +31].

Figure 73 — Use of constraints: clipping shows an example.

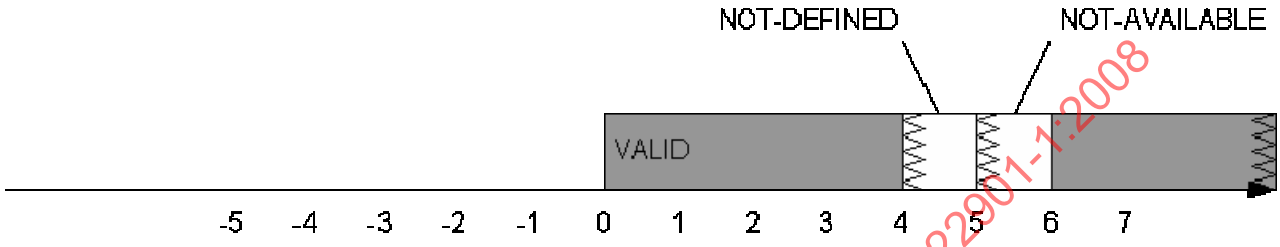


Figure 73 — Use of constraints: clipping

In a similar way, it is possible to specify constraints for the physical value inside PHYS-CONSTR, if the physical type of the DATA-OBJECT-PROP is a numerical type. In this case, the implicit valid range is not restricted by the size or encoding of the data type, only by the general restriction as given in Table 7 — Predefined ODX base data types. If the DATA-OBJECT-PROP is used for calculation of physical value from the internal one, the physical value is matched against the valid physical range after applying the conversion method. If the DATA-OBJECT-PROP is used to transform the physical into the internal value, the physical value given by the user is first matched against the valid physical range.

EXAMPLE 2 Extracting one bit from the PDU and converting it into a string via COMPU-METHOD of type “TEXTTABLE”:

```
<DATA-OBJECT-PROP ID="DOP_DTC_WLCS">
  <SHORT-NAME>DOP_DTC_WLCS</SHORT-NAME>
  <LONG-NAME>DTC Warning Lamp Calibration Status</LONG-NAME>
  <COMPU-METHOD>
    <CATEGORY>TEXTTABLE</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
      <COMPU-SCALES>
        <COMPU-SCALE>
          <SHORT-LABEL>WLD</SHORT-LABEL>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>warning lamp disabled</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <COMPU-SCALE>
          <SHORT-LABEL>WLE</SHORT-LABEL>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">1</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>warning lamp enabled</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
  </COMPU-METHOD>
```

```
  <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32" IS-HIGHLOW-BYTE-ORDER="true">
    <BIT-LENGTH>1</BIT-LENGTH>
  </DIAG-CODED-TYPE>
  <PHYSICAL-TYPE BASE-DATA-TYPE="A_UNICODE2STRING"/>
  <INTERNAL-CONSTR>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">1</UPPER-LIMIT>
  </INTERNAL-CONSTR>
</DATA-OBJECT-PROP>
```

EXAMPLE 3 Extracting two bytes from the PDU and calculating the vehicle velocity in km/h:

```
<DATA-OBJECT-PROP ID = "DOP_VehicleVelocity">
    <SHORT-NAME>DOP_VehicleVelocity</SHORT-NAME>
    <COMPU-METHOD>
        <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
    <DIAG-CODED-TYPE BASE-DATA-TYPE = "A_INT32" xsi:type = "STANDARD-LENGTH-TYPE">
        <BIT-LENGTH>16</BIT-LENGTH>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE = "A_INT32"/>
    <UNIT-REF ID-REF = "Unit_Kmh"/>
</DATA-OBJECT-PROP>
```

where Unit_Kmh and the appropriate physical dimension are defined as follows:

```
<UNIT ID = "Unit_Kmh">
    <SHORT-NAME>Unit_Kmh</SHORT-NAME>
    <DISPLAY-NAME>km/h</DISPLAY-NAME>
    <FACTOR-SI-TO-UNIT>3.6</FACTOR-SI-TO-UNIT>
    <OFFSET-SI-TO-UNIT>0</OFFSET-SI-TO-UNIT>
    <PHYSICAL-DIMENSION-REF ID-REF = "PD_Velocity"/>
</UNIT>


<PHYSICAL-DIMENSION ID = "PD_Velocity">
    <SHORT-NAME>PD_Velocity</SHORT-NAME>
    <LENGTH-EXP>1</LENGTH-EXP>
    <TIME-EXP>-1</TIME-EXP>
</PHYSICAL-DIMENSION>
```

7.3.6.3 Data extraction

7.3.6.3.1 For the data extraction process of PARAM from the PDU to the physical value, the following data objects are used:

— BYTE-POSITION and BIT-POSITION at PARAM which references this DATA-OBJECT-PROP;

— DIAG-CODED-TYPE including all attributes and sub-elements;

— PHYSICAL-TYPE;

— COMPU-METHOD.

7.3.6.3.2 The data extraction process comprises the steps below.

a) Determination of the size of the parameter in bits (BitLength). Dependent on the actual type of DIAG-CODE-TYPE, the length is described by another parameter, which shall be extracted before. In the case of MIN-MAX-LENGTH-TYPE, it is the net number of bits without the optional termination character.

b) Extraction of ByteCount bytes, starting at PARAM /BYTE-POSITION, where ByteCount is computed as follows:

$$ByteCount = \left\lceil \frac{BitPosition + BitLength}{8} \right\rceil$$

where $\lceil \ \rceil$ indicates the ceiling function.

c) Normalization of data: If IS-HIGHLOW-BYTE-ORDER="false" (Intel format), the extracted bytes are to be reversed, so that the byte order corresponds to the Motorola format (HIGHLOW). This step is applied to the extracted bytes as a whole, if the coded type represents a numerical type. In the case of A_UNICODE2STRING, it is separately applied to each character (pair of bytes).

d) Extraction of BIT-LENGTH bits from the normalized byte field starting at PARAM /BIT-POSITION. The valid value range of BIT-POSITION is 0 to 7. Bit counting starts with the least significant (right-most) bit of the least significant byte where BIT-POSITION is equal 0. If no BIT-POSITION is specified, the value of 0 is taken for it.

e) Application of the BIT-MASK[16]: If CONDENSED="false" (the default) apply BIT-MASK as AND-mask without any implicit shift operation. If CONDENSED="true" remove from the field all bits that are set to 0 in the BIT-MASK. The remaining bits form the extracted bit field. Its length is equal to the number of "1"s in BIT-MASK and will affect the decoding in the next step.

f) Decode the extracted bit field according to the bit length, the DIAG-CODED-TYPE/BASE-DATA-TYPE and the DIAG-CODED-TYPE/BASE-TYPE-ENCODING and assign the result to a variable containing the internal value of internal type.

g) Computation of the physical value using COMPU-METHOD.

These steps shall be performed in the reverse direction in the case of a request.

The goal of the first 6 steps is to internally assign the received data to a variable of internal type. This variable is used in the last step to calculate the physical value using the specified COMPU-METHOD. The result is a variable of the physical type holding the physical value corresponding to the received coded value. This physical value is finally displayed to the user (potentially, after converting to desired units).

7.3.6.3.3 The procedure described above is also valid for the extraction of values of coded types A_BYTEFIELD, A_ASCIISTRING, A_UTF8STRING, and A_UNICODE2STRING in principle, but in practice it is slightly modified.

a) The number of bytes to be extracted depends on the content of the PDU. For details see 7.3.6.2.

b) If the coded type is A_UNICODE2STRING, the byte order flag IS-HIGHLOW-BYTE-ORDER is applied to each character separately, i.e. each pair of bytes. For A_BYTEFIELD, A_ASCIISTRING, and A_UTF8STRING, the byte order flag is ignored.

c) Only whole bytes are affected, i.e. the BIT-POSITION is always 0 and no bit shifting is necessary.

16) This step is only related to the element BIT-MASK inside DIAG-CODED-TYPE. The content of the element inside POS-RESPONSE-SUPPRESSABLE is applied to the internal value and only for data packaging.

d) The optional leading length parameter shall be extracted separately from this parameter like an own response parameter. The byte order flag is applied to this leading length parameter, too.

Figure 74 — Example: extraction of a 4-bit value illustrates the extraction process of a 4-bit value from the PDU.

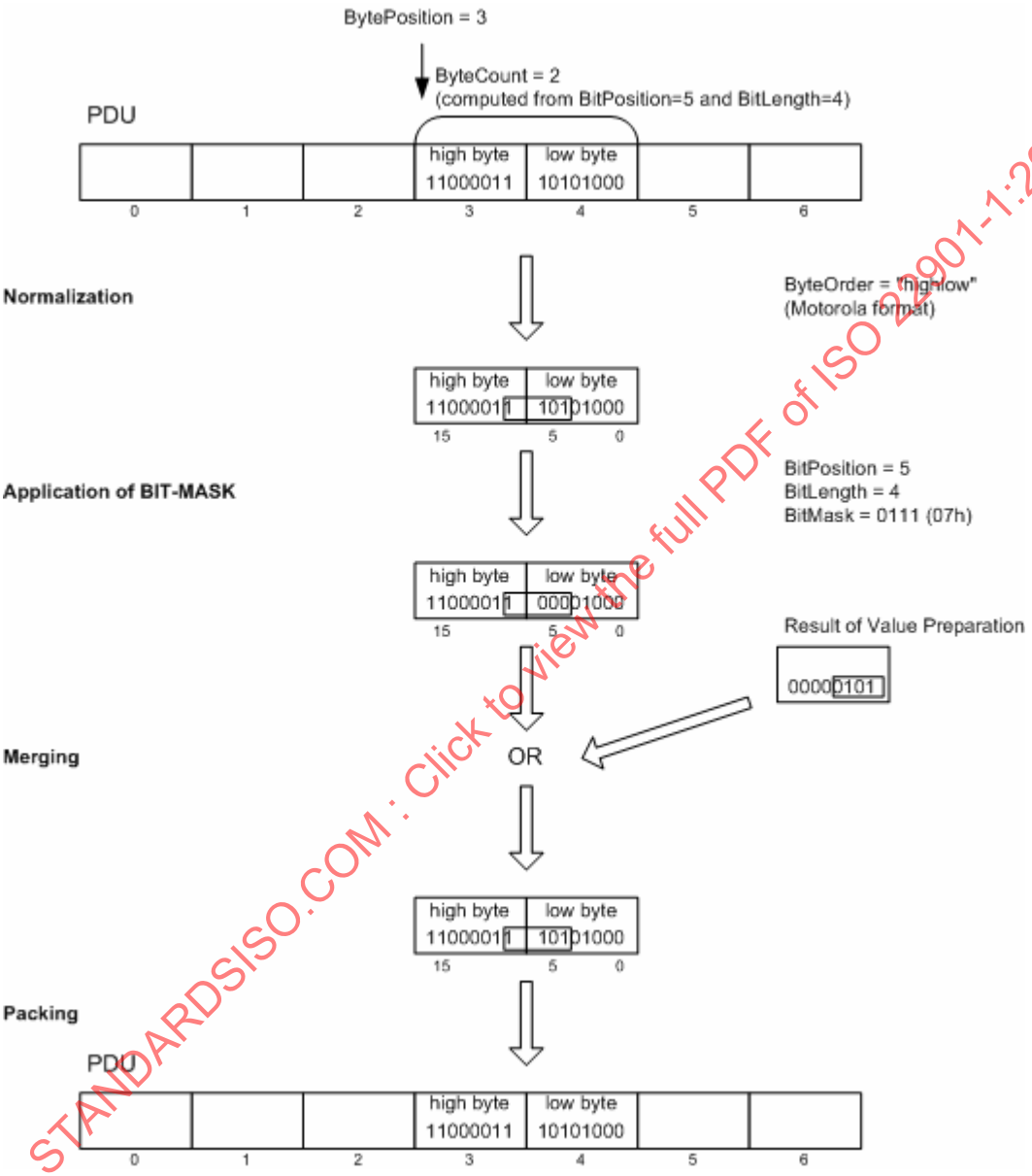


Figure 74 — Example: extraction of a 4-bit value

At first, the value ByteCount is computed, as follows:

$$ByteCount = \left\lceil \frac{BitPosition + BitLength}{8} \right\rceil = \left\lceil \frac{5+4}{8} \right\rceil = 2$$

A two-byte sub-frame is then extracted from the original data frame, consisting of the bytes 3 and 4 (BytePosition = 3, the ByteCount = 2). Since ByteOrder="highlow", the normalization step is not necessary. In the next step, the relevant bits are extracted from the two-byte frame using BitPosition and BitLength. Now the BitMask is applied. The resulting bit field is decoded as an unsigned integer of 4 bit length with default encoding and the result is assigned to a internal variable of the internal type A_UINT32. Finally, the COMPU-METHOD (in this case a TEXTTABLE) is applied to the internal value and the result is assigned to a new string-valued variable.

7.3.6.4 Data packing

The process of data packing, i.e. the conversion of the physical value into its internal representation and putting it into the PDU, follows the steps below.

a) **Value preparation**: the physical value is prepared in order to be put into the PDU. The steps below are necessary.

   1) Computation using COMPU-METHOD: the physical value is converted into the internal representation. The bytes of the target value buffer are set to zero by default.

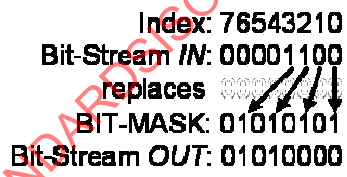
   2) Creation of bit-stream using ENCODING.

   3) Determine the effective BitLength for the coded data types A_BYTEFIELD, A_ASCIISTRING, A_UTF8STRING and A_UNICODE2STRING, including termination character if necessary.

   4) Truncation of the coded value: If CONDENSED="false" clear all bits positioned above BitLength in the bit-stream. If CONDENSED="true" count the number of 1s in the BIT-MASK calling it ValueLength. Clear all bits positioned above ValueLength in the bit-stream.

   5) Application of BIT-MASK[17] if it is specified: the BIT-MASK here is used to manipulate the truncated value that comes out of the computing method. If CONDENSED="false", a bit-wise AND-operation is applied to the bit-stream and the BIT-MASK. If CONDENSED="true", merge the BIT-MASK and the bit-stream *IN* into a new bit-stream *OUT*. Initialize *OUT* with the BIT-MASK. Then replace every "1" in bit-stream *OUT* with bits from bit-stream *IN* from least-significant-bit to most-significant-bit. Replace the least-significant "1" in bit-stream *OUT* with bit 0 of bit-stream *IN*. Replace the next "1" in bit-stream *OUT* with bit 1 of bit-stream *IN* and so on. Use bit-stream *OUT* from there on.

      EXAMPLE Expanding a 4 bit value to a bit length of 8:



b) **Preparation of a PDU cut-out**:

   1) Extraction of ByteCount bytes, starting at PARAM /BYTE-POSITION, where ByteCount is computed as follows:

$$ByteCount = \left\lceil \frac{BitPosition + BitLength}{8} \right\rceil$$

17) This step is only related to the element BIT-MASK inside DIAG-CODED-TYPE. The content of the element inside POS-RESPONSE-SUPPRESSABLE is applied to the internal value, i.e. between the steps a) and b) and it only modifies the internal value by a bit-wise OR operation, but it has no affect to the data packaging.

2) If the bytes do not exist at the time, the parameter is to be processed, they are created and initialized with 0. The result of this step is a PDU cut-out with the length of ByteCount bytes.

3) Normalization: if IS-HIGHLOW-BYTE-ORDER="false" (Intel format), the extracted bytes are to be reversed, so that the byte order corresponds to the Motorola format (HIGHLOW). This step is only applied if coded type represents a numeric value. In the case of UNICODE string, each pair of bytes, representing a single character, is swapped separately.

4) Application of BIT-MASK[18]: the bit in the bit-stream is set to 0, if the corresponding bit in the BIT-MASK is set to 1 (i.e. clear all bits, which have the value of 1 in the BIT-MASK). The bit 0 of the BIT-MASK is applied to the bit x in the PDU cut-out, where x=BitPosition; bit 1 is applied to the bit x+1 and so on. The default BIT-MASK if not specified is always a sequence of bit "1" according to the length BitLength. In other words: If BIT-MASK is not defined the bit-stream shall be copied into the PDU limited by BitLength.

   NOTE BIT-MASK is used to protect bits in the PDU from manipulation, e.g. because they are already set by other PARAMs.

5) Merging: the BitLength bits of the PDU cut-out and the bit-stream are merged by applying a logical OR-operation. The bit 0 of the bit-stream is applied to the bit x in the PDU cut-out, where x=BitPosition; bit 1 is applied to the bit x+1 and so on.

c) **Packing into the PDU**:

1) Normalization: if IS-HIGHLOW-BYTE-ORDER="false" (Intel format), the extracted bytes are to be reversed again to cancel the first normalisation out.

2) Put back the PDU cut-out into the PDU.

18) This description is only related to the element BIT-MASK inside DIAG-CODED-TYPE. The content of the element inside POS-RESPONSE-SUPPRESSABLE is applied to the internal value.

Figure 75 — Example: preparation of physical value for data packing illustrates steps a) 1) to a) 5).

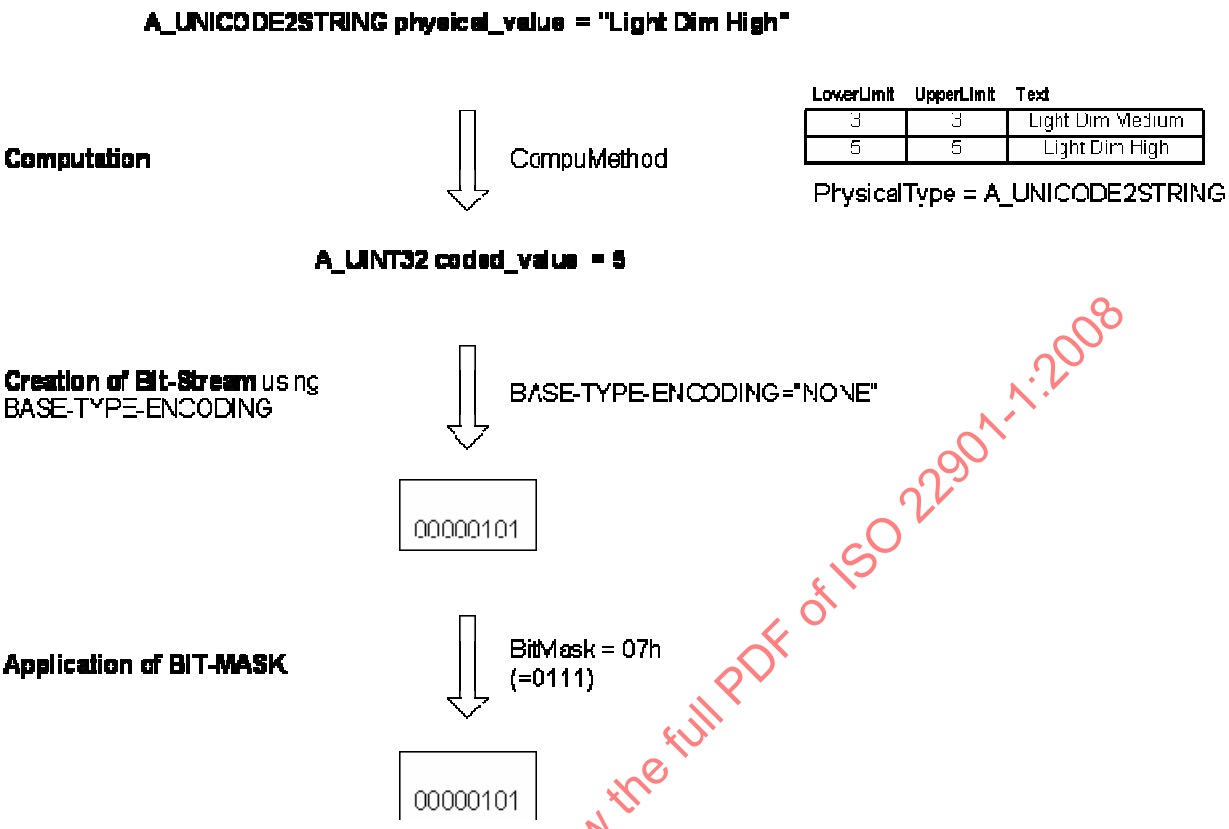


Figure 75 — Example: preparation of physical value for data packing

Figure 76 — Example: data packing illustrates steps b) 1) to c) 2).

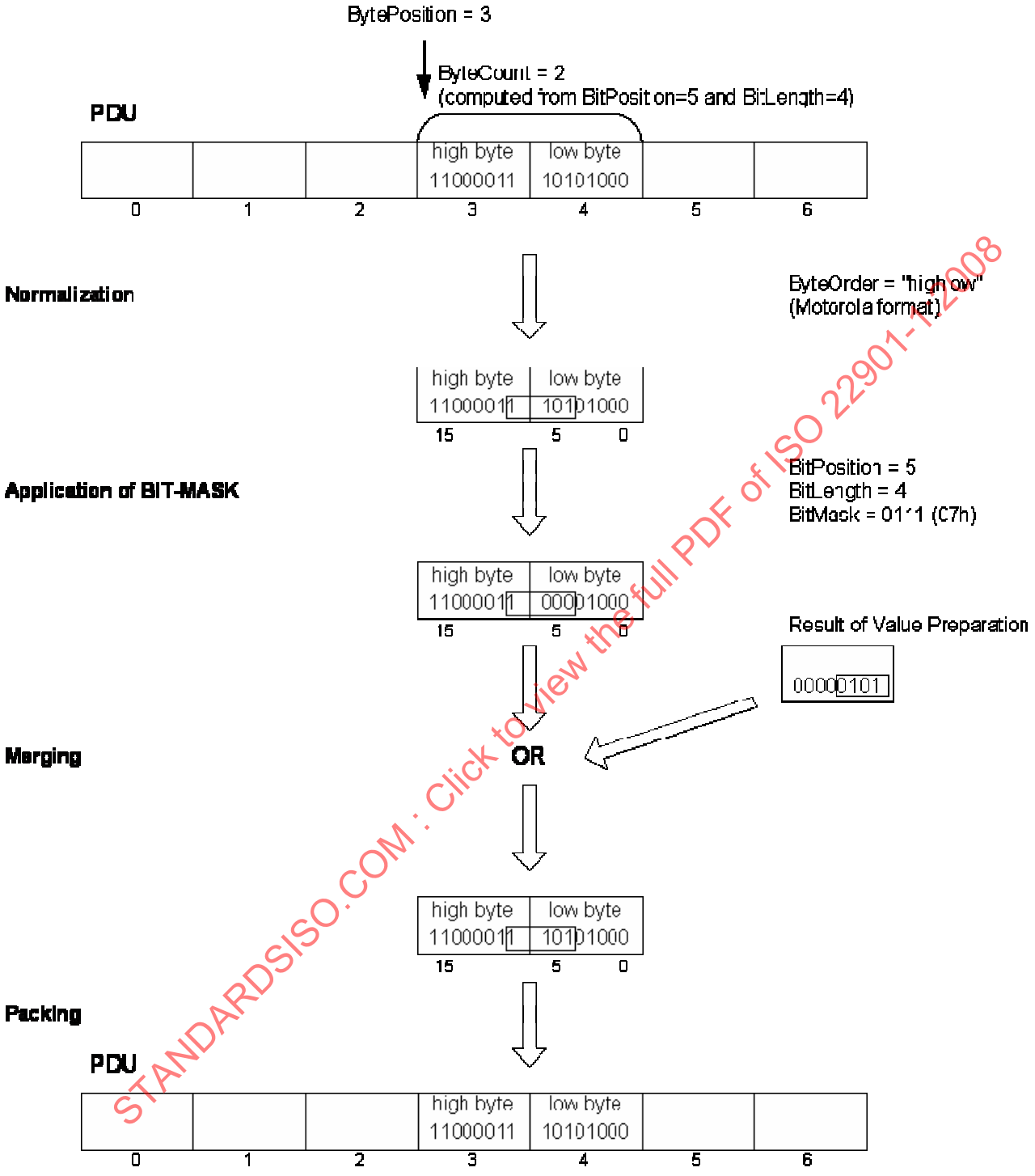


Figure 76 — Example: data packing

The items below should be pointed out.

— If only whole bytes are affected, e.g. for coded types A_BYTEFIELD, A_ASCIISTRING, A_UTF8STRING, and A_UNICODE2STRING without BIT-MASK and BIT-POSITION, the algorithm described above is equivalent to a simple coding and copying of the value into the PDU starting at the BYTE-POSITION. In the case of A_UNICODE2STRING, the flag IS-HIGHLOW-BYTE-ORDER="false" shall be considered for each pair of bytes representing a single UNICODE character.

— The optional BIT-MASK specifies the bits inside the PDU which are changed by the value. The steps 1 d), 1 e), 2 c) and 3) are equivalent to the rule, that only those bits of the bit-stream are moved (copied) into the PDU, those corresponding bit in the bit mask are set to 1 and all other bits in the PDU remains unchanged.

— The data extraction is the logical inversion of the data packing. Some steps are not required for data extraction because the PDU cut-off will not put back into the PDU or the applying of the BIT-MASK in steps 1 d), 1 e) and 2 c) are combined in only one step.

— The optional leading length parameter shall be put separately from the other parameter like an own request parameter. The byte order flag is applied to this length parameter, too.

7.3.6.5 Value comparison

Definition of LOWER-LIMIT and UPPER-LIMIT in COMPU-SCALE, INTERNAL-CONSTR ,SCALE-CONSTR and CASE requires the definition of comparison operators “<” (less than), “==” (equals) and “>” (greater than) for each data type.

In case of A_UINT32, A_INT32, A_FLOAT32 and A_FLOAT64 the values are compared numerically. In case of A_ASCIISTRING, A_UNICODE2STRING, and A_UTF8STRING only the comparison operator “equals” is defined. Two strings are equal, if and only if both strings have the same length and each corresponding character code is equal.

In case of A_BYTEFIELD the comparison operators are defined as follows. The values are filled up with leading 00-Bytes until they are of same length. Right-most byte is the least significant byte. After that the values are interpreted as numerical values (large unsigned integers) and can be compared like other numerical types.

EXAMPLE The following (in-) equations are valid for byte fields:

0x00ff01 == 0x00FF01 == 0xFF01 > 0x00AE02 == 0xAE02 == 0xae02; 0xAE02 < 0xFF01.

7.3.6.6 Computational methods

7.3.6.6.1 General

Computational methods (COMPU-METHOD) are needed to calculate between the internal type and the physical type of the diagnostic values. An internal diagnostic value is for example the result of the extraction of a response parameter from the response message received from the ECU. To this internal diagnostic value a computational method is applied to get the physical value for representation. On the other hand a physical value given by the user is to be converted into internal value in order to be sent to an ECU. This is also done by means of the COMPU-METHOD. The extraction of data items from a response and the coding of data items into a request are made via DIAG-CODED-TYPE that is described in 7.3.6.2, 7.3.6.3, and 7.3.6.4.

See Figure 77 — UML representation of conversion for further detail.

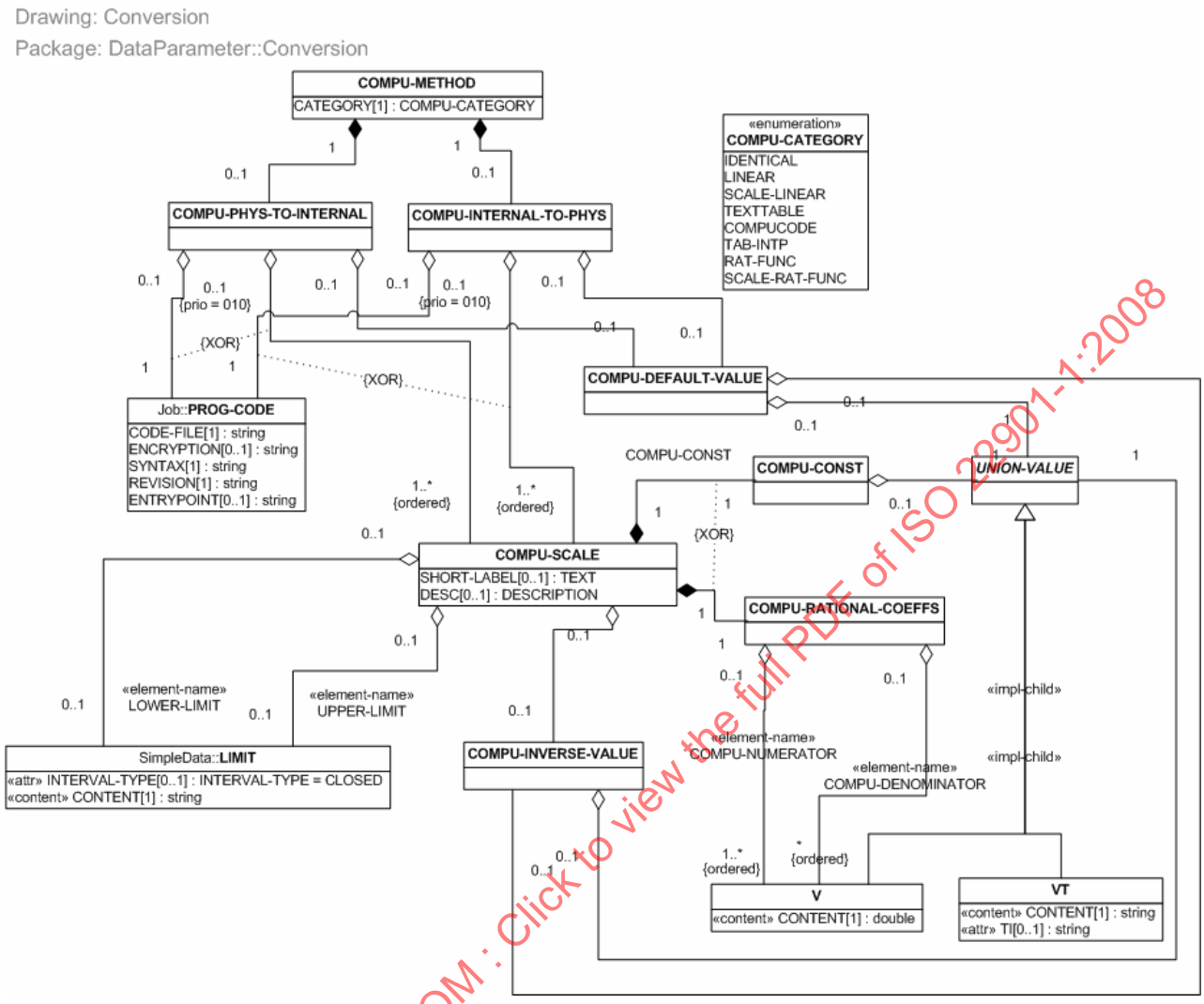


Figure 77 — UML representation of conversion

A COMPU-METHOD can specify a COMPU-INTERNAL-TO-PHYS and/or a COMPU-PHYS-TO-INTERNAL element, which specifies the transformation of a coded value read out of the PDU into a physical value of a type compliant to the DOP-BASE this COMPU-METHOD belongs to or a physical value into a coded value, respectively. For invertible COMPU-METHODS (e.g. a LINEAR function), the inversion shall be performed automatically, e.g. by a D-server. In case the COMPU-INTERNAL-TO-PHYS calculation is not invertible the corresponding COMPU-PHYS-TO-INTERNAL element can be used to define a deterministic inverse calculation. The same holds true, if only a COMPU-PHYS-TO-INTERNAL has been defined at this is not invertible. A COMPU-INTERNAL-TO-PHYS can be used to solve this situation. A COMPU-METHOD of one of the categories RAT-FUNC, SCALE-RAT-FUNC, or COMPUCODE is never invertible and a separate element for the inverse direction shall be specified, if the inverse direction is required. Be aware that the author is responsible for the consistency between the COMPU-INTERNAL-TO-PHYS and the COMPU-PHYS-TO-INTERNAL calculation, if this consistency is expected. There is no possibility for an ODX tool to check that the one is really a valid inversion of the other one.

For an example on the use of COMPU-INVERSE-VALUE, see 7.3.6.6.7, where a TEXTTABLE is specified. Since multiple internal values are mapped onto the same string as physical value, there is no deterministic inversion. However, for every COMPU-SCALE element a COMPU-INVERSE-VALUE is specified. For example, the values 5…9 map onto the string “ON”. Since it is non-deterministic to which value “ON” should

be mapped when converting from physical to internal value, COMPU-INVERSE-VALUE defines this deterministically to be the value 5.

In case both directions (COMPU-PHYS-TO-INTERNAL and COMPU-INTERNAL-TO-PHYS) are defined in one COMPU-METHOD both need to be used in case this COMPU-METHOD appears in a REQUEST and RESPONSE. If only one direction is used, inversion takes place in the respective other direction, if the corresponding conversion is invertible.

Table 10 — COMPU-METHOD outlines the various cases.

Table 10 — COMPU-METHOD

COMPU-METHOD specifies …	calculation of direction internal -> physical (response) uses …	calculation of direction physical -> internal (request) uses …
only COMPU-INTERNAL-TO-PHYS	COMPU-INTERNAL-TO-PHYS	COMPU-INTERNAL-TO-PHYS in inverse direction
both COMPU-INTERNAL-TO-PHYS and COMPU-PHYS-TO-INTERNAL	COMPU-INTERNAL-TO-PHYS	COMPU-PHYS-TO-INTERNAL
only COMPU-PHYS-TO-INTERNAL	COMPU-PHYS-TO-INTERNAL in inverse direction	COMPU-PHYS-TO-INTERNAL

There are eight different types of COMPU-METHODs: IDENTICAL, LINEAR, SCALE-LINEAR, TEXTTABLE, COMPUCODE, TAB-INTP, RAT-FUNC and SCALE-RAT-FUNC. The mandatory member CATEGORY defines the type of the COMPU-METHOD.

A generic approach is used for the definition of functions. The following mathematical function forms the basis of all functions apart from IDENTICAL, TEXTTABLE and COMPUCODE:

$$f(x) = \frac{V_{N0} + V_{N1} * x + V_{N2} * x^2 + \ldots + V_{Nn} * x^n}{V_{D0} + V_{D1} * x + V_{D2} * x^2 + \ldots + V_{Dm} * x^m}$$

Numerator

Denominator

For each interval, a numerator (COMPU-NUMERATOR) and a denominator (COMPU-DENOMINATOR) are defined. Each of them defines a sequence of V-values representing a polynomial. The first V is the coefficient for x^0, the second V is the coefficient for x^1, and so on.

The domain of the a COMPU-METHOD of CATEGORY TEXTTABLE, SCALE-LINEAR or SCALE-RAT-FUNC can be divided into several intervals each specified by one COMPU-SCALE. The domain of categories LINEAR and RAT-FUNC shall define exactly one COMPU-SCALE, which may only restrict the single domain. The categories IDENTICAL and COMPUCODE shall not define any COMPU-SCALE at all. The category TAB-INTP shall specify at least two COMPU-SCALEs, but in this case, each COMPU-SCALE will define only a singular value, not an interval.

A scale for a function is defined via COMPU-SCALE. When applying intervals for scales, defined by UPPER-LIMIT and LOWER-LIMIT, the user shall guarantee that no intervals will overlap (Exception: Only for the COMPU-METHOD of type TEXTTABLE overlapping is allowed.). Therefore the attribute INTERVAL-TYPE and its values “OPEN”, “CLOSED” or “INFINITE” can be used. The value “OPEN” does not include the boundary, whereas the value “CLOSED” does. “INFINITE” denotes an open-end interval. For the categories LINEAR and RAT-FUNC a missing LOWER-LIMIT or UPPER-LIMIT means, that the domain is not restricted in this corresponding direction. For the categories TEXTTABLE, SCALE-LINEAR, and SCALE-RAT-FUNC the LOWER-LIMIT shall be defined in any COMPU-SCALE definition. If no UPPER-LIMIT is defined the COMPU-SCALE will be applied only for the value defined in LOWER-LIMIT. In this case INTERVAL-TYPE at LOWER-

LIMIT shall be defined as CLOSED. For the category TAB-INTP, LOWER-LIMIT contains the singular value and UPPER-LIMIT shall not be present.

The processing order of the scales of any category is the order of specification inside the ODX document: The first matching scale closes the processing in the D-server. This is even the case, if the value falls into several scales of a TEXTTABLE. Besides the COMPU-SCALEs, the categories TEXTTABLE, SCALE-LINEAR, and SCALE-RAT-FUNC may define an optionally default scale COMPU-DEFAULT-VALUE. If the COMPU-METHOD is used in the inverse direction and if the optional COMPU-INVERSE-VALUE is not present inside COMPU-DEFAULT-VALUE, the default scale is ignored.

The use of this optional element differs for both computational directions, if it is really applicable:

— in the case of computational direction internal to physical value, the physical value is only reported as the result, if the internal value does not match any defined interval (wherever they can be defined);

— in the case of computational direction physical to internal value, the internal value is only reported as the result, if the physical value given as input value does not match any defined interval (wherever they can be defined) and if it is equal to the physical value given inside the default scale.

If the value does not fall into any COMPU-SCALE and if the optionally COMPU-DEFAULT-VALUE is not present or not applicable, the D-server shall indicate a runtime error.

The different processing of the default scale for both directions reflects the idea that an invalid input value, e.g. invalid due to a spelling error in the case of a TEXTTABLE, should not be accepted in the case of a request parameter, while any allowed response parameter could use the default scale if applicable.

For all COMPU-METHODs except those of CATEGORY IDENTICAL, TEXTTABLE and COMPUCODE, the calculation type (float or integer) is deduced from the type of the internal and physical values. It has a decisive influence on the precision of the calculation.

The type of the calculation is determined according to the following rules:

a) if one of the types (physical or internal) is A_FLOAT32 or A_FLOAT64, the calculation is of type A_FLOAT64 bit;

b) if both types are A_UINT32, the calculation is of type A_UINT32;

c) in any other case, the calculation is of type A_INT32.

The type of the coefficients in COMPU-NUMERATOR is equal to the type of the calculation. Also the type of the coefficients in COMPU-DENOMINATOR is equal to the type of the calculation, except for the categories LINEAR and SCALE-LINEAR, where the only coefficient is always of type A_UINT32. The COMPU-DENOMINATOR shall be non-zero. If the COMPU-DENOMINATOR is not specified, the numerical value 1 is assumed for the denominator, i.e. no division is executed.

The type of the values of the other elements inside COMPU-INTERNAL-TO-PHYS and COMPU-PHYS-TO-INTERNAL are summarized in Table 10 — COMPU-METHOD.

Table 11 — Elements of COMPU-METHOD

Element	COMPU-INTERNAL-TO-PHYS	COMPU-PHYS-TO-INTERNAL
LOWER-LIMIT	internal type	physical type
UPPER-LIMIT	internal type	physical type
COMPU-INVERSE-VALUE	internal type	physical type
COMPU-CONST	physical type	internal type
COMPU-DEFAULT-VALUE	physical type	internal type
COMPU-DEFAULT-VALUE/ COMPU-INVERSE-VALUE	internal type	physical type

IMPORTANT — Not all elements are applicable for all categories of COMPU-METHOD. Some of the elements have the sub-elements V and VT. In all of theses cases, the sub-element V shall be used for numerical types and VT for non-numerical types.

The content specified inside the XML document shall match the Schema data types according to Table 12 — Schema data types.

Table 12 — Schema data types

BASE-DATA-TYPE	Schema data type
A_INT32	xsd:int
A_UINT32	xsd:unsignedInt
A_FLOAT32	xsd:float
A_FLOAT64	xsd:double
A_BYTEFIELD	xsd:hexBinary
A_ASCIISTRING	xsd:string
A_UTF8STRING	xsd:string
A_UNICODE2STRING	xsd:string

If an overflow or division by zero occurs during the calculation, the D-server shall report an error message.

If the calculation type is A_FLOAT64 and a result type is either A_INT32 or A_UINT32, the commercial rounding is applied, i.e. 0.5 -> 1, 1.3 -> 1, 1.5 ->2.

7.3.6.6.2 Identical

Figure 78 — COMPU-METHOD of type identical shows a COMPU-METHOD of type IDENTICAL (CATEGORY="IDENTICAL") which just passes on the input value so that the internal value and the physical one are identical.

Mathematical function:

$f(x) = x, x \in R$

Certain data types:

— Internal: all types

— Physical: all types

— (physical type = internal type)

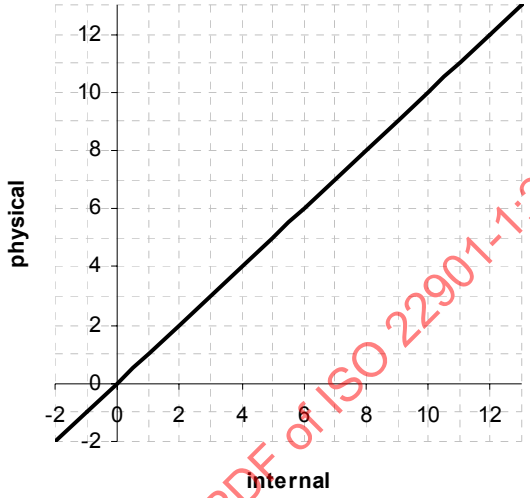


Figure 78 — COMPU-METHOD of type identical

For COMPU-METHODs of this type, the data objects COMPU-INTERNAL-TO-PHYS and COMPU-PHYS-TO-INTERNAL are not allowed. This is the simplest type of a COMPU-METHOD and can be instantiated as follows:

EXAMPLE

```
<COMPU-METHOD>
  <CATEGORY>IDENTICAL</CATEGORY>
</COMPU-METHOD>
```

The value domain of an identical function includes all values of all types (including ASCII strings). This does not become apparent from the drawing above, but it is possible. The only limitation, which shall be observed, is that the physical and the internal types shall be identical. If the coded type (at BASE-TYPE-ENCODING at DIAG-CODED-TYPE) is A_ASCIISTRING or A_UTF8STRING, the conversion to or from the physical BASE-DATA-TYPE A_UNICODE2STRING shall be done implicitly during the decoding or encoding process.

7.3.6.6.3 Linear

Figure 79 — COMPU-METHOD of type linear shows COMPU-METHODs of CATEGORY LINEAR which multiply the input value with a factor and add an offset. As an option, the sum can be divided by a constant unsigned integer. Exactly one COMPU-SCALE shall be defined. It contains COMPU-RATIONAL-COEFFS within which COMPU-NUMERATOR and COMPU-DENOMINATOR are declared. The numerator should contain two values. The first one is the offset (V_{N0}), the second one is the factor (V_{N1}). If the denominator is present it shall specify exactly one unsigned integer value. The LIMITs at COMPU-SCALE can be used to restrict the value domain.

Via this COMPU-METHOD, it is possible to create a function returning a constant value by setting V_{N1} = 0. In this case, COMPU-INVERSE-VALUE shall be defined if the reverse calculation is required.

Mathematical function:

$$f(x) = \frac{V_{N0} + V_{N1} * x}{V_{D0}}$$

Possible data types:

— Internal: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

— Physical: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

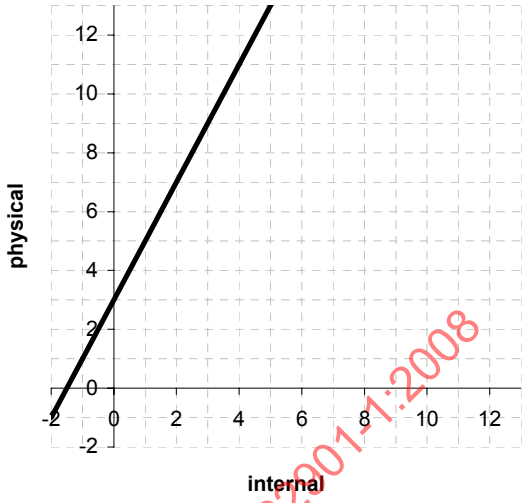


Figure 79 — COMPU-METHOD of type linear

EXAMPLE

$$f(x) = 3 + 2x$$

```
<COMPU-METHOD>
  <CATEGORY>LINEAR</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>3</V><V>2</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>1</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```


7.3.6.6.4 Scale linear

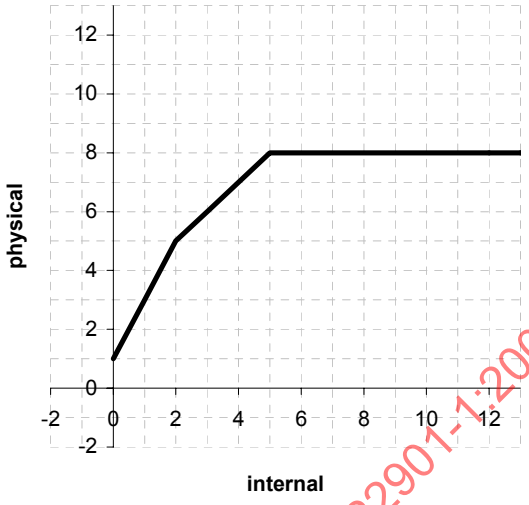
Figure 80 — COMPU-METHOD of type scale linear shows functions (CATEGORY="SCALE-LINEAR") which are similar to the linear ones with the difference that more than one interval can be defined as the domain of the function and different calculations can be applied to the intervals. The boundaries of the intervals shall not overlap.

Condition for invertibility: adjacent COMPU-SCALES shall have the same values on their common boundaries AND the V_{N1} of all intervals shall have the same sign or shall be 0. If V_{N1} = 0, COMPU-INVERSE-VALUE shall be specified.

Mathematical function:

$$f(x) = \begin{cases} \dfrac{V_{N0}^{1} + V_{N1}^{1} * x}{V_{D0}^{1}}, x \in I_1 \\ \cdots \\ \dfrac{V_{N0}^{n} + V_{N1}^{n} * x}{V_{D0}^{n}}, x \in I_n \end{cases}$$

where I_1, … , I_n are arbitrary disjunctive intervals.



Possible data types:

— Internal: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

— Physical: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

Figure 80 — COMPU-METHOD of type scale linear

EXAMPLE

$$f(x) = \begin{cases} 1 + 2x, x \in [0,2) \\ 3 + x, x \in [2,5) \\ 8, x \in [5, +\infty) \end{cases}$$

```
<COMPU-METHOD>
  <CATEGORY>SCALE-LINEAR</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="OPEN">2</UPPER-LIMIT>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>1</V><V>2</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>1</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">2</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="OPEN">5</UPPER-LIMIT>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>3</V><V>1</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>1</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">5</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="INFINITE"/>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>8</V><V>0</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>1</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
```

```
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

7.3.6.6.5 Rational function

Figure 81 — COMPU-METHOD of type rational function shows a function (CATEGORY="RAT-FUNC") which differs from the linear function in the omission of the restrictions for COMPU-NUMERATORs and COMPU-DENOMINATORs. They can have as many V-values as needed. The calculation in the reverse direction is not allowed for this type.

Mathematical function:

$$f(x) = \frac{V_{N0} + V_{N1} * x + V_{N2} * x^2 + \ldots + V_{Nn} * x^n}{V_{D0} + V_{D1} * x + V_{D2} * x^2 + \ldots + V_{Dm} * x^m}$$

Possible data types:

— Internal: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

— Physical: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

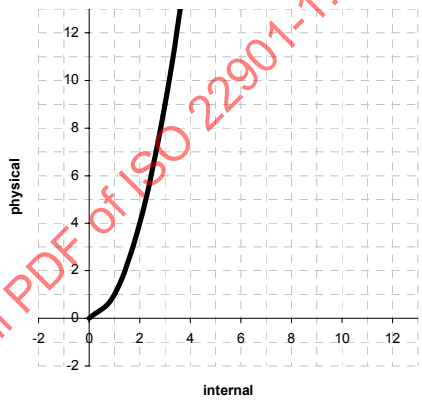


Figure 81 — COMPU-METHOD of type rational function

EXAMPLE

$$f(x) = x^2, x \in [0, +\infty)$$

```
<COMPU-METHOD>
  <CATEGORY>RAT-FUNC</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="INFINITE"/>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR><V>0</V><V>0</V><V>1</V></COMPU-NUMERATOR>
          <COMPU-DENOMINATOR><V>1</V></COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

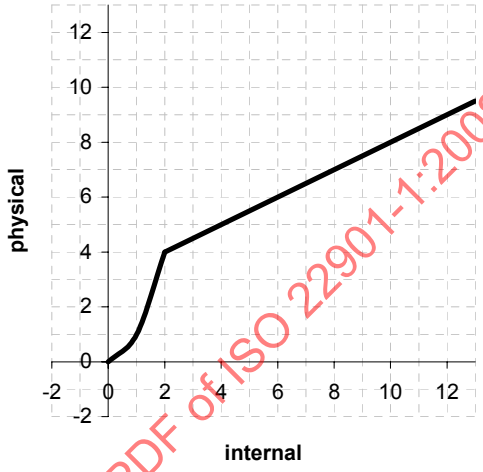
7.3.6.6.6 Scale rational function

Figure 82 — COMPU-METHOD of type scale rational function shows COMPU-METHODs of CATEGORY SCALE-LINEAR of the CATEGORY SCALE-RAT-FUNC which can have more than one COMPU-SCALE to provide different calculations for different intervals. The calculation in the reverse direction is not allowed for this type.

Mathematical function:

$$f(x)=\begin{cases}\dfrac{V_{N0}^{1}+V_{N1}^{1}*x+V_{N2}^{1}*x^{2}+\ldots+V_{Nn}^{1}*x^{n}}{V_{D0}^{1}+V_{D1}^{1}*x+V_{D2}^{1}*x^{2}+\ldots+V_{Dm}^{1}*x^{m}}, x\in I_1\\ \cdots\\ \dfrac{V_{N0}^{k}+V_{N1}^{k}*x+V_{N2}^{k}*x^{2}+\ldots+V_{Nn}^{k}*x^{n}}{V_{D0}^{k}+V_{D1}^{k}*x+V_{D2}^{k}*x^{2}+\ldots+V_{Dm}^{k}*x^{m}}, x\in I_k\end{cases}$$

where $I_1, \ldots, I_k$ are arbitrary disjunctive intervals.



Possible data types:

— Internal: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

— Physical: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

Figure 82 — COMPU-METHOD of type scale rational function

EXAMPLE

$$f(x)=\begin{cases}x^{2}, x\in[0,2)\\ \cdots\\ \dfrac{6+x}{2}, x\in[2,\infty)\end{cases}$$

```
<COMPU-METHOD>
    <CATEGORY>SCALE-RAT-FUNC</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
        <COMPU-SCALES>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">0</LOWER-LIMIT>
                <UPPER-LIMIT INTERVAL-TYPE = "OPEN">2</UPPER-LIMIT>
                <COMPU-RATIONAL-COEFFS>
                    <COMPU-NUMERATOR>
                        <V>0</V>
                        <V>0</V>
                        <V>1</V>
                    </COMPU-NUMERATOR>
                    <COMPU-DENOMINATOR>
                        <V>1</V>
                    </COMPU-DENOMINATOR>
                </COMPU-RATIONAL-COEFFS>
            </COMPU-SCALE>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">2</LOWER-LIMIT>
```

```
                <UPPER-LIMIT INTERVAL-TYPE = "INFINITE"/>
                <COMPU-RATIONAL-COEFFS>
                    <COMPU-NUMERATOR>
                        <V>6</V>
                        <V>1</V>
                    </COMPU-NUMERATOR>
                    <COMPU-DENOMINATOR>
                        <V>2</V>
                    </COMPU-DENOMINATOR>
                </COMPU-RATIONAL-COEFFS>
            </COMPU-SCALE>
        </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

7.3.6.6.7 Text table

Figure 83 — COMPU-METHOD of type Texttable shows the TEXTTABLE which is commonly specified only by COMPU-INTERNAL-TO-PHYS and used in both directions. First, it is described for COMPU-INTERNAL-TO-PHYS. The type TEXTTABLE is used for transformations of the internal value into a textual expression. At least one COMPU-SCALE shall be specified. In each scale UPPER-LIMIT and LOWER-LIMIT define an interval. For a TEXTTABLE, these scales may overlap. To find a single matching interval, the COMPU-SCALEs are to be processed in ODX document order. The first matching interval is taken. The optional COMPU-DEFAULT-VALUE can be used to define the physical value if the internal value does not lie in any given interval. COMPU-CONST with the data object VT defines the resulting text for the appropriate interval. If the reverse calculation is needed, for each scale the COMPU-INVERSE-VALUE can be used to define the reverse calculation result. In case where LOWER-LIMIT is equal to UPPER-LIMIT, the COMPU-INVERSE-VALUE is optional. If it is omitted, the LOWER-LIMIT is used as the inverse value. To guarantee the unambiguous reverse calculation, the COMPU-INVERSE-VALUE of all the COMPU-SCALEs with same physical string value shall be identical. If the internal data type is a string type, the definition of a range is not allowed. If the element UPPER-LIMIT is explicitly specified, its content shall be equal to the content of LOWER-LIMIT.

In a TEXTTABLE inside COMPU-PHYS-TO-INTERNAL, the textual expression shall be specified in LOWER-LIMIT (UPPER-LIMIT should be omitted or shall be equal) and the internal value shall be specified inside COMPU-CONST. Each COMPU-SCALE, therefore, represents a single textual value. The physical value provided is matched in ODX document order against each COMPU-SCALE. Therefore, these COMPUSCALEs may overlap. The COMPU-CONST of the first COMPU-SCALE that represents exactly the same text as physical value contains the resulting internal value.

Example of a TEXTTABLE function:

```
internal  physical
 [0,5)          "OFF"
 [5,9)          "ON"
 [9,∞)          "OVERLOAD"
```

Possible data types:

— Internal: A_INT32, A_UINT32, A_FLOAT32; _A_FLOAT64; A_ASCIISTRING, A_UTF8STRING, A_BYTEFIELD; A_UNICODE2STRING,

— Physical: A_UNICODE2STRING.

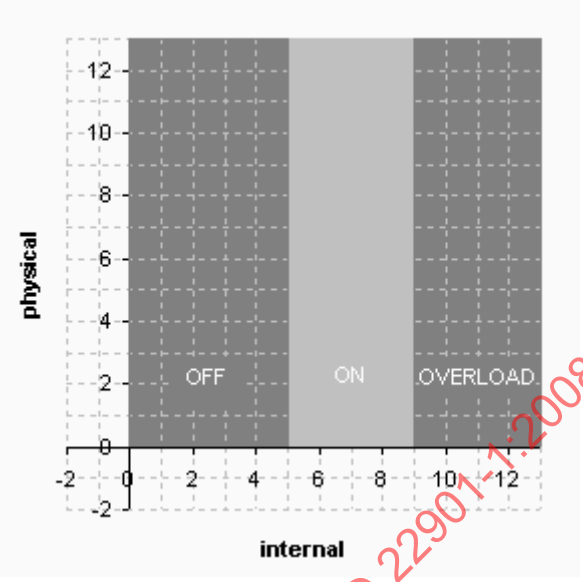


Figure 83 — COMPU-METHOD of type Texttable

The example shown above is implemented as follows:

```
<COMPU-METHOD>
    <CATEGORY>TEXTTABLE</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
        <COMPU-SCALES>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">0</LOWER-LIMIT>
                <UPPER-LIMIT INTERVAL-TYPE = "OPEN">5</UPPER-LIMIT>
                <COMPU-INVERSE-VALUE>
                    <V>0</V>
                </COMPU-INVERSE-VALUE>
                <COMPU-CONST>
                    <VT>OFF</VT>
                </COMPU-CONST>
            </COMPU-SCALE>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">5</LOWER-LIMIT>
                <UPPER-LIMIT INTERVAL-TYPE = "OPEN">9</UPPER-LIMIT>
                <COMPU-INVERSE-VALUE>
                    <V>5</V>
                </COMPU-INVERSE-VALUE>
                <COMPU-CONST>
                    <VT>ON</VT>
                </COMPU-CONST>
            </COMPU-SCALE>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">9</LOWER-LIMIT>
                <UPPER-LIMIT INTERVAL-TYPE = "INFINITE"/>
                <COMPU-INVERSE-VALUE>
                    <V>9</V>
                </COMPU-INVERSE-VALUE>
                <COMPU-CONST>
                    <VT>OVERLOAD</VT>
                </COMPU-CONST>
            </COMPU-SCALE>
        </COMPU-SCALES>
```

```
        <COMPU-DEFAULT-VALUE>
            <VT>UNDEF</VT>
        </COMPU-DEFAULT-VALUE>
    </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

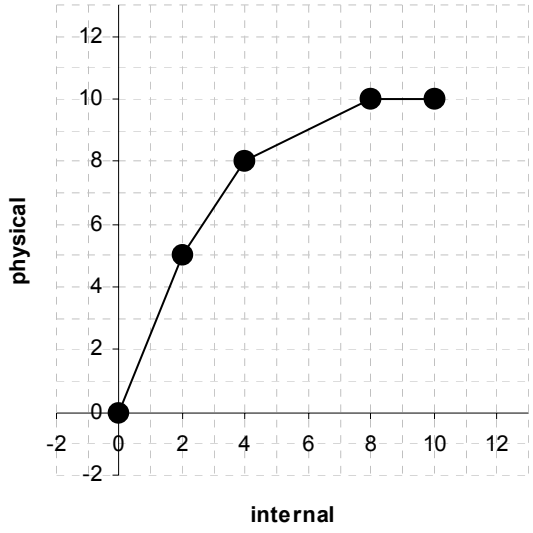
7.3.6.6.8 Tab interpolated

Figure 84 — COMPU-METHOD of type tab interpolated shows a COMPU-METHOD of CATEGORY TAB-INTP which defines a set of points with linear interpolation between them. Only LOWER-LIMIT and COMPU-CONST/V are used for that purpose (UPPER-LIMIT should be omitted or shall be equal to LOWER-LIMIT). If the calculation is used in the direction of specification, it is performed as follows: The input value is matched against the value intervals in the order they are defined by the COMPU-SCALEs; the limits of a value interval are defined by the LOWER-LIMIT values of two adjacent COMPU-SCALEs. The processing starts with the first and second COMPU-SCALEs, in the next step the second and the third COMPU-SCALEs are checked, then the third and forth, and so on. Once the input value matches an end point or lies inside an interval the iteration is stopped and the result value is to be calculated via linear interpolation for the interval specified by the values inside COMPU-CONST/V of the same COMPU-SCALEs. If the internal value is less than the smallest defined X-value or greater than the greatest defined X-value a runtime error shall be indicated by the D-server.

The calculation in the reverse direction is performed in an analogous manner and also in the same order, with the difference that the input value is tested against the value intervals, which are defined by the COMPU-CONST/V values of two adjacent COMPU-SCALEs. The first pair of COMPU-SCALEs matching the value will be used to calculate the result in reverse direction by interpolation the interval specified by their LOWER-LIMITs.

Example of a TAB-INTP function:

Internal	Physical
0	0
2	5
4	8
8	10
10	10



Possible data types:

— Internal: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

— Physical: A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64

Figure 84 — COMPU-METHOD of type tab interpolated

The following XML code shows the implementation of this example:

```
<COMPU-METHOD>
    <CATEGORY>TAB-INTP</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
        <COMPU-SCALES>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">0</LOWER-LIMIT>
```

```
                <COMPU-CONST>
                    <V>0</V>
                </COMPU-CONST>
            </COMPU-SCALE>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">2</LOWER-LIMIT>
                <COMPU-CONST>
                    <V>5</V>
                </COMPU-CONST>
            </COMPU-SCALE>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">4</LOWER-LIMIT>
                <COMPU-CONST>
                    <V>8</V>
                </COMPU-CONST>
            </COMPU-SCALE>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">8</LOWER-LIMIT>
                <COMPU-CONST>
                    <V>10</V>
                </COMPU-CONST>
            </COMPU-SCALE>
            <COMPU-SCALE>
                <LOWER-LIMIT INTERVAL-TYPE = "CLOSED">10</LOWER-LIMIT>
                <COMPU-CONST>
                    <V>10</V>
                </COMPU-CONST>
            </COMPU-SCALE>
        </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

7.3.6.6.9 Computational code

A COMPU-METHOD of CATECORY COMPUCODE is used when a computer program should do the computation. This shall be coded in Java programming language. Java libraries allowed to use are listed in Table 13 — Java packages for computational code. A java class with computational code shall implement the interface `I_CompuCode`.

Interface of computational code:

```
public interface I_CompuCode {
public Object compute(Object input);
}
```

As this interface definition shows, no parameterization of the Java code is possible. The only way to signal an error condition inside the Java code is to throw a Java exception.

For both input and result parameter, the mapping of the coded and physical BASE-DATA-TYPE of ODX to the Java classes is defined as follows:

— A_INT32 to Integer;

— A_UINT32 to Long;

— A_FLOAT32 to Float;

— A_FLOAT64 to Double;

— A_BYTEFIELD to byte[];

— A_ASCIISTRING to String (Unicode);

— A_UTF8STRING to String (Unicode);

— A_UNICODE2STRING to String (Unicode).

The Java code may expect an object of this class as input parameter and shall return the result value as an object of this class. Otherwise, the Java code or the D-server may report a runtime error. All objects of the Java string class are always encoded as UCS-2, corresponding to A_UNICODE2STRING. If the coded type (at BASE-TYPE-ENCODING at DIAG-CODED-TYPE) is A_ASCIISTRING the Java String shall only contain characters that can be represented in an A_ASCIISTRING. The conversion between Java String and the specified internal type A_ASCIISTRING or A_UTF8STRING is done by the D-server during the decoding or encoding process. Therefore, the encoding or decoding is generally not the task of the Java code. If the encoding or decoding is desired to be done inside the Java code, a A_BYTEFIELD shall be used as coded type.

Table 13 — Java packages for computational code

Package	Description
Java.lang	Provides classes that are fundamental to the design of the Java programming language.
Java.math	Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal).
Java.text	Provides Classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages.
Java.util	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalisation, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

No instance of COMPU-SCALE or COMPU-DEFAULT-VALUE is allowed for this type of computational method. All data types for physical and internal value are applicable. COMPUODE is not invertible. If required, COMPU-INTERNAL-TO-PHYS shall be specified for computation from internal type to physical and COMPU-PHYS-TO-INTERNAL for physical to internal.

To refer to the right computational code, the name of the source file or class file shall be placed as value of the member CODE-FILE. ENCRYPTION provides a possibility to put in the information about encryption method if used. This member defines all details necessary for the encryption. The definition takes place in a user specific way and is not specified in this part of ISO 22901. There are three storage kinds for computational java code which are defined by the member SYNTAX: “JAVA” for java source code, “CLASS” for java byte code (compiled) and “JAR” for java archive. REVISION indicates the version of the file to be used. It can be empty if no versioning is supported. When storing as java archive an ENTRYPOINT specifies the name of the class containing the program code. The standard only supports jobs written in Java. If jobs in another syntax (e.g. a Windows DLL) need to be supported, a corresponding value for the SYNTAX field and the entry point into the DLL can be included.

EXAMPLE 1 A COMPUCODE function defined in XML referencing a java file:

```
<COMPU-METHOD>
    <CATEGORY>COMPUCODE</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
        <PROG-CODE>
            <CODE-FILE>myCompucode2.java</CODE-FILE>
            <SYNTAX>JAVA</SYNTAX>
```

```
            <REVISION>1.0</REVISION>
        </PROG-CODE>
    </COMPU-INTERNAL-TO-PHYS>
    <COMPU-PHYS-TO-INTERNAL>
        <PROG-CODE>
            <CODE-FILE>myCompucode2.java</CODE-FILE>
            <SYNTAX>JAVA</SYNTAX>
            <REVISION>1.0</REVISION>
        </PROG-CODE>
    </COMPU-PHYS-TO-INTERNAL>
</COMPU-METHOD>
```

EXAMPLE 2 A computational code in a java file (e.g. my_compucode1.java):

```
import java.lang.* ;
public class myCompucode1
implements I_CompuCode {
public Object compute(Object input){
.. .. ..
}
}
```

EXAMPLE 3 A COMPUCODE function defined in XML referencing a java class:

```
<COMPU-METHOD>
    <CATEGORY>COMPUCODE</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
        <PROG-CODE>
            <CODE-FILE>myCompucode1</CODE-FILE>
            <SYNTAX>CLASS</SYNTAX>
            <REVISION>1.0</REVISION>
        </PROG-CODE>
    </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

EXAMPLE 4 Referencing the class "myCompucode1" in the archive "allCompucode.jar":

```
<COMPU-METHOD>
    <CATEGORY>COMPUCODE</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
        <PROG-CODE>
            <CODE-FILE>allCompucode.jar</CODE-FILE>
            <SYNTAX>JAR</SYNTAX>
            <REVISION>1.0</REVISION>
            <ENTRYPOINT>myCompucode1</ENTRYPOINT>
        </PROG-CODE>
    </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

7.3.6.7 Units

7.3.6.7.1 General

Usually units are used to augment the value with additional information (e.g. “m/s” or “litre”). That is necessary for a correct interpretation of the physical value for input and output processes. The application shows the DISPLAY-NAME of the unit besides the physical value of the diagnostic data.

The conversion of values into other units like “km/h” into “miles per hour” is also possible. Therefore the unit involves information about its physical dimensions. The substructure of physical dimensions defines all used quantities in the SI-System (e.g. velocity as length/time corresponds to m/s). The unit references one physical dimension. If the physical dimension of two units is identical a conversion between them is possible.

Figure 85 — UML representation of unit defines the modelling.

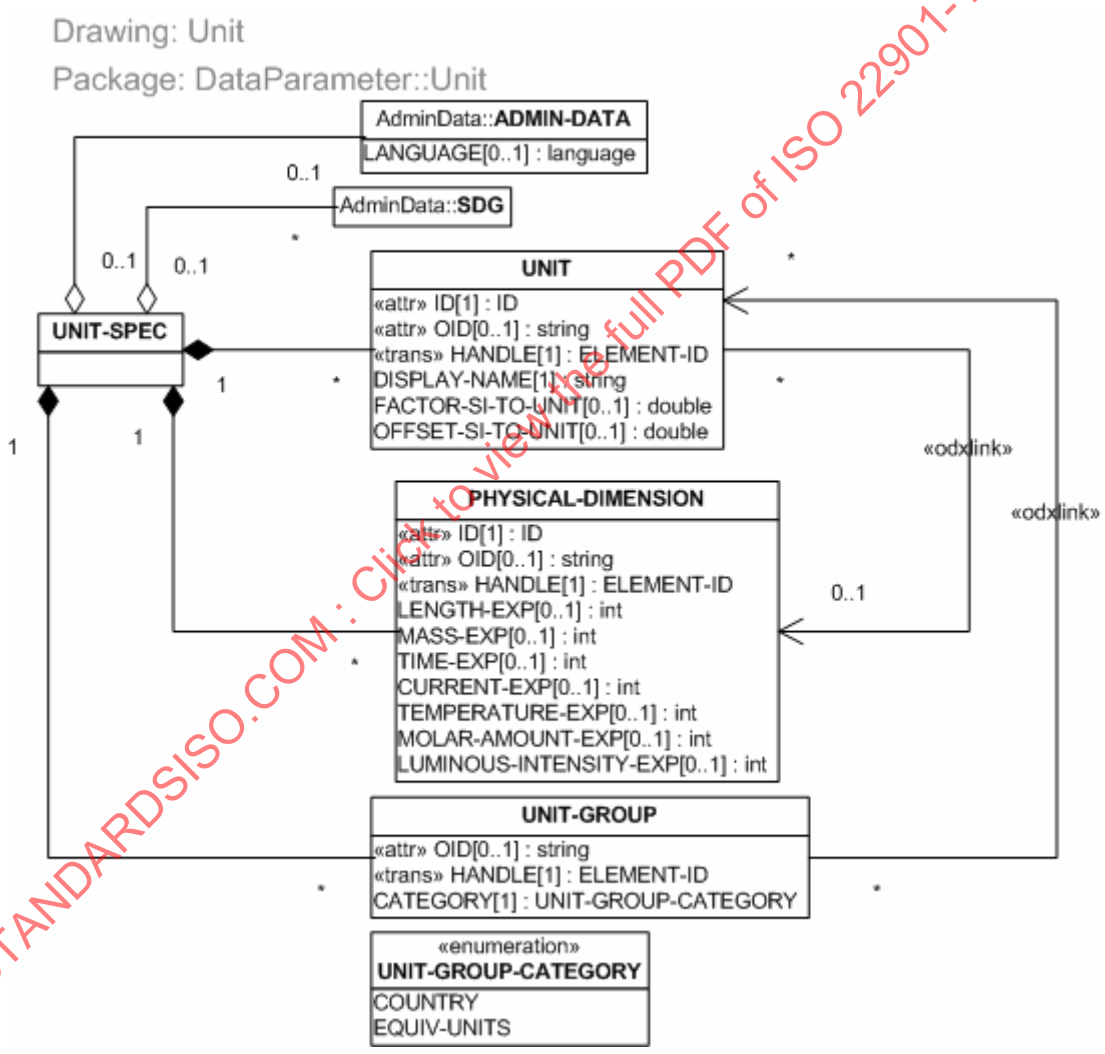


Figure 85 — UML representation of unit

In the model implemented, all units that might be defined should stem from SI units. In order to convert one unit into another, factor and offset are defined. For the calculation from SI-unit to the defined unit, the factor (FACTOR-SI-TO-UNIT) and the offset (OFFSET-SI-TO-UNIT) are applied, as follows:

UNIT = SI-UNIT * FACTOR-SI-TO-UNIT + OFFSET-SI-TO-UNIT

FACTOR-SI-TO-UNIT is the (value using SI unit) when (value using this unit) = 1.

OFFSET-SI-TO-UNIT is the (value using SI unit) when (value using this unit) = 0.

For the calculation from a unit to SI-unit, the reciprocal of the factor (FACTOR-SI-TO-UNIT) and the negation of the offset (OFFSET-SI-TO-UNIT) are applied, as follows:

SI-UNIT = (UNIT - OFFSET-SI-TO-UNIT) / FACTOR-SI-TO-UNIT

There is also a possibility of defining units without referencing a PHYSICAL-DIMENSION. In this case, FACTOR-SI-TO-UNIT and OFFSET-SI-TO-UNIT shall not be specified. Only the DISPLAY-NAME shall be given, which is used for displaying. It is recommended to avoid the definition of such units, though.

EXAMPLE Unit calculations:

cm (centimetre) = FACTOR-SI-TO-UNIT * (metre) = 0.01 m

km/h (kilometre per hour) = FACTOR-SI-TO-UNIT * m/s (metre per second) = 0.2777778 m/s

mph (miles per hour) = FACTOR-SI-TO-UNIT * m/s (metre per second) = 0.44704 m/s

Celsius = FACTOR-SI-TO-UNIT * K + OFFSET-SI-TO-UNIT = K + 273,15

The following XML example shows some definitions of units and the corresponding conversion from one unit to another.

```
<UNIT-SPEC>
    <UNITS>
        <UNIT ID = "Unit_Celsius">
            <SHORT-NAME>Unit_Celsius</SHORT-NAME>
            <DISPLAY-NAME>°C</DISPLAY-NAME>
        </UNIT>
        <UNIT ID = "Unit_Kmh">
            <SHORT-NAME>Unit_Kmh</SHORT-NAME>
            <DISPLAY-NAME>km/h</DISPLAY-NAME>
            <FACTOR-SI-TO-UNIT>0.27777777777777777777777777778</FACTOR-SI-TO-UNIT>
            <OFFSET-SI-TO-UNIT>0</OFFSET-SI-TO-UNIT>
            <PHYSICAL-DIMENSION-REF ID-REF = "PD_Velocity"/>
        </UNIT>
        <UNIT ID = "Unit_MilesPerHour">
            <SHORT-NAME>Unit_MilesPerHour</SHORT-NAME>
            <DISPLAY-NAME>mph</DISPLAY-NAME>
            <FACTOR-SI-TO-UNIT>0.44704</FACTOR-SI-TO-UNIT>
            <OFFSET-SI-TO-UNIT>0</OFFSET-SI-TO-UNIT>
            <PHYSICAL-DIMENSION-REF ID-REF = "PD_Velocity"/>
        </UNIT>
    </UNITS>
    <PHYSICAL-DIMENSIONS>
        <PHYSICAL-DIMENSION ID = "PD_Temp">
            <SHORT-NAME>PD_Temp</SHORT-NAME>
            <TEMPERATURE-EXP>1</TEMPERATURE-EXP>
        </PHYSICAL-DIMENSION>
        <PHYSICAL-DIMENSION ID = "PD_Velocity">
            <SHORT-NAME>PD_Velocity</SHORT-NAME>
            <LONG-NAME>Metre Per Second</LONG-NAME>
            <LENGTH-EXP>1</LENGTH-EXP>
            <TIME-EXP>-1</TIME-EXP>
        </PHYSICAL-DIMENSION>
    </PHYSICAL-DIMENSIONS>
</UNIT-SPEC>
```

For the conversion of a quantity with value x and unit “km/h” into the same quantity with value y and unit “mph” via SI-UNIT (m/s), the following equations result:

x km/h $= x \times 0{,}277\ 777\ 8$ m/s (step 1)

$x \times 0{,}277\ 777\ 8$ m/s / $0{,}447\ 04 = y$ mph (step 2)

resulting in

x km/h $= 0{,}621\ 4 \times x = x$ mph $= y$ mph

There is also a unit “Unit_Celsius” in this example, which does not reference a PHYSICAL-DIMENSION. If a DATA-OBJECT-PROP refers to this unit, the physical value, extracted using this DOP, is simply displayed with the string “°C” behind it.

7.3.6.7.2 Unit groups

Units can be grouped with the help of UNIT-GROUP. Each UNIT-GROUP may refer to an arbitrary number of UNITs. The member CATEGORY of the UNIT-GROUP denotes the unit system that the UNIT-GROUP's units are associated to. In this way, country-specific unit systems (CATEGORY=“COUNTRY”) can be defined as well as specific unit systems for certain application domains. For example, we can define a group of equivalent units, which are used in different countries, by setting CATEGORY=“EQUIV-UNITS”. For example, KmPerHour and MilesPerHour could be combined to one group of this category named “vehicle_speed”. The unit MetrePerSec would not belong to this group because it is normally not used for vehicle speed. But all of the mentioned units could be combined to one group named “speed”.

Figure 86 — UNIT-GROUP example shows the relation between UNIT-GROUP (CATEGORY=COUNTRY), UNIT and UNIT-GROUP (CATEGORY=EQUIVUNITS).

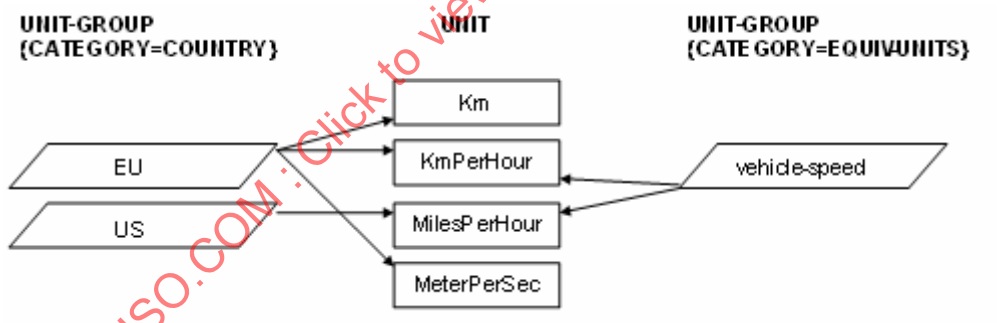


Figure 86 — UNIT-GROUP example

An ODX tool can use information stored in UNIT-GROUPS, e.g. to present a list of the available unit categories to the user; then, the user can select the appropriate unit system from this list and the following computations would be performed using the units belonging to this system.

A UNIT-GROUP always refers to UNITs that are defined in the same UNIT-SPEC as the UNIT-GROUP itself.

EXAMPLE

```
<UNIT-SPEC>
    <UNIT-GROUPS>
        <UNIT-GROUP>
            <SHORT-NAME>europe_unitgroup</SHORT-NAME>
            <CATEGORY>COUNTRY</CATEGORY>
            <UNIT-REFS>
                <UNIT-REF ID-REF = "Unit_Metre"/>
                <UNIT-REF ID-REF = "Unit_Kmh"/>
```

```
            </UNIT-REFS>
        </UNIT-GROUP>
        <UNIT-GROUP>
            <SHORT-NAME>usa_unitgroup</SHORT-NAME>
            <CATEGORY>COUNTRY</CATEGORY>
            <UNIT-REFS>
                <UNIT-REF ID-REF = "Unit_Foot"/>
                <UNIT-REF ID-REF = "Unit_Mph"/>
            </UNIT-REFS>
        </UNIT-GROUP>
        <UNIT-GROUP>
            <SHORT-NAME>VehicleSpeed</SHORT-NAME>
            <LONG-NAME>Vehicle Speed</LONG-NAME>
            <CATEGORY>EQUIV-UNITS</CATEGORY>
            <UNIT-REFS>
                <UNIT-REF ID-REF = "Unit_Kmh"/>
                <UNIT-REF ID-REF = "Unit_Mph"/>
            </UNIT-REFS>
        </UNIT-GROUP>
    </UNIT-GROUPS>
</UNIT-SPEC>
```

7.3.6.8 DTC data object property

Figure 87 — UML representation of DTC data object property shows the interpretation of DTCs (diagnostic trouble codes) received from the ECU is a core functional task during diagnostics. The D-server usually offers the detailed information for each received DTC. In addition, environment data can also be given for a DTC. As a consequence, the classes DTC-DOPs, ENV-DATA and ENV-DATA-DESC are introduced for the handling of DTCs.

A DTC-DOP is simple DOP with data extraction parameters (DIAG-CODED-TYPE), a conversion method (COMPU-METHOD), and a result type (PHYSICAL-TYPE). In addition, it provides a set of DTC objects, which contain the TROUBLE-CODE itself, a TEXT with an error message for this DTC and a LEVEL.

In an early stage of the ODX document creation a DTC may be marked with the attribute IS-TEMPORARY. These temporary DTCs only serve for documentation purposes during the coordination between manufacturer and supplier. They should exist only during the design phase of the ODX data. A D-server should ignore all temporary DTCs.

The value of TROUBLE-CODE is an unsigned integer number (A_UINT32) that is used for matching (see “Processing of DTC-DOP” below). As a consequence, the BASE-DATA-TYPE member of PHYSICAL-TYPE shall be set to A_UINT32.

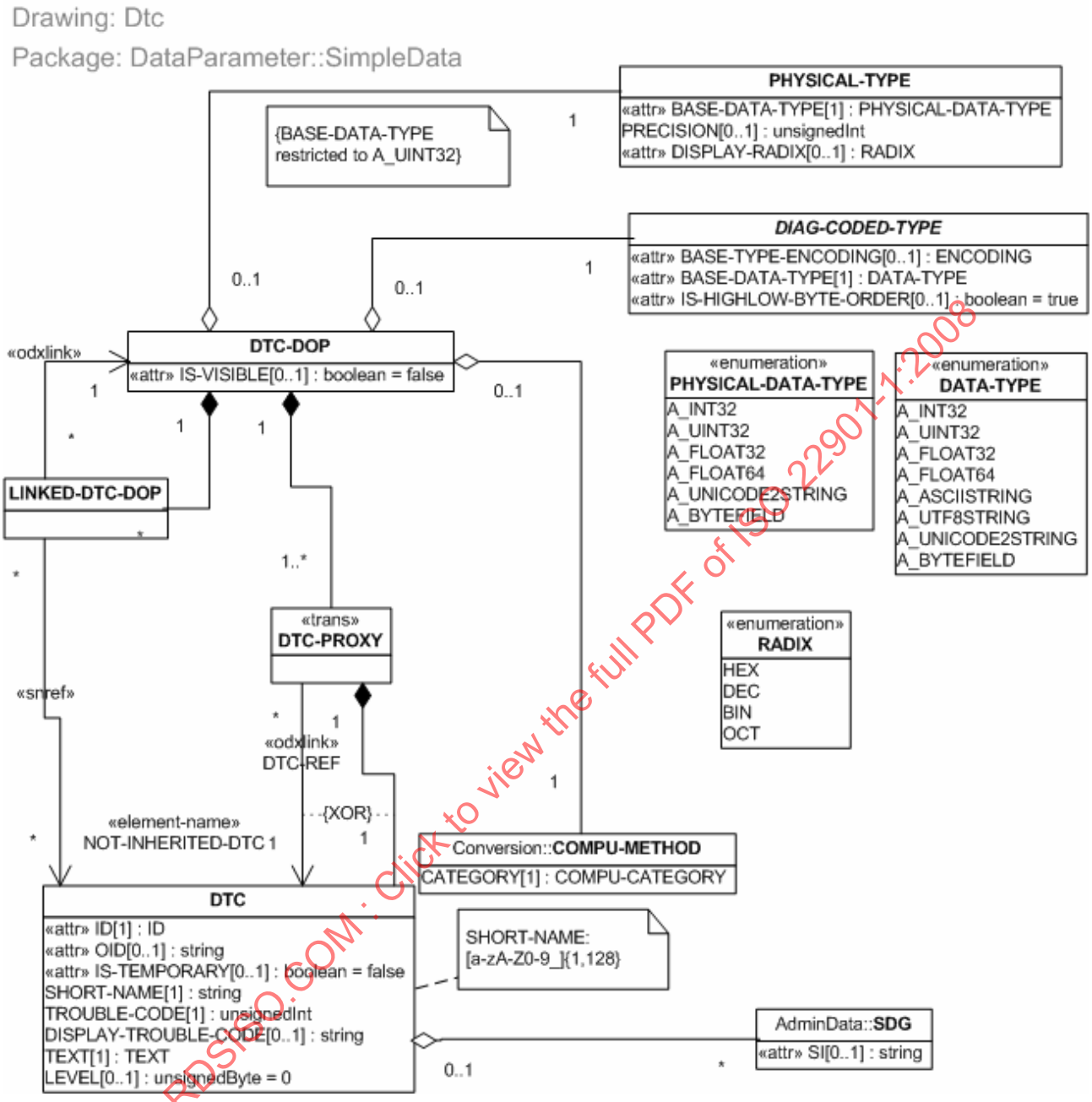


Figure 87 — UML representation of DTC data object property

The LEVEL is used to filter the DTCs. Its value domain is 0...255. The default value for a DTC level is 0. It may for example relate to the DTCSeverity in ISO 14229. The D-server can provide a method to request DTCs of a given level range. The method has two parameters for the closed interval range of the DTC levels to be returned. E.g. [10...200] returns all DTCs with levels set to numbers between 10 and 200 inclusively. Therefore, the interval [0…255] will contain all defined DTCs. A DISPLAY-TROUBLE-CODE (for example the SAE code) can be used to specify the display value for this DTC if it differs from the numeric TROUBLE-CODE. DISPLAY-RADIX is handled in the same manner as for DATA-OBJECT-PROPs.

Specific information like error set conditions may be added as SDGs (see 7.1.2.1 for a more detailed description of SDGs).

To provide the reuse of DTC objects, DTC-DOP may reference to DTCs that are contained in other diagnostic layers (see explanation below).

For better understanding, the DTC-DOP can be visualized as a global table of error codes (see Table 14 — Table visualisation of DTC-DOP data).

Table 14 — Table visualisation of DTC-DOP data

SHORT-NAME	TROUBLE-CODE	DISPLAY-TROUBLE-CODE	TEXT	LEVEL (1-255)
DTC_0120	288		Throttle Position Malf.	5
DTC_0130	304	B0130	Sensor Circuit Malf.	1
…	…	…	…	…

Figure 88 — Structure of service ReadDiagnosticTroubleCodesByStatus (0x18) of ISO 14230-3 shows the structure of the service ReadDiagnosticTroubleCodesByStatus (Service Id = 0x18) of ISO 14230-3. The service consists of a request and a response. Each of them consists of several parameters on its part. The first parameter of the response is the Service Id + 0x40 the second gives the number of DTCs. The parameter "ListOfDTCAndStatus" refers to an END-OF-PDU-FIELD, which indicates, that the referenced STRUCTURE is repeated until the end of the PDU. This structure contains two parameters. One of them refers to a DTC-DOP.

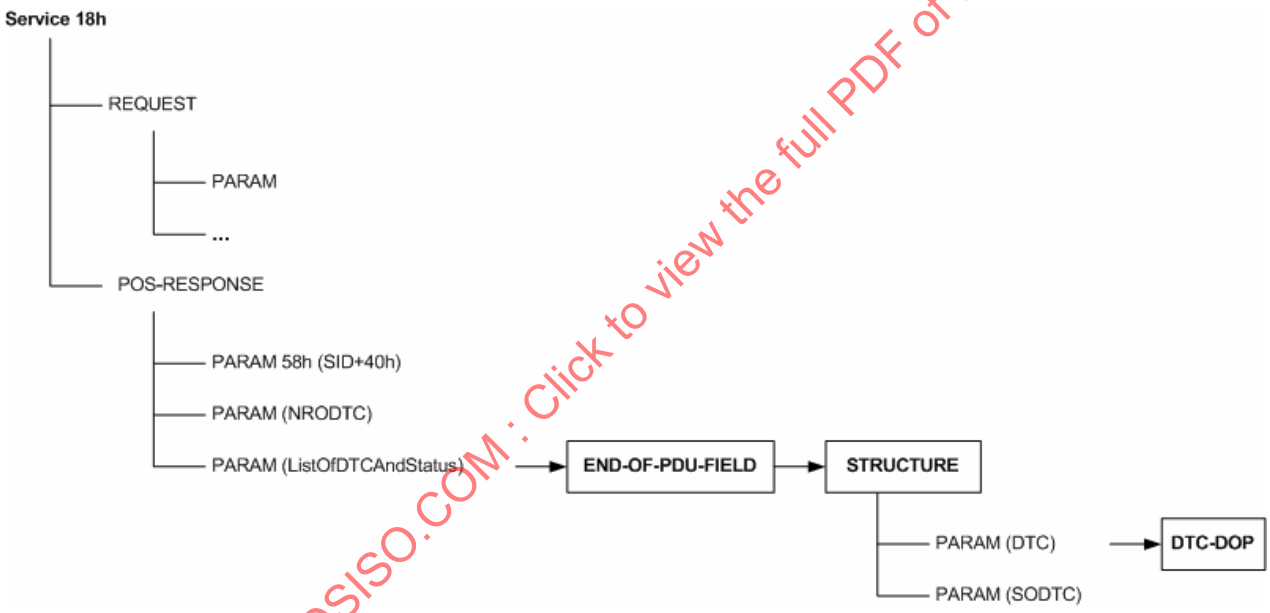


Figure 88 — Structure of service ReadDiagnosticTroubleCodesByStatus (0x18) of ISO 14230-3

Figure 89 — Processing of DTC-DOP shows the situation if a response parameter refers to a DTC-DOP that the D-server shall extract and convert the value of DTC in the same manner as other response parameters using DATA-OBJECT-PROP. The BASE-DATA-TYPE of PHYSICAL-TYPE is restricted to A_UINT32. The physical DTC value, i.e. the result of the execution of the COMPU-METHOD of the DTC-DOP, is then used as a switch-key (Figure 89, switch-key = 0120h). Afterwards, it is taken to find the matching DTC object by looking for a matching TROUBLE-CODE. A DTC object with TROUBLE-CODE = switch-key is finally displayed to the user.

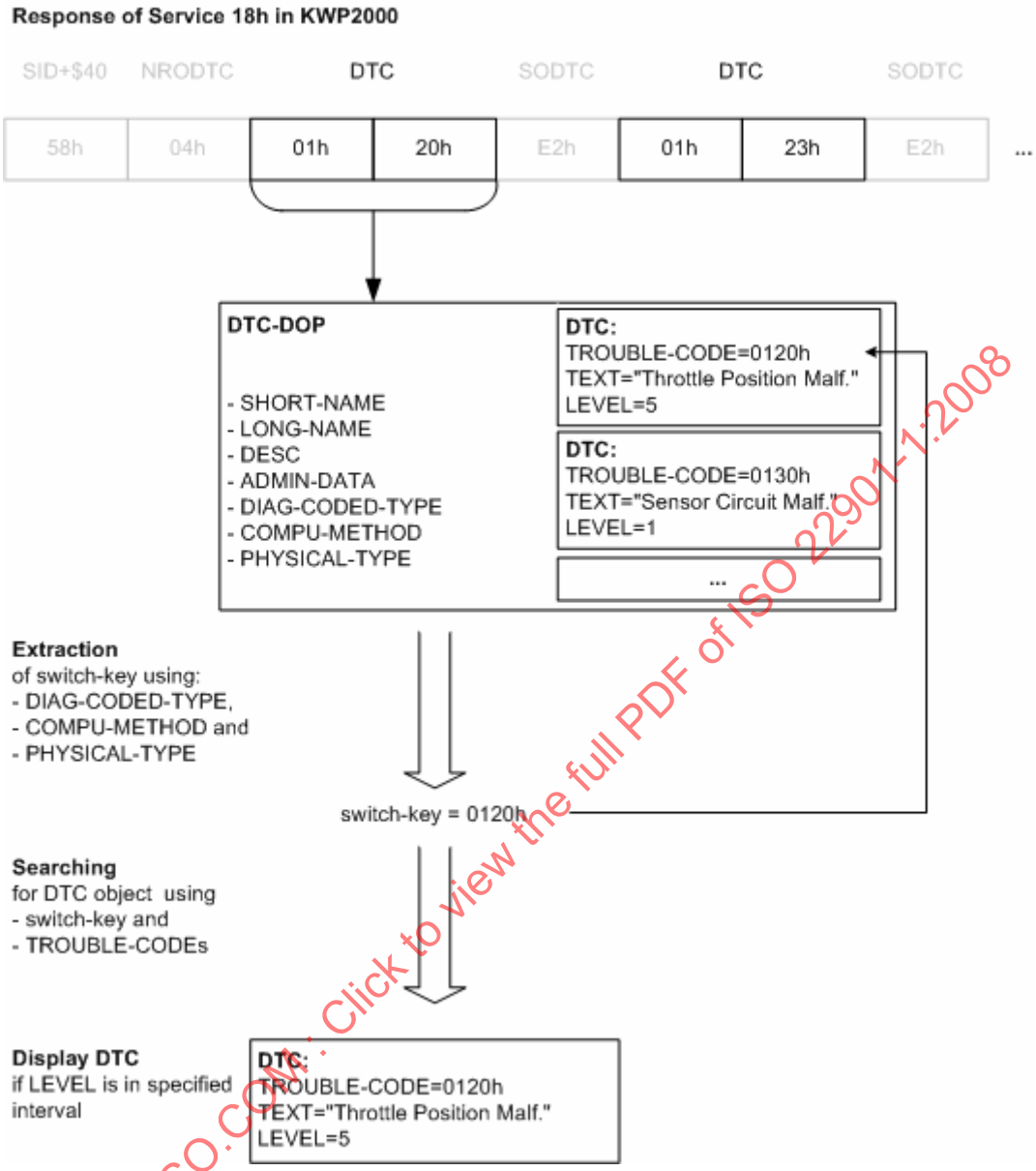


Figure 89 — Processing of DTC-DOP

EXAMPLE DTC-DOP in XML

```
<DTC-DOP ID = "DTCDOP_All">
    <SHORT-NAME>DTCDOP_All</SHORT-NAME>
    <DIAG-CODED-TYPE BASE-DATA-TYPE = "A_UINT32" xsi:type = "STANDARD-LENGTH-TYPE">
        <BIT-LENGTH>16</BIT-LENGTH>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE = "A_UINT32"/>
    <COMPU-METHOD>
        <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
    <DTCS>
        <DTC ID = "DTC0120">
            <SHORT-NAME>DTC0120</SHORT-NAME>
            <TROUBLE-CODE>288</TROUBLE-CODE>
            <TEXT>Throttle Position Malf.</TEXT>
            <LEVEL>5</LEVEL>
```

```
        </DTC>
        <DTC ID = "DTC0130">
            <SHORT-NAME>DTC0130</SHORT-NAME>
            <TROUBLE-CODE>304</TROUBLE-CODE>
            <DISPLAY-TROUBLE-CODE>B0130</DISPLAY-TROUBLE-CODE>
            <TEXT>Sensor Circuit Malf.</TEXT>
            <LEVEL>1</LEVEL>
        </DTC>
    </DTCS>
</DTC-DOP>
```

Figure 90 — Reuse of DTCs shows the DTC-DOPs which can be reused by means of inheritance or by referencing the objects in a ECU-SHARED-DATA layer. The DTC objects can also be reused. For example, if in a lower layer new DTCs should be added to the existing DTC list defined in the upper layer (DTC-DOP A in Figure 90), a new object shall be created in the lower layer (DTC-DOP B). This object references to the DTCs to be reused from the object A and defines the new DTCs that should be added. The advantage of this approach is that if certain DTCs are not to be used in the lower layer, they can be eliminated in an easy manner: not used DTCs are simply not referenced.

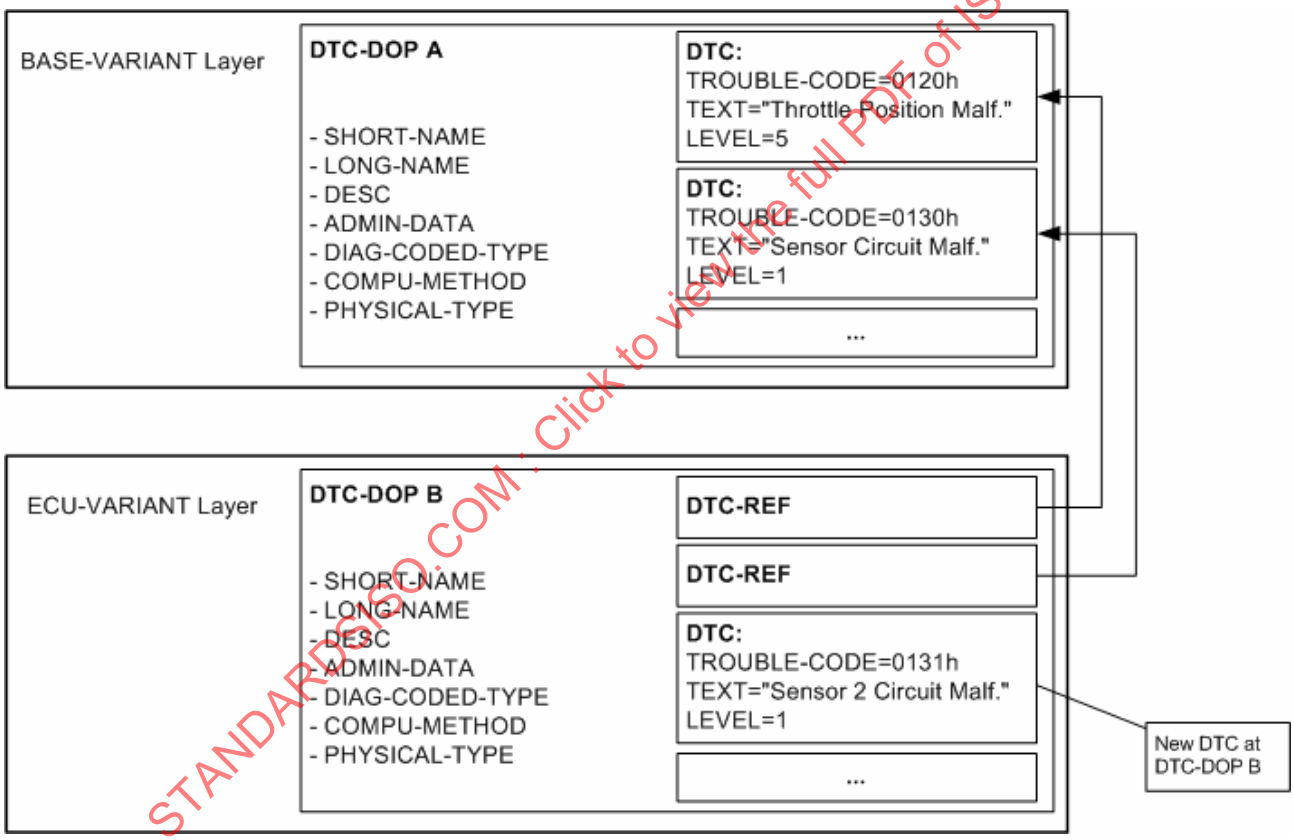


Figure 90 — Reuse of DTCs

A DTC-DOP may also be used in a REQUEST. A request parameter referencing a DTC-DOP provides the possibility to enter an existing DTC by TROUBLE-CODE. The COMPU-METHOD of the DTC-DOP is used to calculate the internal value which shall be passed to the ECU.

A further alternative to reuse DTC-DOPs and their containing DTCs is to work with LINKED-DTC-DOPs. This feature allows the use of categorised DTCs inside one DTC-DOP. That means a DTC-DOP might be defined which contains DTCs, describing for instance common behaviour and also a DTC-DOP might be defined which contains DTCs, describing for instance ECU specific behaviour in particular. This helps to prevent redundancy.

To cite an example the ODX-container for the base variant of an ECU “Engine Control Module” shall list all the DTCs, applicable for this ECU. This list of necessary DTCs is made up of common DTCs (that might be important for all ECUs) and DTCs only applicable for this ECU. In this example the DTC-DOP defined in the base variant contains all ECU specific DTCs. In an ECU-SHARED-DATA layer a DTC-DOP is defined that contains all common DTCs. To get those common DTCs in focus inside the DTC-DOP with the ECU applicable DTCs in the base variant of the “Engine Control Module” ECU, a LINKED-DTC-DOP reference shall be established within the DTC-DOP defining the ECU specific DTCs. Now all the common DTCs referenced via the LINKED-DTC-DOP are also visible inside the DTC-DOP with the ECU specific DTCs.

The most important circumstance with LINKED-DTC-DOPs is, that all listed DTCs within such a linked DTC-DOP are known in the layer, where the LINKED-DTC-DOP is established. The DTCs do not need to be referenced via a DTC-REF to use them.

The below mentioned XML code example will show the described issue. The listed ECU-SHARED-DATA container shows the DTC-DOP with the common DTCs.

```
<ECU-SHARED-DATA ID="ES_DataPool">
  <SHORT-NAME>ES_DataPool</SHORT-NAME>
  <LONG-NAME>Data Pool</LONG-NAME>
  <DIAG-DATA-DICTIONARY-SPEC>
    <DTC-DOPS>
      <DTC-DOP ID="DTCDOP_CommonDTCs">
        <SHORT-NAME>DTCDOP_CommonDTCs</SHORT-NAME>
        <LONG-NAME>Common DTCs</LONG-NAME>
        <DESC>
          <p>The following described DTCs are applicable for all ECUs</p>
        </DESC>
        <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
          <BIT-LENGTH>16</BIT-LENGTH>
        </DIAG-CODED-TYPE>
        <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
        <COMPU-METHOD>
          <CATEGORY>IDENTICAL</CATEGORY>
        </COMPU-METHOD>
        <DTCS>
          <DTC ID="DTC3004">
            <SHORT-NAME>DTC3004</SHORT-NAME>
            <TROUBLE-CODE>3004</TROUBLE-CODE>
            <DISPLAY-TROUBLE-CODE>P0BBC</DISPLAY-TROUBLE-CODE>
            <TEXT>Fault path for ACC Sensor Error</TEXT>
            <LEVEL>2</LEVEL>
          </DTC>
          <DTC ID="DTC0130">
            <SHORT-NAME>DTC0130</SHORT-NAME>
            <TROUBLE-CODE>304</TROUBLE-CODE>
            <DISPLAY-TROUBLE-CODE>B0130</DISPLAY-TROUBLE-CODE>
            <TEXT>Sensor Circuit Malf.</TEXT>
            <LEVEL>1</LEVEL>
          </DTC>
        </DTCS>
      </DTC-DOP>
    </DTC-DOPS>
  </DIAG-DATA-DICTIONARY-SPEC>
</ECU-SHARED-DATA>
```

The listed BASE-VARIANT container shows the DTC-DOP with the ECU specific DTCs and also shows the reference (via LINKED-DTC-DOP) to the common DTCs that are also important for this ECU.

```
<BASE-VARIANT ID="BV_EngineControlModule">
  <SHORT-NAME>BV_EngineControlModule</SHORT-NAME>
  <LONG-NAME>Engine control module</LONG-NAME>
  <DIAG-DATA-DICTIONARY-SPEC>
    <DTC-DOPS>
      <DTC-DOP ID="DTC_DOP_ListOfDTCsForEngineControlModule">
        <SHORT-NAME>DTC_DOP_ListOfDTCsForEngineControlModule</SHORT-NAME>
        <LONG-NAME>List of DTCs for engine control module</LONG-NAME>
        <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
          <BIT-LENGTH>16</BIT-LENGTH>
        </DIAG-CODED-TYPE>
        <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
        <COMPU-METHOD>
          <CATEGORY>IDENTICAL</CATEGORY>
        </COMPU-METHOD>
        <DTCS>
          <DTC ID="DTC_0120">
            <SHORT-NAME>DTC0120</SHORT-NAME>
            <TROUBLE-CODE>288</TROUBLE-CODE>
            <DISPLAY-TROUBLE-CODE>B0120</DISPLAY-TROUBLE-CODE>
            <TEXT>Throttle Position Malf.</TEXT>
            <LEVEL>5</LEVEL>
          </DTC>
        </DTCS>
        <LINKED-DTC-DOPS>
          <LINKED-DTC-DOP>
            <DTC-DOP-REF ID-REF="DTCDOP_CommonDTCs" DOCREF="ES_DataPool" DOCTYPE="LAYER"/>
          </LINKED-DTC-DOP>
        </LINKED-DTC-DOPS>
      </DTC-DOP>
    </DTC-DOPS>
  </DIAG-DATA-DICTIONARY-SPEC>
</BASE-VARIANT>
```

To make it clear, an application now shows two DTCs DTC3004, DTC0130 and the DTC0120.

It also could be possible that not all DTCs, made available with the DTC-DOP for the common DTCs via LINKED-DTC-DOP, are necessary for the ECU “Engine Control Module”. Therefore, the NOT-INHERITED-DTC-SNREF element can be used to exclude the unwanted DTCs from the common DTC data pool.

```
<LINKED-DTC-DOP>
    <NOT-INHERITED-DTC-SNREFS>
        <NOT-INHERITED-DTC-SNREF SHORT-NAME = "DTC3004"/>
    </NOT-INHERITED-DTC-SNREFS>
    <DTC-DOP-REF ID-REF = "DTCDOP_CommonDTCs" DOCREF = "ES_DataPool" DOCTYPE = "LAYER"/>
</LINKED-DTC-DOP>
```

An application now shows DTC0130 and the DTC0120 as the possible DTCs.

Regardless how the DTCs become part of the DTC-DOP, whether by local definition (DTC), by import (DTC-REF), or by inclusion of the DTCs from a linked DTC-DOP (LINKED-DTC-DOP), the SHORT-NAMEs of all DTCs shall be unique. It is not possible to override a DTC inside a referenced DTC-DOP by a local definition of DTC. Because the content of TROUBLE-CODE is used similar to a switch-case, the values of TROUBLE-CODEs shall be unique, too, except the DTC is marked as temporary.

A DTC-DOP may also be used in REQUESTs (see Table 6 —). Services $17 in KWP and Service $1906, $1910 are examples for services containing DTCs in a request. A VALUE referencing a DTC-DOP can be used to describe this in ODX. The 3D-System offers the possibility to enter the TROUBLE-CODE. The COMPU-METHOD of the DTC-DOP is used to calculate the coded value which shall be passed to the ECU in this case.

7.3.6.9 Environment data description

7.3.6.9.1 General

A DTC may be followed by environment data that describes the circumstances in which the error occurred [see service ReadStatusOfDTC (service id = 17h) of ISO 14230]. ENV-DATA-DESC is a complex DOP that is used to define the interpretation of environment data. Since environment data may be defined individually for each DTC, a reference to the response parameter PARAM-SNREF is used to indicate a switch-key e.g. the DTC in the PDU. This parameter shall reference a simple DOP in order to extract the switch-key value. The structure of the environment message data varies between different DTCs. Therefore, several ENV-DATA objects may be referenced. As several DTCs may share the same environment data structure, more than one DTC-VALUE may be given for one ENV-DATA if the corresponding DTCs have the same structure of environment data. The values shall be given in their physical representation.

The BIT-POSITION of a PARAM using an ENV-DATA directly or via any kind of COMPLEX-DOP shall be equal to “0” because an ENV-DATA is a BASIC-STRUCTURE that starts always at byte edge.

In many cases a set of environment data is valid for all DTCs. To avoid duplicate specification of these common environmental data as PARAMs within all affected ENV-DATA objects, a separate ENV-DATA with a special flag ALL-VALUE is intended. In case all DTCs share at least some parts of the environment data, the ALL-VALUE element can be used to identify an ENV-DATA object as the common part of the environment data of all DTCs in a given ENV-DATA-DESC. The ALL-VALUE ENV-DATA is then expected at the beginning of the environment data message of all DTCs of this ENV-DATA-DESC and is evaluated first by the D-server. This is why there cannot be more than one ALL-VALUE element in a given ENV-DATA-DESC object. In a second step the specific DTC-VALUE’s ENV-DATA is extracted from the response message. Every DTC can only have one specific ENV-DATA object (other than the ALL-VALUE ENV-DATA) associated to it.

To provide reuse of ENV-DATA objects, an ENV-DATA-DESC may reference ENV-DATAs that are contained in the same or in other diagnostic layers in the valid data pool. See also Figure 95 and Figure 96.

Use of ENV-DATA-DESC in a FIELD means that the ENV-DATA is repeated in a response.

See Figure 91 — UML representation of ENV-DATA-DESC.

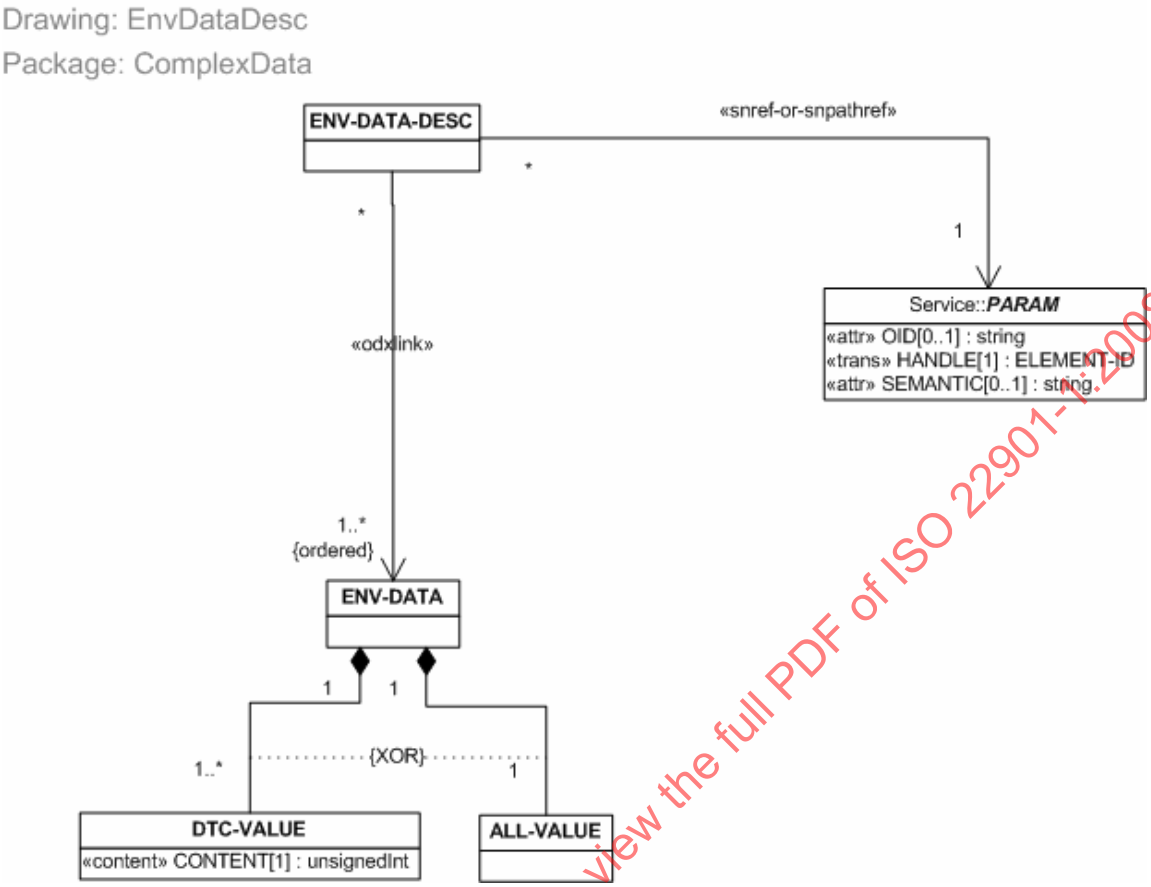


Figure 91 — UML representation of ENV-DATA-DESC

Figure 92 — Structure of service 0x17 in ISO 14230-3 shows the service ReadStatusOfDTC to access the environment data. The first parameter of the response is the Service Id + 0x40, the second gives the number of DTCs. The following parameters are the DTC, the status of DTC (SODTC) and the environment data.

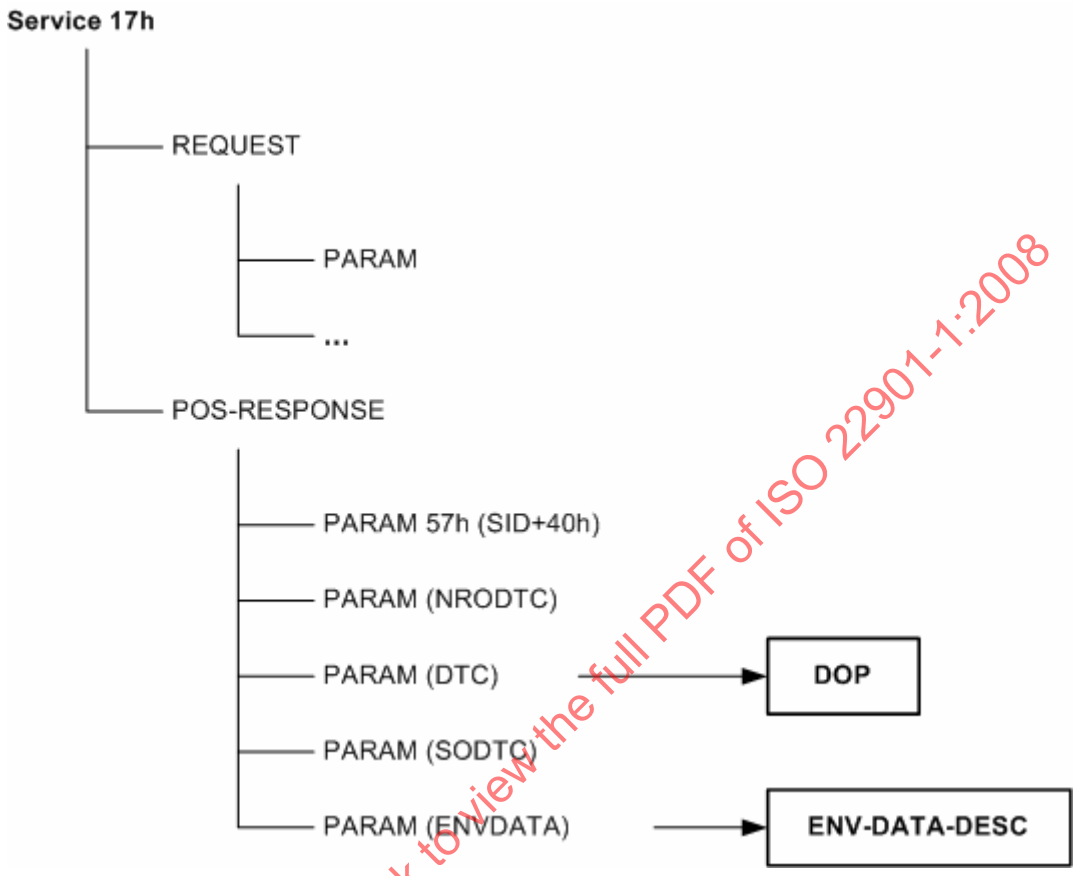


Figure 92 — Structure of service 0x17 in ISO 14230-3

Figure 93 — Processing of ENV-DATA-DESC illustrates the situation when the D-server detects a reference to ENV-DATA-DESC during the processing of the response, it shall determine the physical value from the response parameter referenced by PARAM-SNREF. This value is used as a switch-key and shall be of type A_UINT32 (PHYSICAL-TYPE.BASE-DATA-TYPE of the referenced DOP shall be equal to “A_UINT32”). The D-server shall check whether an ENV-DATA object with the flag ALL-VALUE exists. In this case, this ENV-DATA shall be used first to decode the environment data in the response message. Second, the D-server shall search for an ENV-DATA object with a DTC-VALUE equal to the physical value of the PARAM referenced by the ENV-DATA-DESC. This ENV-DATA object shall then be used to decode the remainder of the environment data in the response message.

For each ENV-DATA found, all contained PARAMs are processed in order of their BYTE- and BIT-POSITION. These positions are given relatively to the start position of the ENV-DATA-DESC in the PDU. If several PARAMs share the same BYTE- and BIT-POSITION, they are ordered according to the position in the ODX instance. This implies that the order of PARAMs within an ENV-DATA and of ENV-DATAs within an ENV-DATA-DESC shall not be changed by the application. In this way the author of the ODX document can define the order, which is later used to sort the data items for displaying. If no ENV-DATA with ALL-VALUE flag and no ENV-DATA with DTC-VALUE = switch-key are found, the D-server shall report an error message.

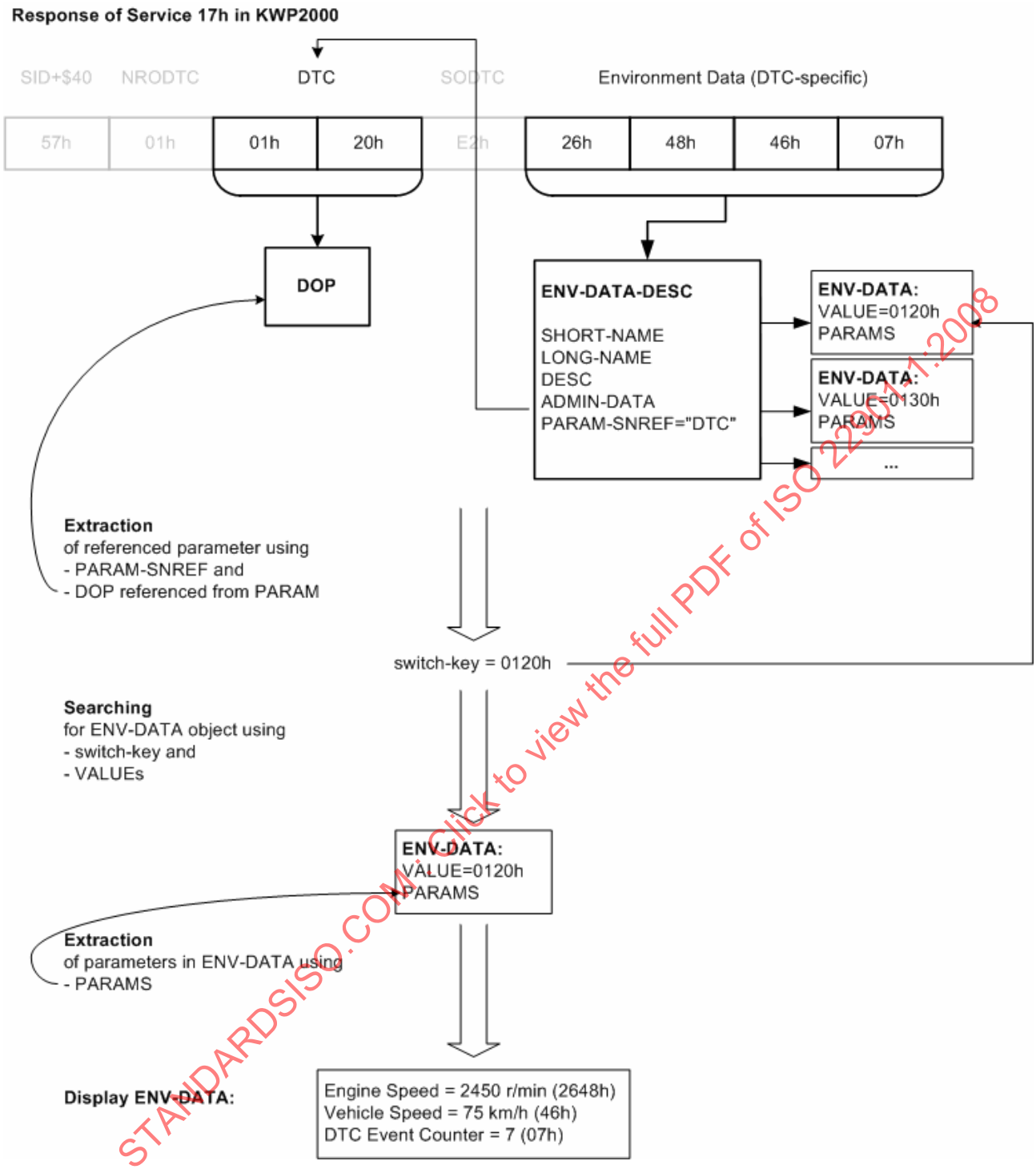


Figure 93 — Processing of ENV-DATA-DESC

EXAMPLE ENV-DATA-DESC in XML:

```
<DIAG-DATA-DICTIONARY-SPEC>

  <ENV-DATA-DESCS>
    <ENV-DATA-DESC ID="ENVDESC_EnvDataDesc1">
      <SHORT-NAME>ENVDESC_EnvDataDesc1</SHORT-NAME>
```

```
      <PARAM-SNREF SHORT-NAME="SwitchKeyDTC"/>
      <ENV-DATA-REFS>
        <ENV-DATA-REF ID-REF="ED_1"/>
        <ENV-DATA-REF ID-REF="ED_2"/>
        <ENV-DATA-REF ID-REF="ED_3"/>
        <ENV-DATA-REF ID-REF="ED_All"/>
      </ENV-DATA-REFS>
    </ENV-DATA-DESC>
  </ENV-DATA-DESCS>

  <ENV-DATAS>
    <ENV-DATA ID="ED_1">
      <SHORT-NAME>ED_DTC0120</SHORT-NAME>
      <PARAMS>
        <PARAM xsi:type="VALUE">
          <SHORT-NAME>EngineSpeed</SHORT-NAME>
          <BYTE-POSITION>0</BYTE-POSITION>
          <DOP-REF ID-REF="DOP_EngineSpeed"/>
        </PARAM>
        <PARAM xsi:type="VALUE">
          <SHORT-NAME>VehicleSpeed</SHORT-NAME>
          <BYTE-POSITION>2</BYTE-POSITION>
          <DOP-REF ID-REF="DOP_1ByteHex"/>
        </PARAM>
      </PARAMS>
      <DTC-VALUES>
        <DTC-VALUE>288</DTC-VALUE>
      </DTC-VALUES>
    </ENV-DATA>

    <ENV-DATA ID="ED_2">
      <SHORT-NAME>ED_DTC0130</SHORT-NAME>
      <PARAMS>
        <PARAM xsi:type="VALUE">
          <SHORT-NAME>EngineSpeed</SHORT-NAME>
          <BYTE-POSITION>0</BYTE-POSITION>
          <DOP-REF ID-REF="DOP_EngineSpeed"/>
        </PARAM>
        <PARAM xsi:type="VALUE">
          <SHORT-NAME>Voltage</SHORT-NAME>
          <BYTE-POSITION>2</BYTE-POSITION>
          <DOP-REF ID-REF="DOP_Voltage"/>
        </PARAM>
      </PARAMS>
      <DTC-VALUES>
        <DTC-VALUE>304</DTC-VALUE>
      </DTC-VALUES>
    </ENV-DATA>

    <ENV-DATA ID="ED_3">
      <SHORT-NAME>DTC2345_AND_DTC1234</SHORT-NAME>
      <PARAMS>
        <PARAM xsi:type="VALUE">
          <SHORT-NAME>Voltage</SHORT-NAME>
          <BYTE-POSITION>0</BYTE-POSITION>
          <DOP-REF ID-REF="DOP_Voltage"/>
        </PARAM>
        <PARAM xsi:type="VALUE">
```

```
          <SHORT-NAME>EngineSpeed</SHORT-NAME>
          <BYTE-POSITION>1</BYTE-POSITION>
          <DOP-REF ID-REF="DOP_EngineSpeed"/>
        </PARAM>
      </PARAMS>
      <DTC-VALUES>
        <DTC-VALUE>4660</DTC-VALUE>
        <DTC-VALUE>9029</DTC-VALUE>
      </DTC-VALUES>
    </ENV-DATA>

    <ENV-DATA ID="ED_All">
      <SHORT-NAME>ED_All</SHORT-NAME>
      <PARAMS>
        <PARAM xsi:type="VALUE">
          <SHORT-NAME>EventCounter</SHORT-NAME>
          <BYTE-POSITION>3</BYTE-POSITION>
          <DOP-REF ID-REF="DOP_1ByteHex"/>
        </PARAM>
      </PARAMS>
      <ALL-VALUE/>
    </ENV-DATA>
  </ENV-DATAS>

</DIAG-DATA-DICTIONARY-SPEC>
```

In the above example, three DTC-specific ENV-DATA objects are referenced (ID = “ED_1”, “ED_2” and “ED_3”). Environment data with ID=“ED_1” is defined for the DTC 0120h, with ID=“ED_2” for 0130h, and ENV-DATA with ID=“ED_3” is defined for two DTCs: 1234h and 2345h. In addition, an ENV-DATA is referenced that is valid for all DTCs (ID = ED_All). In this example, byte at the byte position 3 relatively to the start of the ENV-DATA-DESC is always an event counter, which gives how often the error has occurred.

7.3.6.9.2 Rules for authoring

Environmental data common for all DTCs should be specified inside a single ENV-DATA with the flag ALL-VALUE. All environmental data specific for one or more DTCs shall be specified inside a single ENV-DATA with an explicit specification of the corresponding DTCs. This means that at most one ENV-DATA with the flag ALL-VALUE may exist.

An ENV-DATA-DESC may only be referenced by a response parameter of a service whose DIAGNOSTIC-CLASS attribute indicates an error service (DIAGNOSTIC-CLASS=“FAULTREAD”). Furthermore, the response shall contain exactly one parameter whose SHORT-NAME matches the SHORT-NAME attribute of PARAM-SNREF in the ENV-DATA-DESC object.

7.3.6.9.3 Reuse of ENV-DATA objects

To enable reuse of ENV-DATA objects, ENV-DATA-DESC only holds references to ENV-DATA objects. New ENV-DATA objects shall be created and added to the collection ENV-DATAS in the DIAG-LAYER-CONTAINER and subsequently be referenced from an ENV-DATA-DESC. Unreferenced ENV-DATAs in ENV-DATAS may exist and may be stored in the ODX file for later use or for data exchange.

7.3.6.10 Complex data (complex data object property)

7.3.6.10.1 Overview

Complex diagnostic data object properties (Complex DOPs) are used to define and interpret complex ECU responses (or structured parts of a tester request) containing multiple data items, which are grouped in a structure or repeated in a field. In addition, there is a multiplexer, which is used to interpret alternative data depending on a switch-key. They are needed to describe ECU responses which are dynamic, i.e. the size and actual structures that are known only at runtime.

Like the simple DOPs, complex DOPs have the standard elements SHORT-NAME, LONG-NAME, DESC, ID, SDGS, and ADMIN-DATA. Most complex DOPs own the attribute IS-VISIBLE. The latter can be set to “true” or “false”. Set to “true”, this attribute causes the diagnostic application to show the structural information given by this complex DOP. Otherwise, it is not shown.

NOTE The D-server does not evaluate this attribute, but passes it to the diagnostic application only.

Complex DOPs are referenced from a PARAM of a response in the same way like simple DOPs. In contrast to simple DOPs, the complex DOPs usually contain directly or indirectly further PARAMs, which can also refer to a further complex DOP. Because infinite recursion, e.g. by self-referencing, is forbidden, every branch shall terminate in a PARAM leaf which references DATA-OBJECT-PROP or DTC-DOP. The BYTE- and BIT-POSITIONs of a PARAM are given there as absolute positions in the PDU if the PARAM is directly part of a POS-RESPONSE, NEG-RESPONSE, GLOBAL-NEG-RESPONSE or REQUEST. In contrast to that, the BYTE-POSITION of a PARAM used inside a complex DOP is always given relatively to the start of this complex DOP (see Figure 94 — Absolute and relative byte position). The complex DOP it self shall start at a byte edge, i.e. the BIT-POSITION of the referencing parameter shall be 0 or absent. In a description of an ECU response a PARAM of type VALUE at absolute byte position 2 is defined that refers to a complex DOP of type STRUCTURE (description will follow below). Inside the STRUCTURE two PARAM at relative byte positions 1 and 3 are defined.

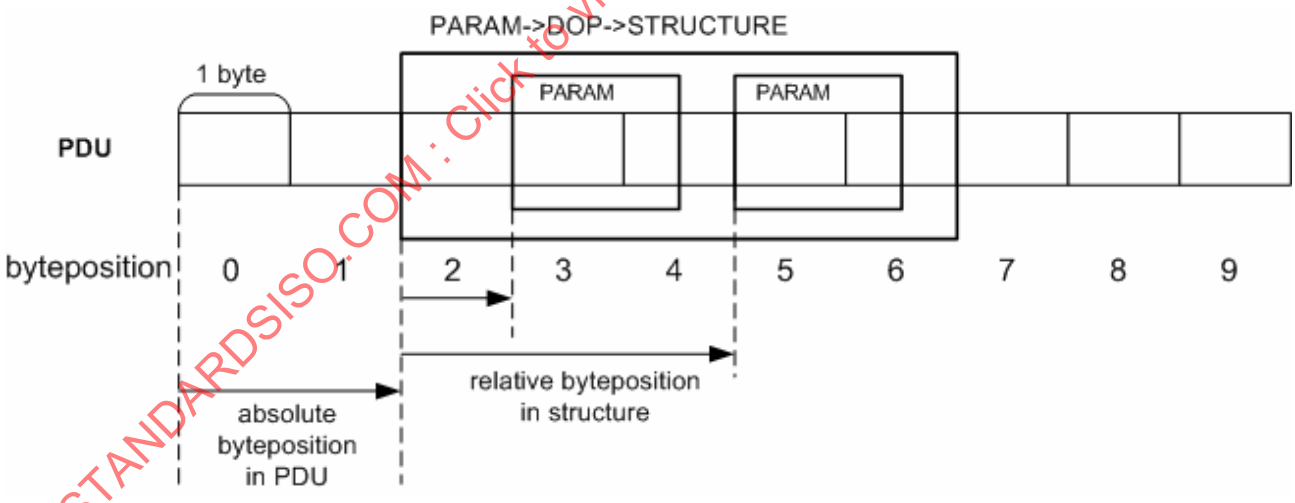


Figure 94 — Absolute and relative byte position

Most classes derived from COMPLEX-DOP are allowed to be used only inside a response description. Only STRUCTURE and END-OF-PDU-FIELD are usable directly or indirectly inside a REQUEST.

See Figure 95 — UML representation of complex data.

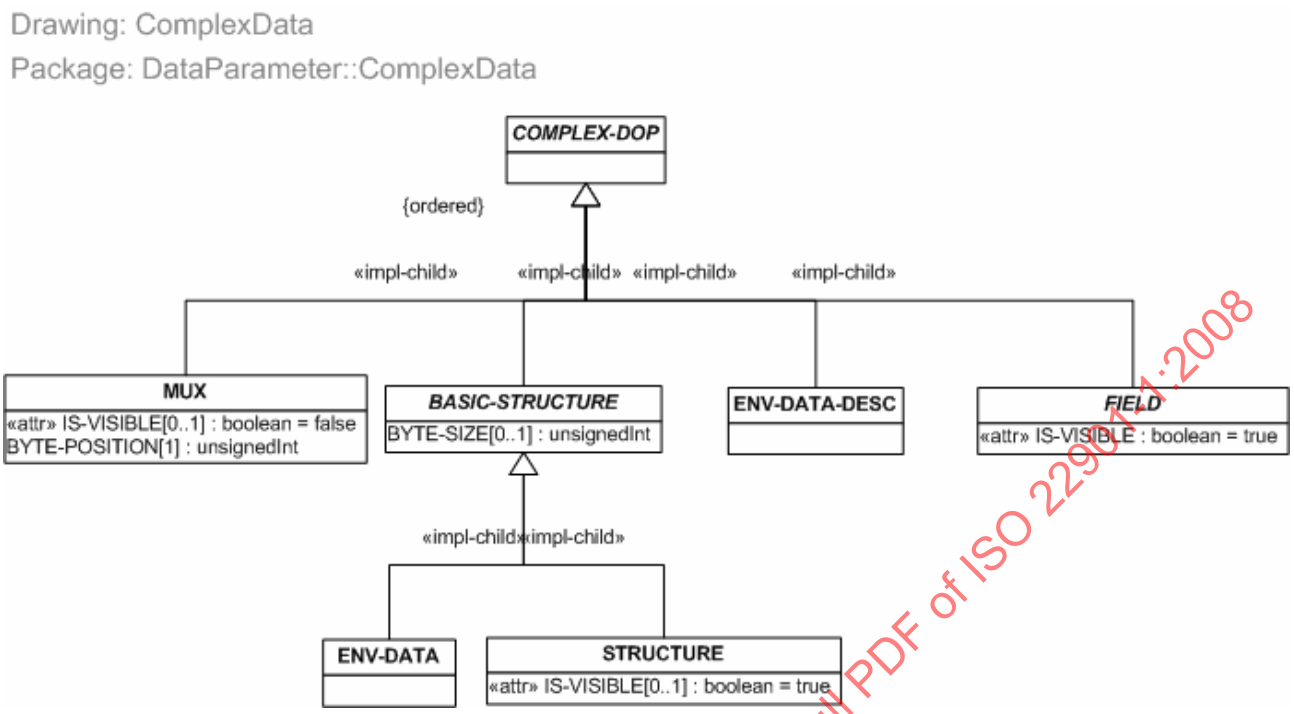


Figure 95 — UML representation of complex data

7.3.6.10.2 to 7.3.6.10.7 describe all types of complex DOPs in detail.

7.3.6.10.2 Structure

Figure 96 — UML representation of structure illustrates a structure (STRUCTURE) which is a kind of wrapper to enable recursion that combines several PARAMs into a group of parameters belonging together. This is possible for REQUEST and RESPONSE. Beside information structuring, the use of STRUCTUREs enables an easy reuse of parameter groups. The BIT-POSITION of a PARAM using a STRUCTURE directly or via any kind of COMPLEX-DOP shall be absent or equal to “0”, i.e. a STRUCTURE starts always at byte edge. For example, the status of DTC might be defined as a STRUCTURE containing a DTC Warning Lamp Calibration Status, a DTC Storage Status, a DTC Readiness Flag and a DTC Fault Symptom. In ISO 14230, this structure can then be used both at service 0x17 and at service 0x18.

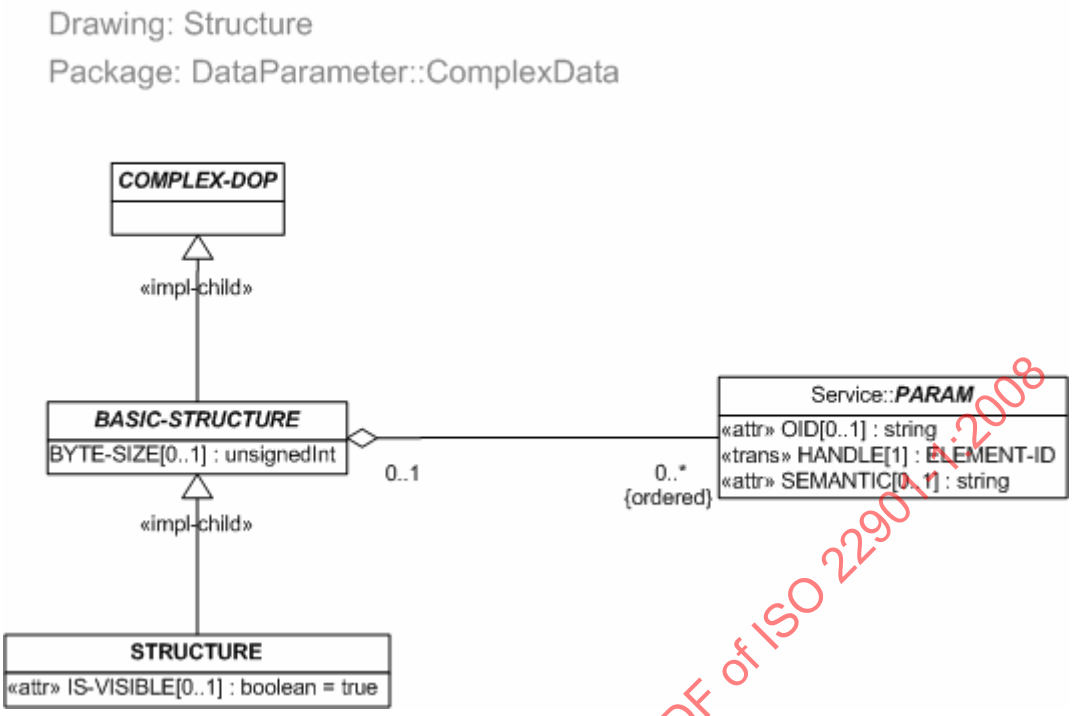


Figure 96 — UML representation of structure

In general, one or more PARAMs are defined in the same way as within a response inside a STRUCTURE. Each PARAM represents an item which is a member of the group built by this STRUCTURE. But in some use cases, the existence of parameters depends on the actual variant of the ECU. For example, no parameter is available inside the base variant, but inside specific variants. This can be specified by a SHORT-NAME reference to a STRUCTURE without any parameters inside the BASE-VARIANT, which is overridden inside the ECU-VARIANT by a STRUCTURE with the actual parameters available in this variant.

The optional attribute BYTE-SIZE gives the size of the whole structure in bytes. It shall not be less than the total size of the included parameters. It can be greater than the total size of the included items if the developer aims to create space after the last item. If the STRUCTURE has any dynamic components (e.g. MUX or any FIELD with dynamic length) the BYTE-SIZE should not be given.

Since a complex DOP can contain PARAMs, which themselves can use complex DOPs, it is possible to define complex data structures recursively. The flag IS-VISBLE is used to define if the STRUCTURE that is used as a wrapper should be visible at the diagnostic application. For example, if a STRUCTURE S2 consists of parameter PARAM_2A, a parameter PARAM_2B and the attribute IS-VISIBLE="true" this results in a structure of name S2 at the application interface (that contains "2A" and "2B"). If the attribute IS-VISIBLE="false" is set, no structure but simply two result parameters "2A" and "2B" are returned.

Figure 97 — Recursive definition of data structures using STRUCTURE illustrates an example about recursive definition of data structures using STRUCTURE.

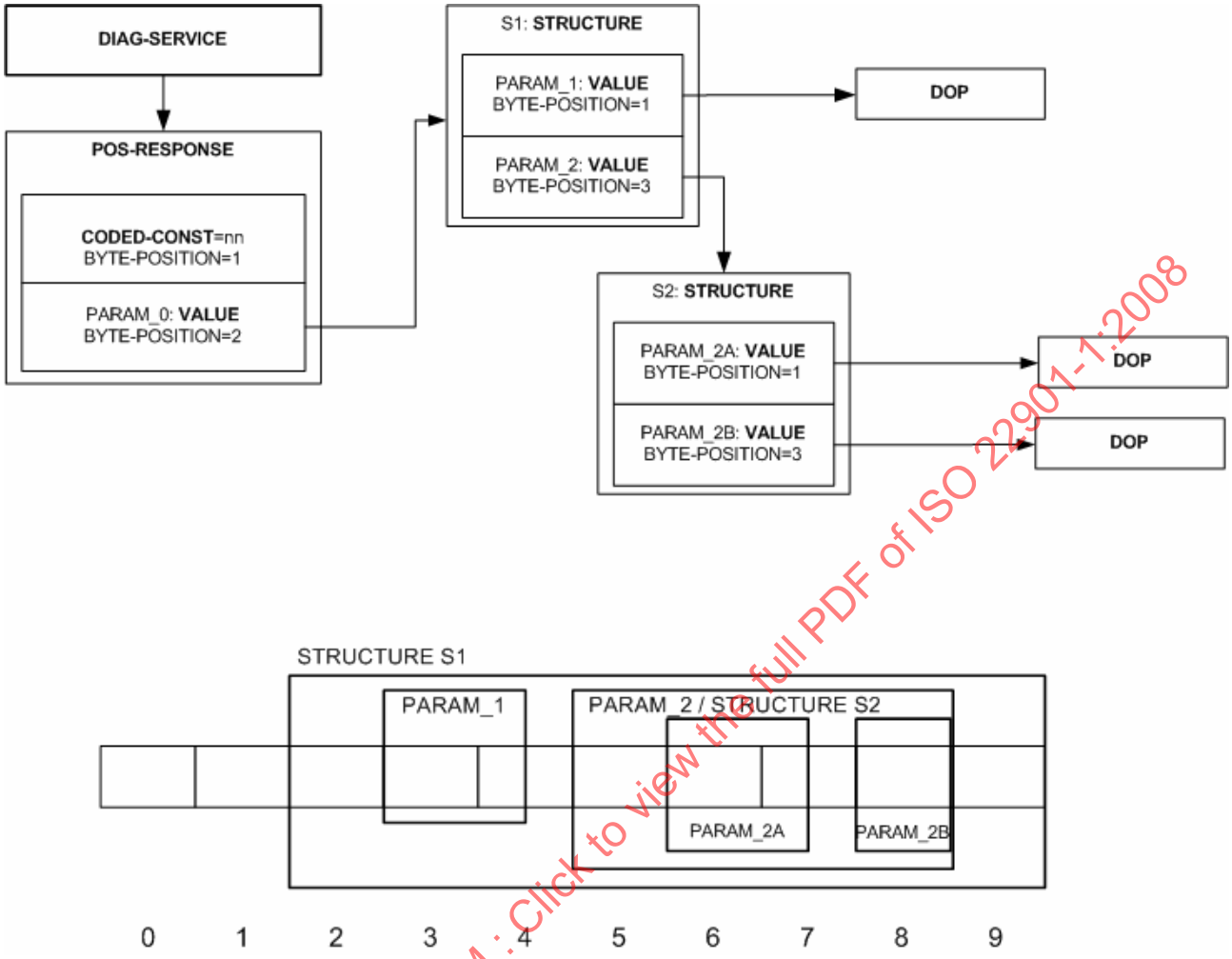


Figure 97 — Recursive definition of data structures using STRUCTURE

In XML, the structure S1 appears as follows:

```
<STRUCTURE ID = "STRUCT_S1" IS-VISIBLE = "false">
    <SHORT-NAME>S1</SHORT-NAME>
    <BYTE-SIZE>8</BYTE-SIZE>
    <PARAMS>
        <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
            <SHORT-NAME>PARAM_1</SHORT-NAME>
            <BYTE-POSITION>1</BYTE-POSITION>
            <DOP-REF ID-REF = "DOP-ID"/>
        </PARAM>
        <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
            <SHORT-NAME>PARAM_2</SHORT-NAME>
            <BYTE-POSITION>3</BYTE-POSITION>
            <DOP-REF ID-REF = "STRUCT_S2"/>
        </PARAM>
    </PARAMS>
</STRUCTURE>
```

EXAMPLE The following structure implements the example mentioned above. It represents the status of DTC in ISO 14230 containing a DTC Warning Lamp Calibration Status, a DTC Storage Status, a DTC Readiness Flag and DTC Fault Symptom in one byte.

```
<STRUCTURE ID = "STRUCT_SODTC" IS-VISIBLE = "false">
    <SHORT-NAME>SODTC</SHORT-NAME>
    <LONG-NAME>Status of DTC</LONG-NAME>
    <BYTE-SIZE>1</BYTE-SIZE>
    <PARAMS>
        <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
            <SHORT-NAME>DTC_WLCS</SHORT-NAME>
            <LONG-NAME>DTC Warning Lamp Calibration Status</LONG-NAME>
            <BYTE-POSITION>0</BYTE-POSITION>
            <BIT-POSITION>7</BIT-POSITION>
            <DOP-REF ID-REF = "DOP_DTC_WLCS"/>
        </PARAM>
        <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
            <SHORT-NAME>DTC_SS</SHORT-NAME>
            <LONG-NAME>DTC Storage Status</LONG-NAME>
            <BYTE-POSITION>0</BYTE-POSITION>
            <BIT-POSITION>5</BIT-POSITION>
            <DOP-REF ID-REF = "DOP_DTC_SS"/>
        </PARAM>
        <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
            <SHORT-NAME>DTC_RF</SHORT-NAME>
            <LONG-NAME>DTC Readyness Flag</LONG-NAME>
            <BYTE-POSITION>0</BYTE-POSITION>
            <BIT-POSITION>4</BIT-POSITION>
            <DOP-REF ID-REF = "DOP_DTC_RF"/>
        </PARAM>
        <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
            <SHORT-NAME>DTC_FS</SHORT-NAME>
            <LONG-NAME>DTC Fault Symptom</LONG-NAME>
            <BYTE-POSITION>0</BYTE-POSITION>
            <BIT-POSITION>0</BIT-POSITION>
            <DOP-REF ID-REF = "DOP_DTC_FS"/>
        </PARAM>
    </PARAMS>
</STRUCTURE>
```


7.3.6.10.3 Static field

Figure 98 — UML representation of field illustrates static fields (STATIC-FIELD) which are used when the PDU contains a recurring structure and the number of repetitions is fixed and does not have to be determined dynamically. In other words, a STATIC-FIELD is used to describe a repetition of a BASIC-STRUCTURE or ENV-DATA-DESC with an a priori fixed number of repetitions. The meaning of the reference to ENV-DATA-DESC is that the actually selected ENV-DATA out of the ENV-DATAs the ENV-DATA-DESC references to is taken for the repetitions.

The flag IS-VISIBLE is used to define if the list itself should be visible at the diagnostic application. Set to “true”, this attribute causes the diagnostic application to show the structural information. Otherwise, it is not shown.

NOTE The D-server does not evaluate this attribute but passes it to the diagnostic application only.

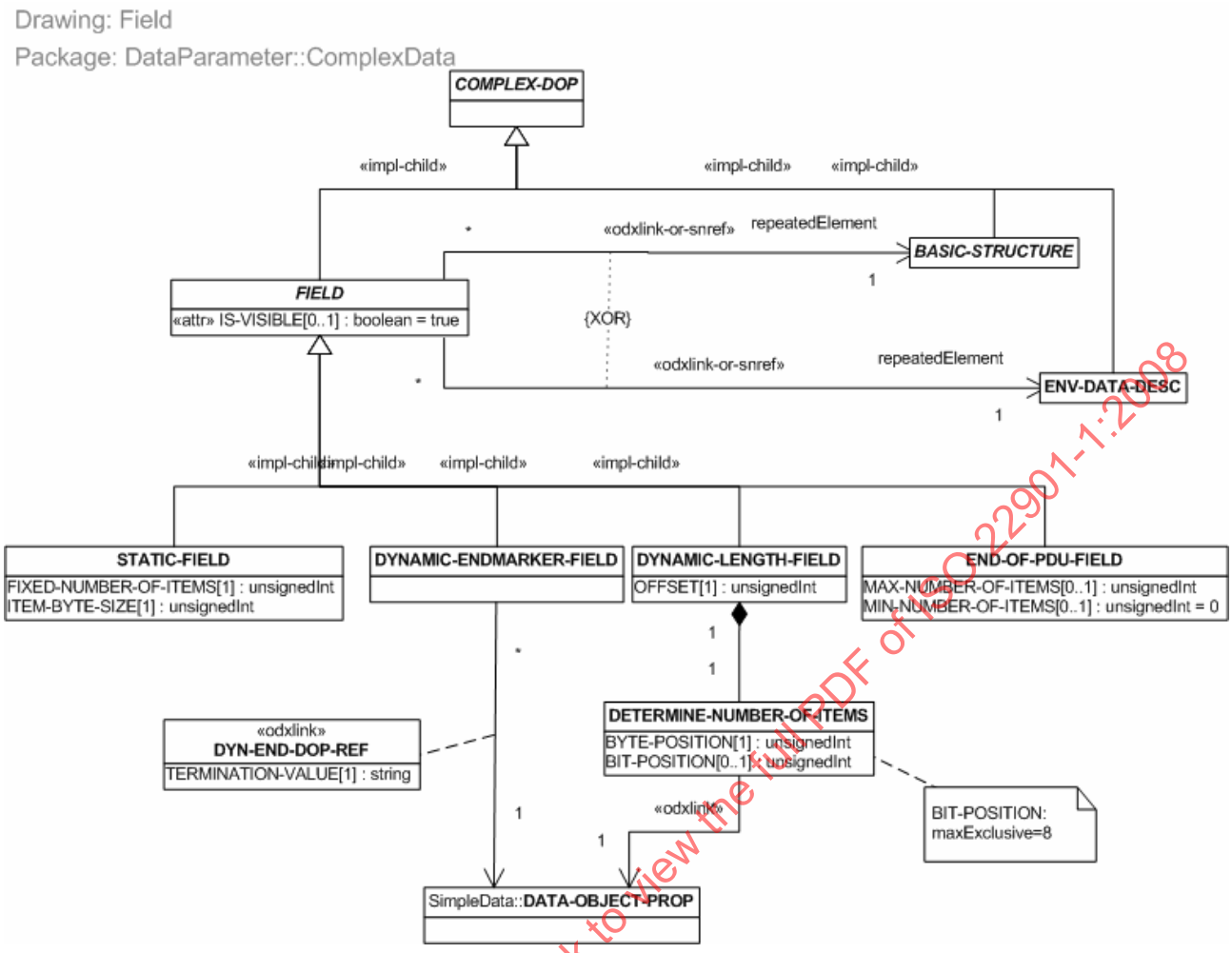


Figure 98 — UML representation of field

The number of repetitions is given via the attribute FIXED-NUMBER-OF-ITEMS. The reference (by «odxlink» or «snref») to the BASIC-STRUCTURE or ENV-DATA-DESC defines the complex DOP to be repeatedly applied. The first item starts at the same byte position as the STATIC-LIST. For each further item, the byte position is increased by ITEM-BYTE-SIZE which determines the length of one item in the field. It is given in bytes and shall not be less than the length occupied by the PARAMs included by the repeated complex DOP or the optional BYTE-SIZE of the BASIC-STRUCTURE referenced to.

IMPORTANT — It shall be ensured that the items shall not exceed this fixed byte length, even if they contain dynamic data objects.

EXAMPLE In the following example, a STATIC-FIELD is defined, which contains two elements with the structure defined by the object with ID=“ STRUCT_S_1”:

```
<STATIC-FIELD ID="SF-1">
     <SHORT-NAME>SF_1</SHORT-NAME>
     <BASIC-STRUCTURE-REF ID-REF="STRUCT_S-1"/>
     <FIXED-NUMBER-OF-ITEMS>2</FIXED-NUMBER-OF-ITEMS>
     <ITEM-BYTE-SIZE>4</ITEM-BYTE-SIZE>
</STATIC-FIELD>
```

See Figure 99 — Using STATIC-FIELD.

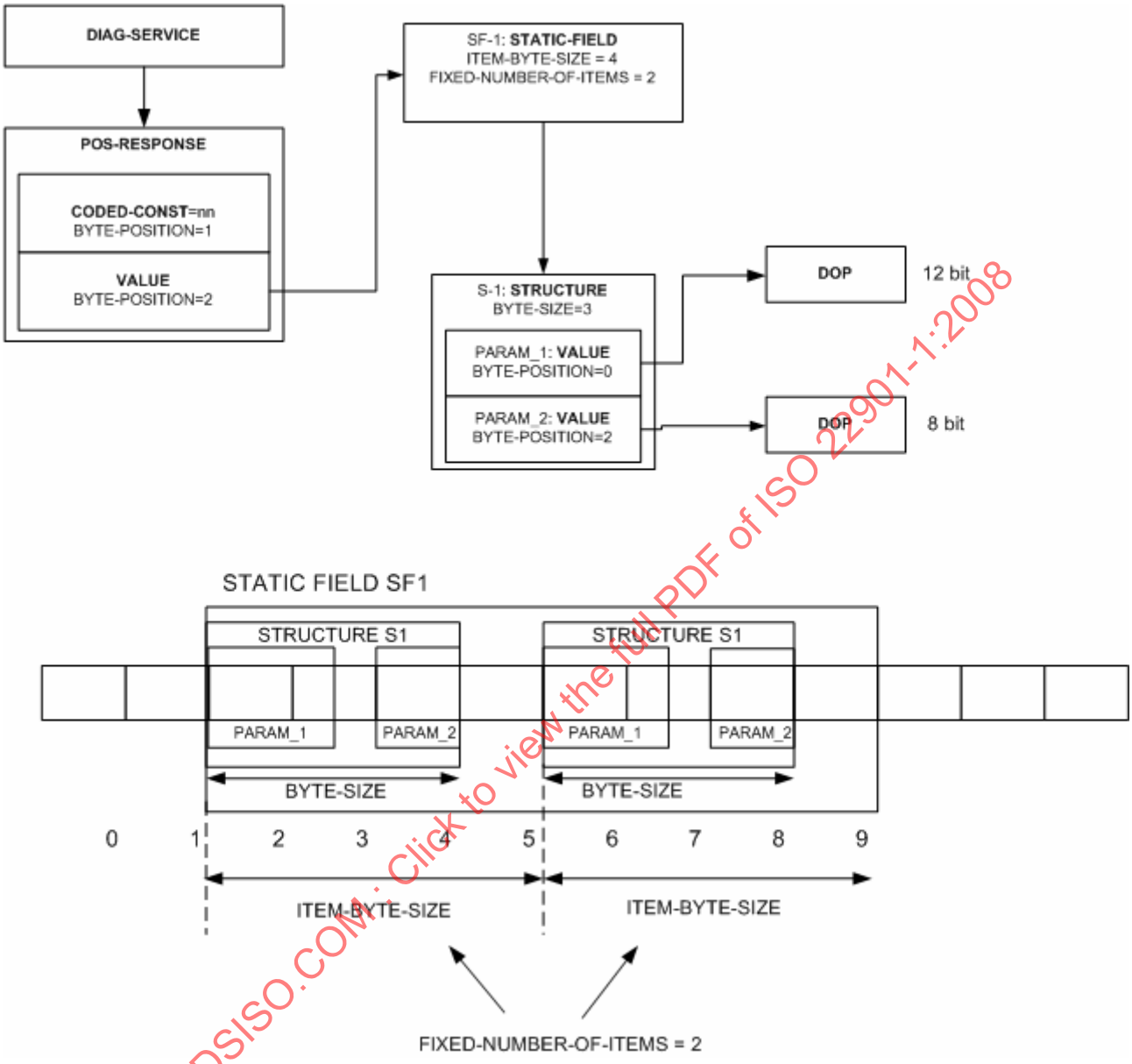


Figure 99 — Using STATIC-FIELD

7.3.6.10.4 Dynamic length field

Dynamic length fields (DYNAMIC-LENGTH-FIELD) are fields of items (BASIC-STRUCTURE or ENV-DATA-DESC) with variable number of repetitions, which can be determined only at runtime.

The determination of the number of repetitions is described by DETERMINE-NUMBER-OF-ITEMS. The DATA-OBJECT-PROP referenced to is used to calculate the repetition number, which is contained in its physical value of type A_UINT32. Its BYTE-POSITION is relative to that of the DYNAMIC-LENGTH-FIELD. The optional BIT-POSITION shall be between 0 and 7. The reference (by «odxlink» or «snref») to the BASIC-STRUCTURE or ENV-DATA-DESC defines the complex DOP to be repeatedly applied. For the first item, the byte position is given by OFFSET relatively to the byte position of the DYNAMIC-LENGTH-FIELD. Each further item starts at the byte edge following the item before (similar to PARAMs without explicitly specified BYTE-POSITION).

See Figure 100 — Using DYNAMIC-LENGTH-FIELD and the example below for further explanation.

EXAMPLE In the following example, a DYNAMIC-LENGTH-FIELD is defined, which contains a field of elements with the structure defined by the object with ID="STRUCT_S-1". The DOP with ID="DOP_1" extracts a number from the PDU at BYTE-POSITION=0 and calculates the element count.

```
<DYNAMIC-LENGTH-FIELD ID = "DLF1" IS-VISIBLE = "true">
    <SHORT-NAME>DLF1</SHORT-NAME>
    <BASIC-STRUCTURE-REF ID-REF = "STRUCT_S-1"></BASIC-STRUCTURE-REF>
    <OFFSET>2</OFFSET>
    <DETERMINE-NUMBER-OF-ITEMS>
        <BYTE-POSITION>0</BYTE-POSITION>
        <DATA-OBJECT-PROP-REF ID-REF = "DOP_1"/>
    </DETERMINE-NUMBER-OF-ITEMS>
</DYNAMIC-LENGTH-FIELD>
```

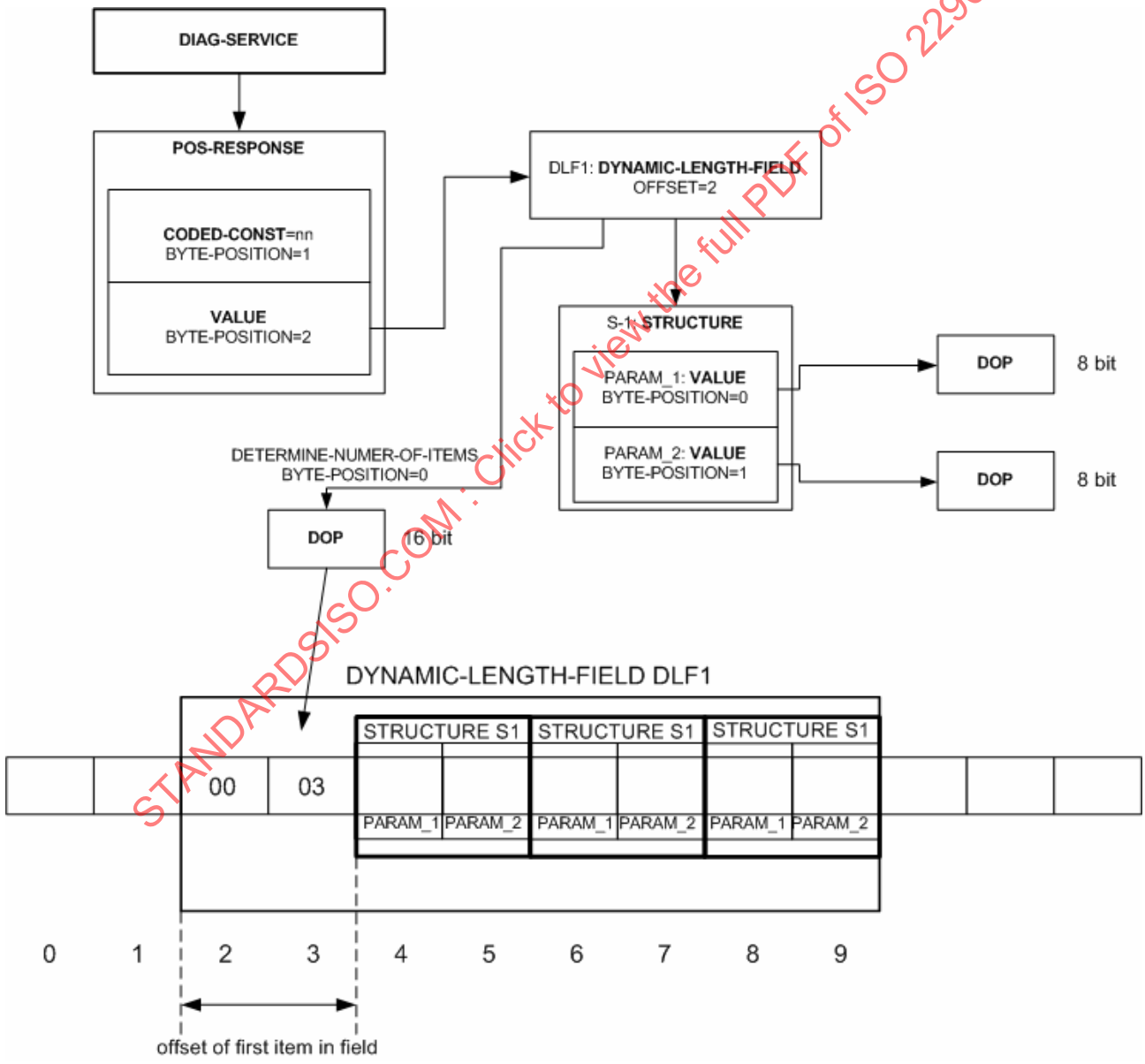


Figure 100 — Using DYNAMIC-LENGTH-FIELD

7.3.6.10.5 Dynamic endmarker field

Figure 101 — Using DYNAMIC-ENDMARKER-FIELD illustrates dynamic endmarker fields (DYNAMIC-ENDMARKER-FIELD) which are fields of items (BASIC-STRUCTUREs or ENV-DATA-DESCs) which are repeated until an end-marker is found (the TERMINATION-VALUE). It is also terminated by the end of the PDU. Before each iteration step the end of PDU condition is tested. If the end of PDU is reached, the processing of the iteration stops immediately. Otherwise, the referenced DATA-OBJECT-PROP is used to calculate a physical value of the parameter at the current position in the PDU. Before the first iteration, this is the byte position of the DYNAMIC-ENDMARKER-FIELD. If the resulting physical value matches the TERMINATION-VALUE (inside DATA-OBJECT-PROP-REF), the field ends without any additional item. Therefore, the value inside the TERMINATION-VALUE shall be given in the physical type of this referenced DATA-OBJECT-PROP.

NOTE If an additional item is desired, a parameter referencing the same STRUCTURE or ENV-DATA-DESC might follow the parameter referencing this DYNAMIC-ENDMARKER-FIELD.

The reference (by «odxlink» or «snref») to the BASIC-STRUCTURE or ENV-DATA-DESC defines the complex DOP to be repeatedly applied. For the first item, the byte position is given by the byte position of the DYNAMIC-ENDMARKER-FIELD. Each further item starts at the byte edge following the item before (similar to PARAMs without explicitly specified BYTE-POSITION).

The bytes defined by the ENDMARKER are not considered to be consumed. A parameter without BYTE-POSITION defined after the ENDMARKER-FIELD starts directly after the last structure of the field. If the interpretation of the ENDMARKER bytes is not desired they can be skipped with a RESERVED parameter.

In the following example a DYNAMIC-ENDMARKER-FIELD is defined, which contains a field of elements with the structure defined by the object with ID="STRUCT_S1". Before each iteration the DATA-OBJECT-PROP with ID="DOP_1" is calculated at the current position and converts it to a string, which is compared with TERMINATION-VALUE="TERMINATE". This string is returned by the DATA-OBJECT-PROP if the internal value is between 0xF1 and 0xFF. The iteration is stopped when a value between 0xF1 and 0xFF is reached.

EXAMPLE

```
<DYNAMIC-ENDMARKER-FIELD ID = "DEF1" IS-VISIBLE = "true">
    <SHORT-NAME>DEF1</SHORT-NAME>
    <BASIC-STRUCTURE-REF ID-REF = "STRUCT_S1"/>
    <DATA-OBJECT-PROP-REF ID-REF = "DOP_1">
        <TERMINATION-VALUE>TERMINATE</TERMINATION-VALUE>
    </DATA-OBJECT-PROP-REF>
</DYNAMIC-ENDMARKER-FIELD>
```

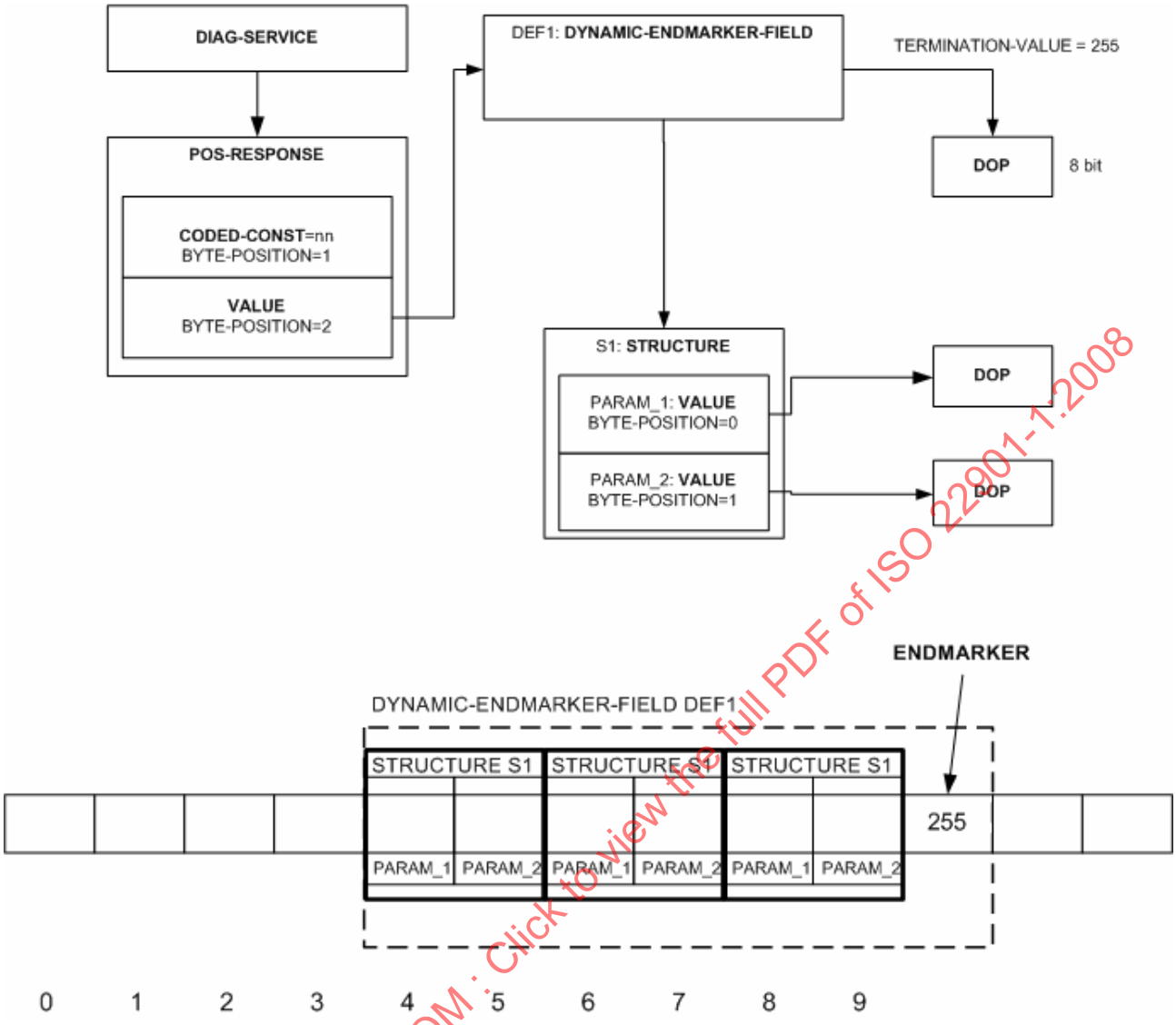


Figure 101 — Using DYNAMIC-ENDMARKER-FIELD

7.3.6.10.6 End of PDU - Field

Figure 102 — Using END-OF-PDU-FIELD illustrates end of PDU – Fields (END-OF-PDU-FIELD) which are similar to DYNAMIC-ENDMARKER-FIELDs with the difference, that the item is always repeated until the end of the PDU. END-OF-PDU-FIELD is the only class of type FIELD which is allowed to be used directly or indirectly inside a REQUEST.

In the case of building a request, the application shall set the number of items within the bounds defined by MIN-NUMBER-OF-ITEMS and/or MAX-NUMBER-OF-ITEMS before it is able to set the physical values of the VALUE parameters, which are contained inside the STRUCTURE referenced to.

IMPORTANT — A reference to an ENV-DATA or ENV-DATA-DESC is not allowed, if the END-OF-PDU-FIELD is used directly or indirectly inside a REQUEST.

In the case of analyzing a response, the values of MAX-NUMBER-OF-ITEMS and MIN-NUMBER-OF-ITEMS are ignored. Before each iteration step the end of PDU condition is tested. If the end of PDU is reached, the processing of the iteration stops immediately. The reference (by «odxlink» or «snref») to the BASIC-STRUCTURE or ENV-DATA-DESC defines the complex DOP to be repeatedly applied. For the first item, the

byte position is given by the byte position of the END-OF-PDU-FIELD. Each further item starts at the byte edge following the item before (similar to PARAMs without explicitly specified BYTE-POSITION).

EXAMPLE

```
<END-OF-PDU-FIELD ID = "EOP1" IS-VISIBLE = "false">
    <SHORT-NAME>EOP1</SHORT-NAME>
    <BASIC-STRUCTURE-REF ID-REF = "STRUCT_S1"></BASIC-STRUCTURE-REF>
    <MAX-NUMBER-OF-ITEMS>6</MAX-NUMBER-OF-ITEMS>
    <MIN-NUMBER-OF-ITEMS>1</MIN-NUMBER-OF-ITEMS>
</END-OF-PDU-FIELD>
```

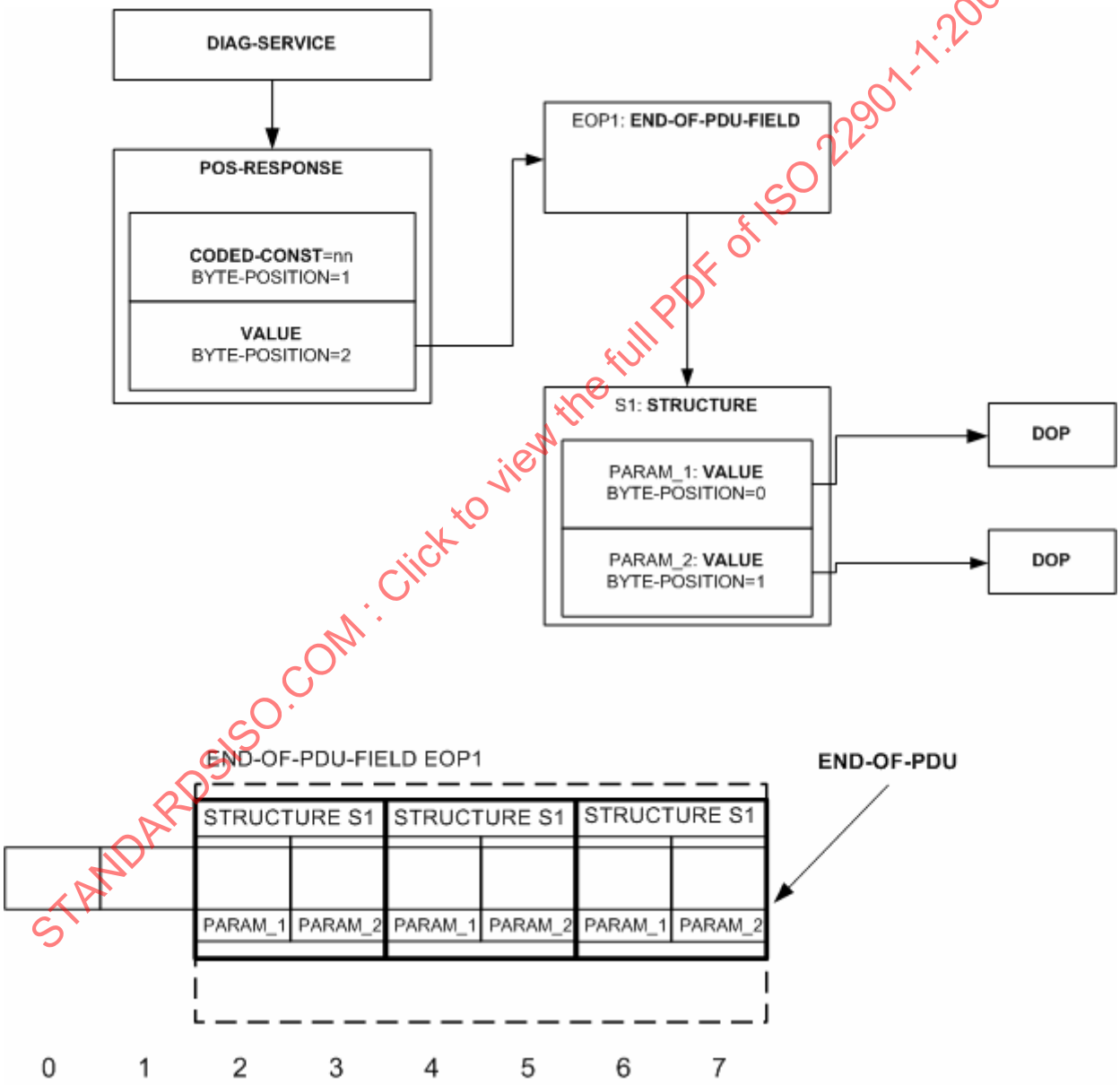


Figure 102 — Using END-OF-PDU-FIELD

7.3.6.10.7 Multiplexer

Figure 103 — UML representation of Multiplexer/MUX illustrates Multiplexers (MUX) which are used to interpret data stream depending on the value of a switch-key (similar to switch-case statements in programming languages like C or Java).

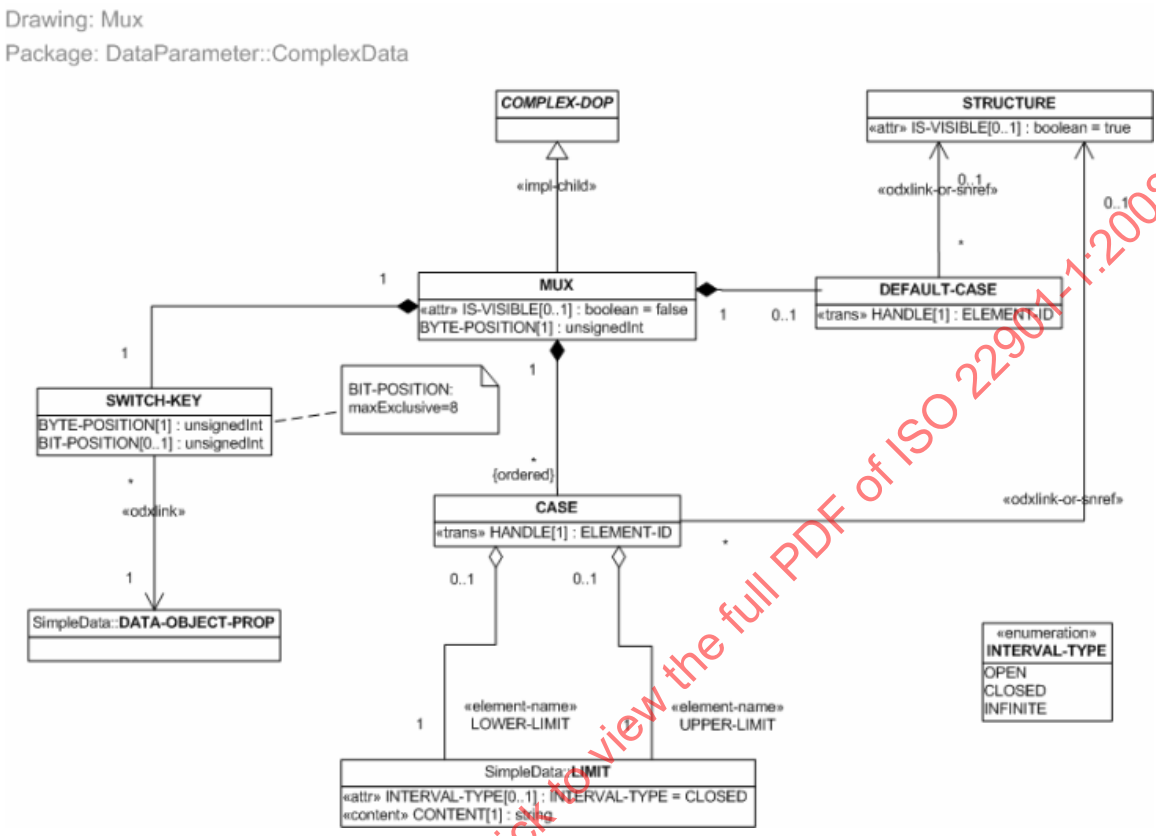


Figure 103 — UML representation of Multiplexer/MUX

The determination of the actual switch-key is described inside SWITCH-KEY. The DATA-OBJECT-PROP referenced to is used to calculate the switch-key, which is contained in its physical value. The DIAG-CODED-TYPE of that DATA-OBJECT-PROP shall be of type STANDARD-LENGTH-TYPE. Its BYTE-POSITION is relative to the that of the MUX. The optional BIT-POSITION shall be between 0 and 7. The DIAG-CODED-TYPE of that DATA-OBJECT-PROP shall be of type STANDARD-LENGTH-TYPE.

Any number of CASEs can be specified. If a MUX object has no CASES sub-object, it shall have a DEFAULT-CASE sub-object. Each of them defines a LOWER-LIMIT and an UPPER-LIMIT. The values shall match the physical type of the switch-key. The actual value of the switch-key is compared with the values inside LOWER-LIMIT and UPPER-LIMIT in the same way as for SCALE-CONSTR and COMPU-SCALE (see 7.3.6.2). The comparison rules are described in 7.3.6.5. If the physical type is A_UNICODE2STRING only identity is allowed, i.e. LOWER-LIMIT and UPPER-LIMIT shall be equal. If a matching CASE is found, the referenced STRUCTURE is analyzed at the BYTE-POSITION (child element of MUX) relatively to the byte position of the MUX. If a matching CASE cannot be found and the optional DEFAULT-CASE is specified, the STRUCTURE referenced by this DEFAULT-CASE is analyzed in the same way. Otherwise, the D-server shall report an error.

The reference to the STRUCTURE is optional inside CASE and DEFAULT-CASE. If it is not specified, the D-server will not signal an error condition if this case needs to be used. By this way it is possible to specify a CASE or DEFAULT-CASE with the only purpose to prevent an error message of the D-server, i.e. because the actual value of the switch-key is valid, but it is not connected with further data. The ranges defined by the CASEs shall not overlap. If no CASEs are specified, the optional DEFAULT-CASE shall be specified.

EXAMPLE The following example code defines a MUX, which interprets the data structure depending on the byte before the data structure referenced by this multiplexer.

```
<MUX ID="M1">
     <SHORT-NAME>M1</SHORT-NAME>
     <BYTE-POSITION>1</BYTE-POSITION>
     <SWITCH-KEY>
          <BYTE-POSITION>0</BYTE-POSITION>
          <DATA-OBJECT-PROP-REF ID-REF="DOP_1"/>
     </SWITCH-KEY>
     <CASES>
          <CASE>
               <SHORT-NAME>case_1</SHORT-NAME>
               <STRUCTURE-REF ID-REF="SCASE-24"/>
               <LOWER-LIMIT>24</LOWER-LIMIT>
               <UPPER-LIMIT>24</UPPER-LIMIT>
          </CASE>
          <CASE>
               <SHORT-NAME>case_2</SHORT-NAME>
               <STRUCTURE-REF ID-REF="SCASE-25"/>
               <LOWER-LIMIT>25</LOWER-LIMIT>
               <UPPER-LIMIT>25</UPPER-LIMIT>
          </CASE>
     </CASES>
</MUX>
```

Figure 104 — Using MUX clarifies the circumstances. The third byte of the shown PDU is the switch-key. In the first step the switch-key is extracted using BYTE-POSITION and the DOP referenced by SWITCH-KEY. In the second step the matching CASE is searched. This appropriate structure is then used to interpret the rest of the message. The physical type of the DOP used by the SWITCH-KEY shall match the data type in LOWER-LIMIT and UPPER-LIMIT.

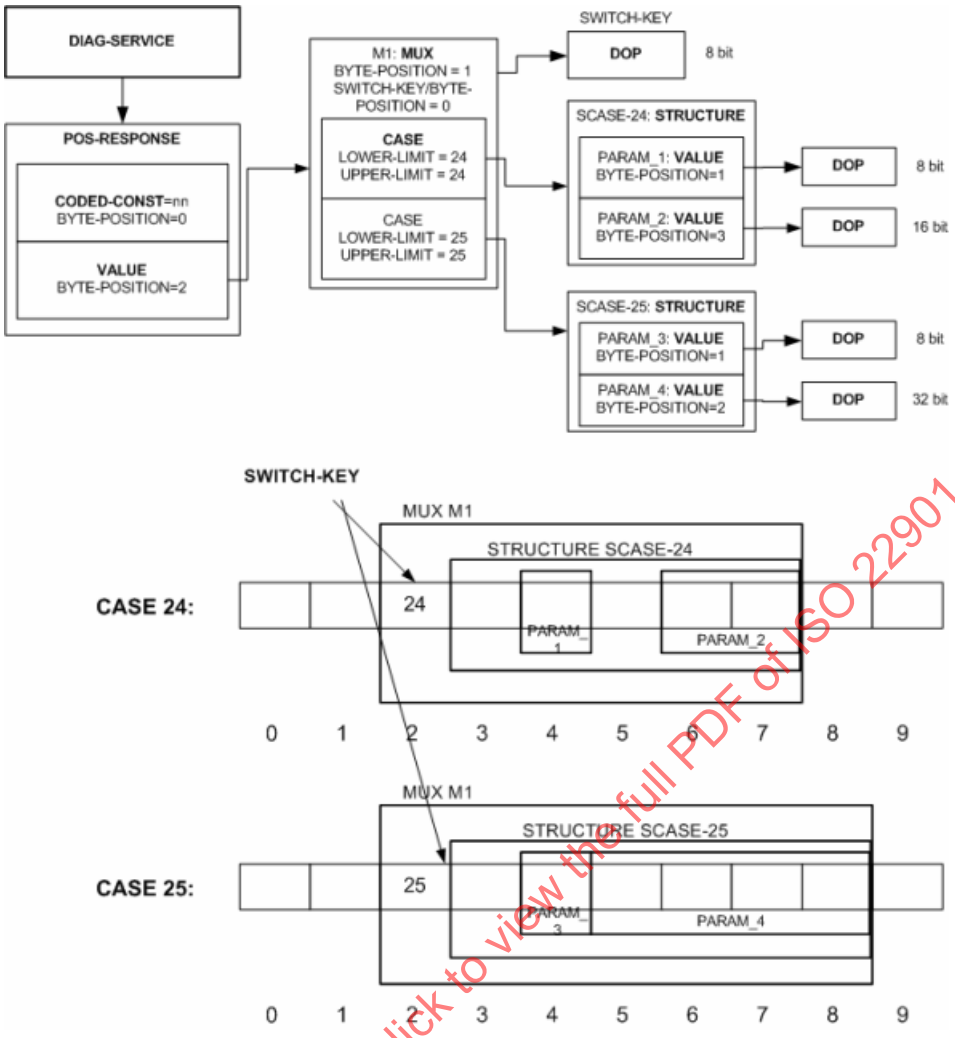


Figure 104 — Using MUX

7.3.6.11 TABLE – compound data object

The concept of data identifiers or parameter identifiers describes the association of a data structure definition to a numerical identifier. This data structure may contain a single parameter or a list of parameters. Typically there is a single parameter in case of analogue values like sensor voltages or a list of parameters in case of discrete values like switch statuses.

Most protocols use data identifier to describe data structures contained in response messages, however ISO 14229-1 defines for example the InputOutputControlByIdentifier (0x2F) or WriteDataByIdentifier (0x2E) service, which extends the use of DataIdentifier also to specify data structures contained in request messages. For this proceeding the element TABLE can be used. See Figure 105 — UML representation of TABLE for further modelling detail.

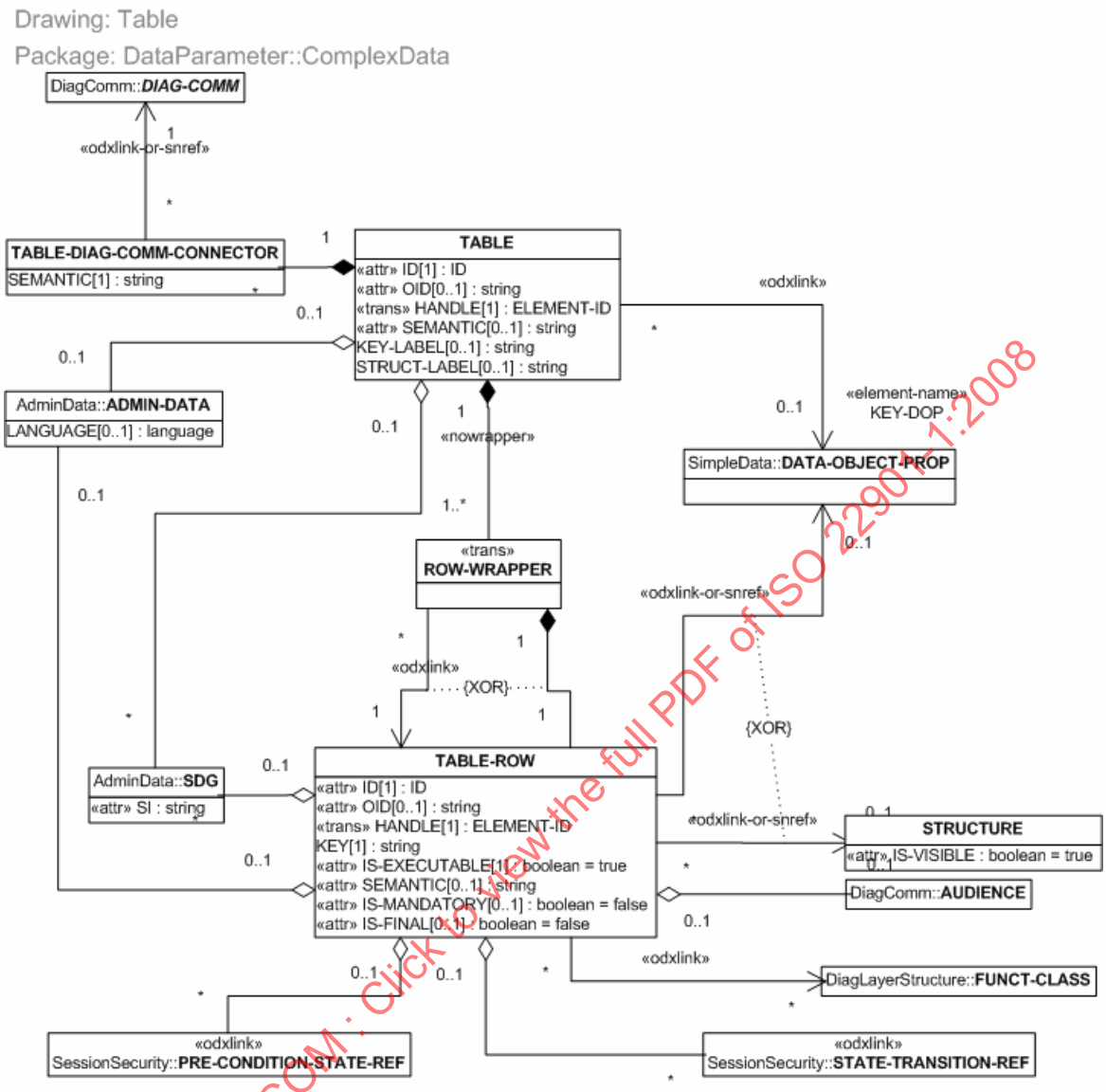


Figure 105 — UML representation of TABLE

The TABLE is similar to the COMPLEX-DOP MUX because both are multiplexers with a switch-key and cases which connect the various switch-key values with a data structures (e. g. STRUCTURE). In many cases, the use of a TABLE could be replaces by a MUX and vice versa. But the TABLE is more comfortable to specify in many cases and is connected with a lot of special semantics in the data model as well as in the processing of the D-server, e.g. the execution of diagnostic services by selection of the switch-key or dynamically defined services. Because the TABLE is not derived from DOP-BASE it can only be used by at the PARAM of types TABLE-KEY and TABLE-ENTRY unlike DOP-BASE-derived objects. The TABLE can be used for different kind of KEYs (data identifier). On account of this it has a SEMANTIC attribute. Values of SEMANTIC may be for example DATA-ID (DataIdentifier), PACKAGE-ID (PackageIdentifier), PARAMETER-ID (ParameterIdentifier), LID (LocalIdentifier), CID (CommonIdentifier) and ADDRESS (MemoryAddress in the ECU).

According to its name TABLE is a composed of a KEY-column and a STRUCT-column. The columns are labelled by KEY-LABEL and STRUCT-LABEL for the documentation use-case. A TABLE-ROW associates a data identifier called KEY with an optional parameter definition via reference (by «odxlink» or «snref») to a STRUCTURE or DATA-OBJECT-PROP. A TABLE-ROW already defined in another TABLE can be reused by reference. Because this is in fact an import (see 7.3.2.5) this referenced TABLE-ROW is considered as locally defined.

The DATA-OBJECT-PROP referenced by KEY-DOP-REF defines the conversion and coding of the KEY in any request parameter and the extraction and conversion of the KEY out of any response parameter. The KEY value in the XML-instance is always a physical value of the physical type of the DATA-OBJECT-PROP referenced to.

Three PARAM types are introduced for usage with TABLE, as described below.

a) TABLE-KEY: A parameter of type TABLE-KEY is the part of the PDU where the KEY (data identifier) shall be read in the case of response or written in the case of request. The parameter can refer the TABLE for dynamic parameter definition (similar to parameter type VALUE without PHYSICAL-DEFAULT-VALUE) or it refers a TABLE-ROW for static definition similar to parameter type PHYS-CONST. The TABLE-KEY-parameter in the response shows the position where the KEY should be extracted using the DATA-OBJECT-PROP referenced by TABLE. The physical result shall match with exactly one KEY-value in the TABLE. The TABLE-ROW containing the matching KEY refers the STRUCTURE or DATA-OBJECT-PROP describing the data parameter connected with this KEY. To include this data inside the message, use a TABLE-STRUCT parameter referring to this TABLE-KEY parameter.

b) TABLE-STRUCT: This parameter type defines the position of the STRUCTURE or DATA-OBJECT-PROP referenced by the TABLE-ROW which was selected by the corresponding TABLE-KEY parameter. The TABLE-KEY is referenced inside TABLE-STRUCT by «odxlink-or-snref».

c) TABLE-ENTRY: This parameter type can be used for including 'foreign' table row definitions inside a TABLE-STRUCT. That means that when using a TABLE-ENTRY parameter within a structure defined by a table row, this parameter will be interpreted based on the definitions referenced by the TABLE-ENTRY object. The TARGET attribute of a TABLE-ENTRY element indicates whether the KEY-DOP (attribute value KEY, in this case the TABLE shall define a KEY-DOP element) or the STRUCTURE (attribute value STRUCT) referenced by the table row the TABLE-ENTRY points to should be used for interpretation. This parameter type can only be used in a structure referenced by a TABLE-ROW.

A TABLE-KEY always corresponds to a TABLE. The corresponding TABLE can be determined in different ways.

The TABLE is either:

— the one the TABLE-KEY refers to directly via the <<odxlink>> from the <<trans>>parent element ODXLINK-TO-TABLE;

— the one the TABLE-KEY refers to directly via a <<snref>> from the <<trans>>parent element SNREF-TO-TABLE;

— the one which actually defines (not imported via «odxlink») the TABLE-ROW, that the TABLE-KEY refers to via «odxlink»; other TABLEs are ignored, even if they import the same TABLE-ROW element.

The TABLE that corresponds with a TABLE-KEY shall define a KEY-DOP.

A TABLE-ENTRY corresponds to the TABLE which actually defines (not imported via «odxlink») TABLE-ROW the TABLE-ENTRY itself references via <<odxlink>>. Other TABLEs are ignored, even if they import the same TABLE-ROW element. The TABLE that corresponds with a TABLE-ENTRY shall define a KEY-DOP, if the ROW-FRAGMENT of the TABLE-ENTRY is KEY.

See Figure 106 — UML representation of parameter for further modelling detail.

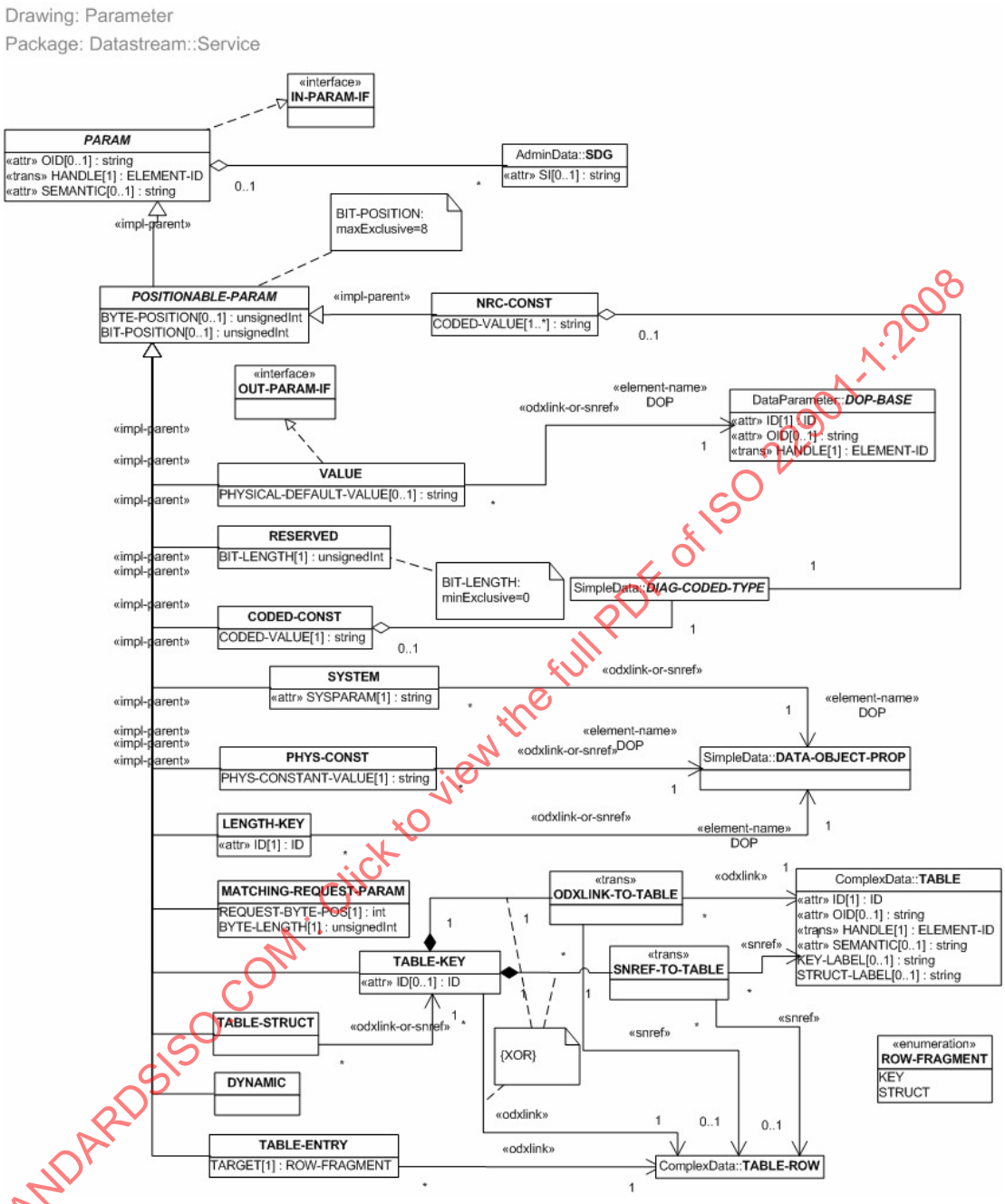


Figure 106 — UML representation of parameter

EXAMPLE In the following figures, some examples for writing and reading data identifier are shown. Two of them are implemented in static and dynamic manner. In case of dynamic implementation of a request parameter, a TABLE-KEY refers a TABLE as a whole. Identical to PARAM-type VALUE without PHYSICAL-DEFAULT-VALUE, the tester needs to choose a TABLE-ROW during runtime.

At the static implementation, the TABLE-KEY refers a TABLE-ROW directly and defined a permanent parameter value in this way identical to PARAM-type PHYS-CONST.

It is possible to restrict the use of a TABLE-ROW. In addition to the diagnostic service connected with the TABLE, each TABLE-ROW can define its own attribute IS-EXECUTABLE with the default value “true”. If its value is set “false” the D-server shall reject the execution via API of the diagnostic service which refers to this

TABLE-ROW inside its request, regardless whether by a static or dynamic implementation of TABLE-KEY. Such a service is only executable inside a job. If the service is marked as not executable the attribute value at the TABLE-ROW is irrelevant. That means a value “true” at TABLE-ROW cannot override the value “false” at DIAG-COMM.

In general, the TABLE-ROW refers to a STRUCTURE with at least one parameter PARAM. But in some cases this normal behaviour is not a suitable description for a set of diagnostic services which should be combined inside one TABLE. One example is the diagnostic class RoutineControl with the subfunctions startRoutine, stopRoutine, and requestRoutineResults. While using a TABLE with these subfunctions as TABLE-ROW the problem arises that some parameters, e.g. routineControlOptionRecord, are optional and not suitable for each subfunction. In such a case it is possible to specify a TABLE-ROW without a reference to a STRUCTURE. In another use case, the existence of parameters depends on the actual variant of the ECU. For example, no parameter is available inside the base variant, but inside specific variants. This can specified by a SHORT-NAME reference to a STRUCTURE without any parameters inside the BASE-VARIANT, which is overridden inside the ECU-VARIANT by a STRUCTURE with the actual parameters available in this variant. See Figure 107 — ISO 15031-5 Service 0x01 PID example, Figure 108 — ISO 14229-1 Read data by identifier example (dynamic) and Figure 109 — ISO 14229-1 Read data by identifier example (static) for further modelling detail.

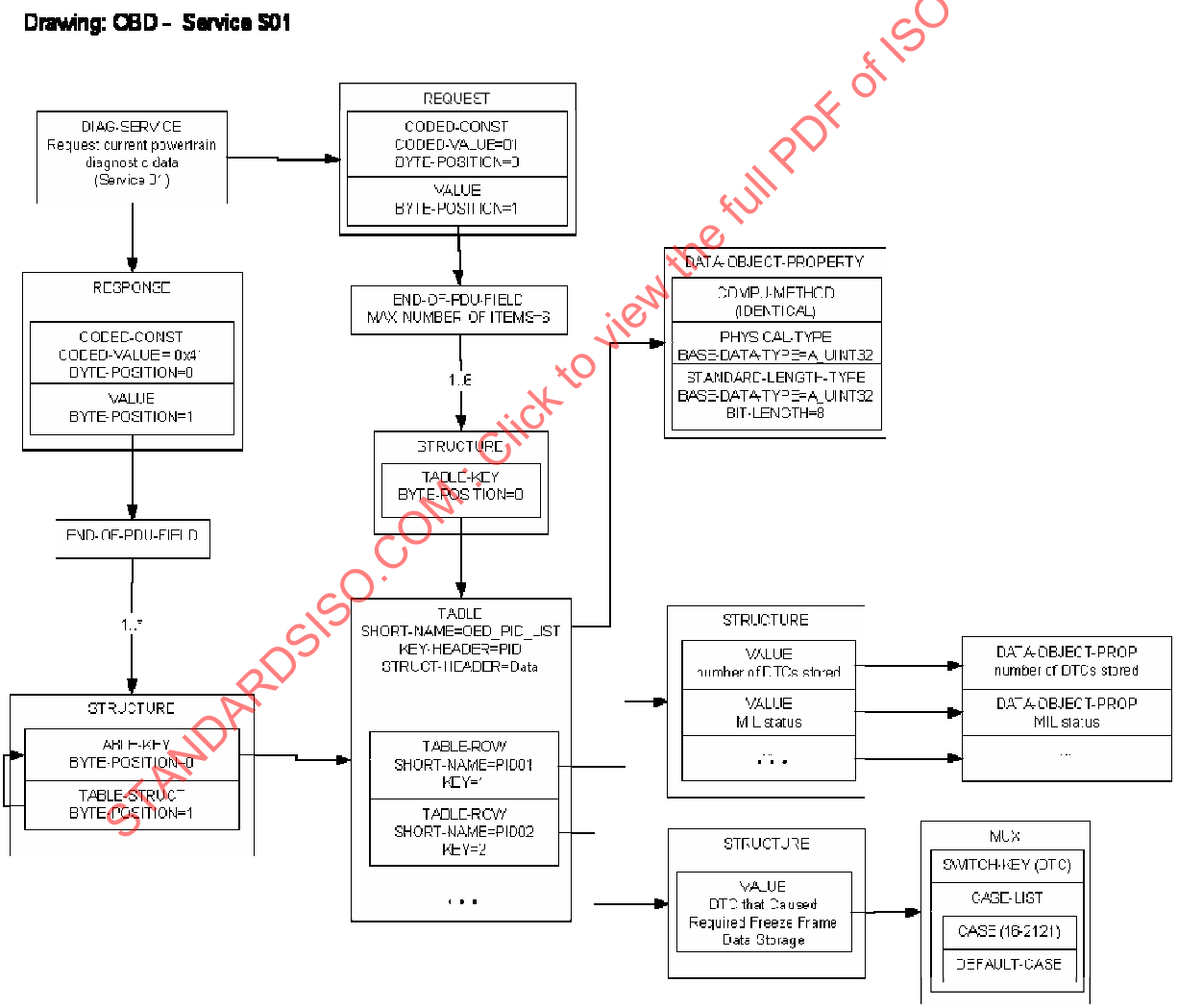


Figure 107 — ISO 15031-5 Service 0x01 PID example

Drawing: ISO14229-1 RDBI (dyn)

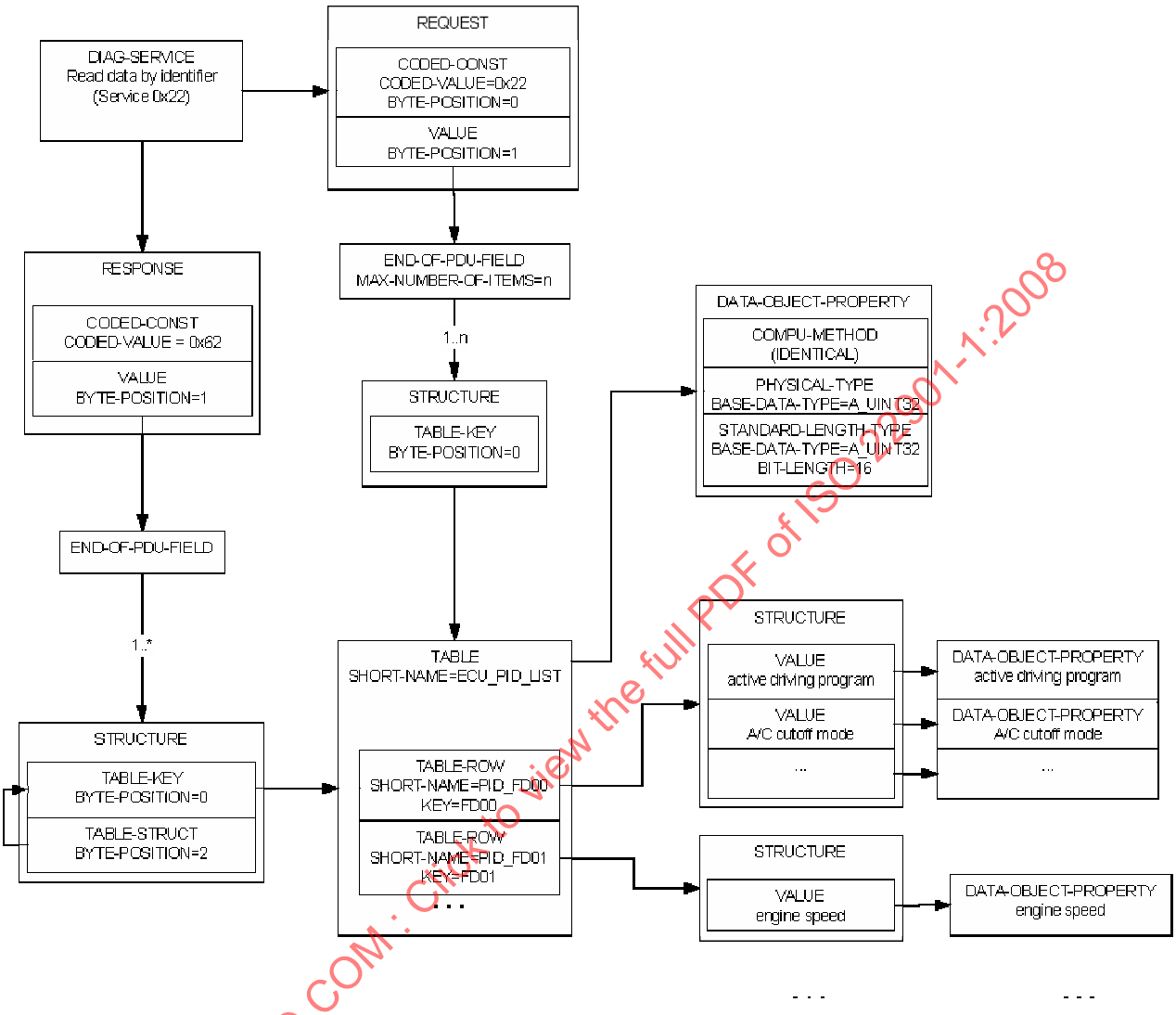


Figure 108 — ISO 14229-1 Read data by identifier example (dynamic)

Drawing: ISO14229-1 RDBI (static)

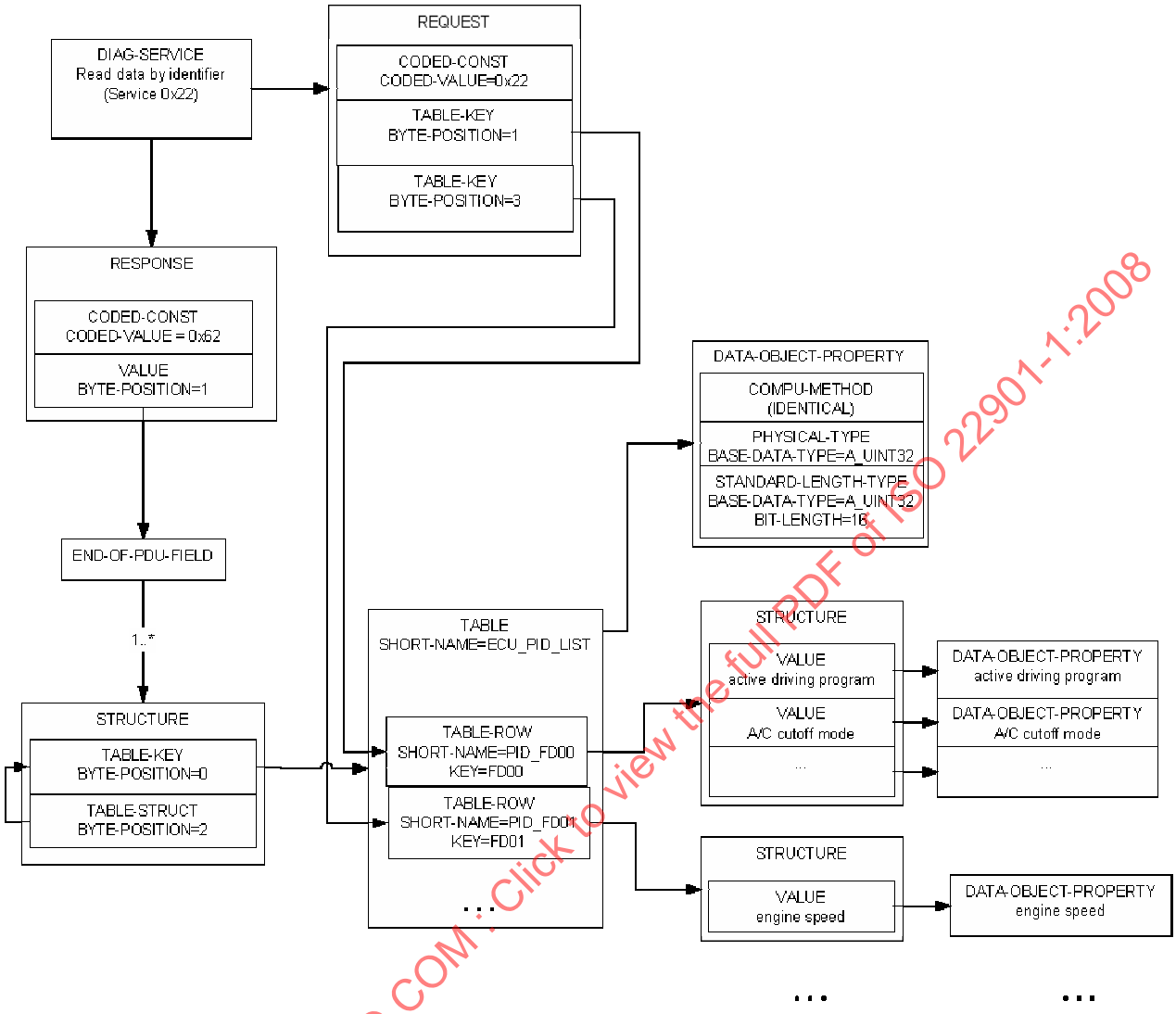


Figure 109 — ISO 14229-1 Read data by identifier example (static)

In the dynamic case, the user shall select the case via TABLE-KEY, see Figure 110 — ISO 14229-1 Write data by identifier (dynamic).

IMPORTANT — The PARAMs of type VALUE defined shall be offered to be set by the application.

According to the selection of the TABLE-KEY, the corresponding TABLE-STRUCT shall be used by the application to determine the PARAMs to be set by the user.

In the static case that selection is not allowed because TABLE-KEYs are statically defined by the ODX data. Nevertheless, if the data structure included by TABLE-STRUCT contains parameters of type VALUE, the user is able or obliged (if no PHYSICAL-DEFAULT-VALUE is specified) to set their values, see Figure 111 — ISO 14229-1 Write data by identifier (static).

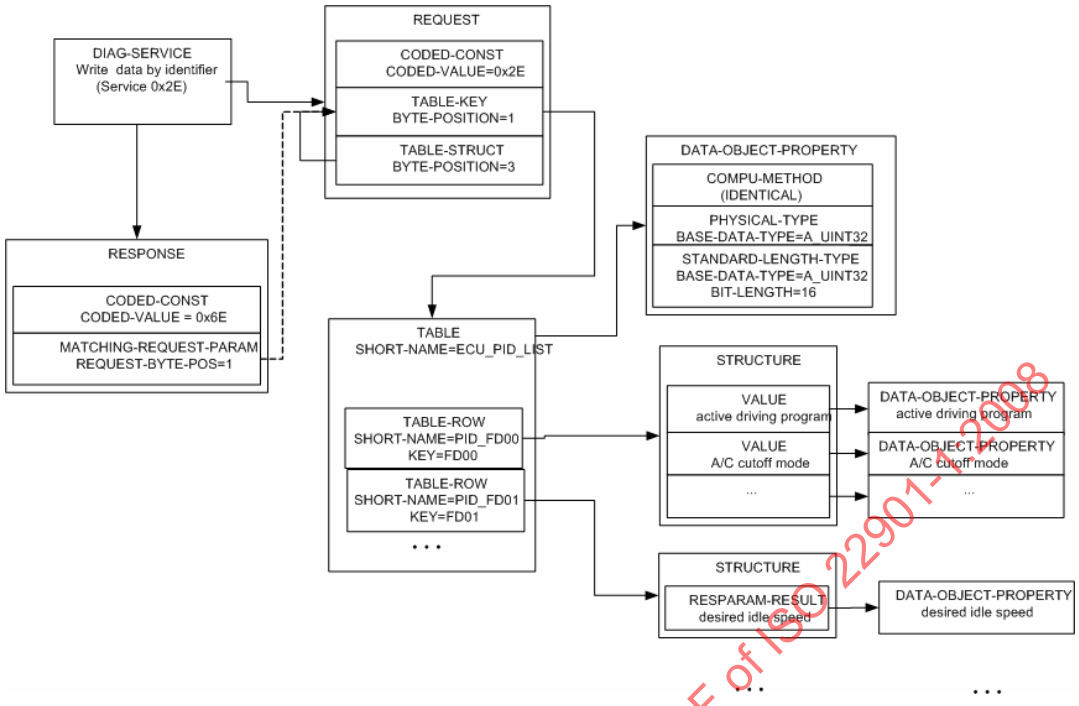


Figure 110 — ISO 14229-1 Write data by identifier (dynamic)

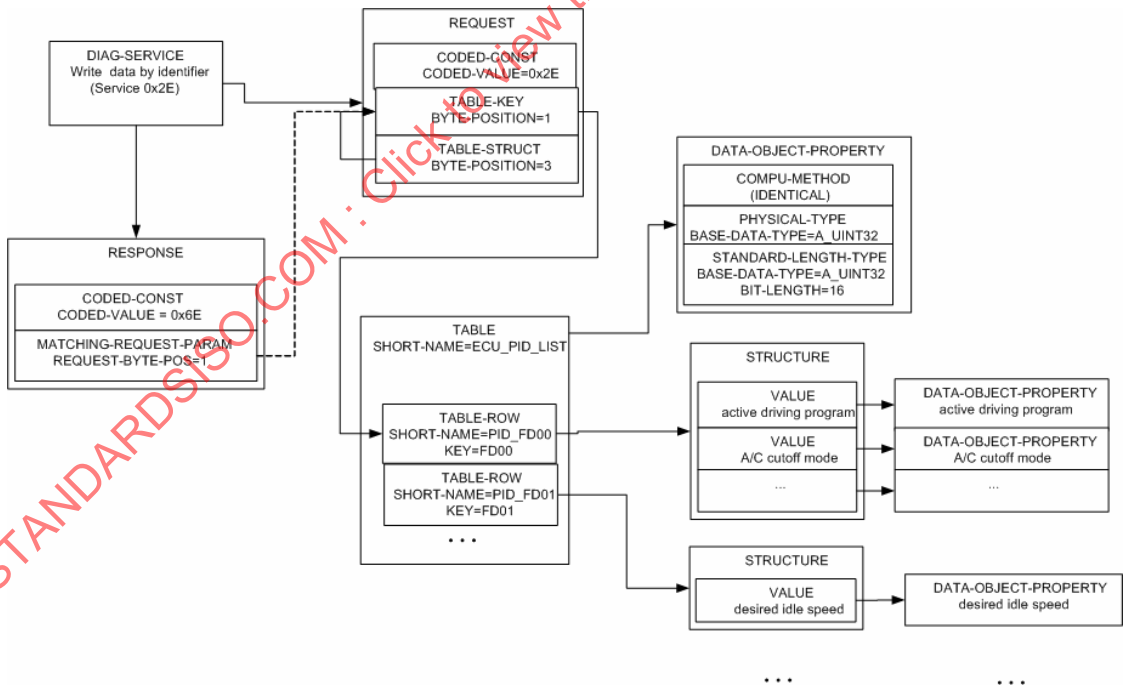


Figure 111 — ISO 14229-1 Write data by identifier (static)

Figure 112 — Use of TABLE-ENTRY (GMlan DPID Example) shows an example for the usage of TABLE-ENTRY.

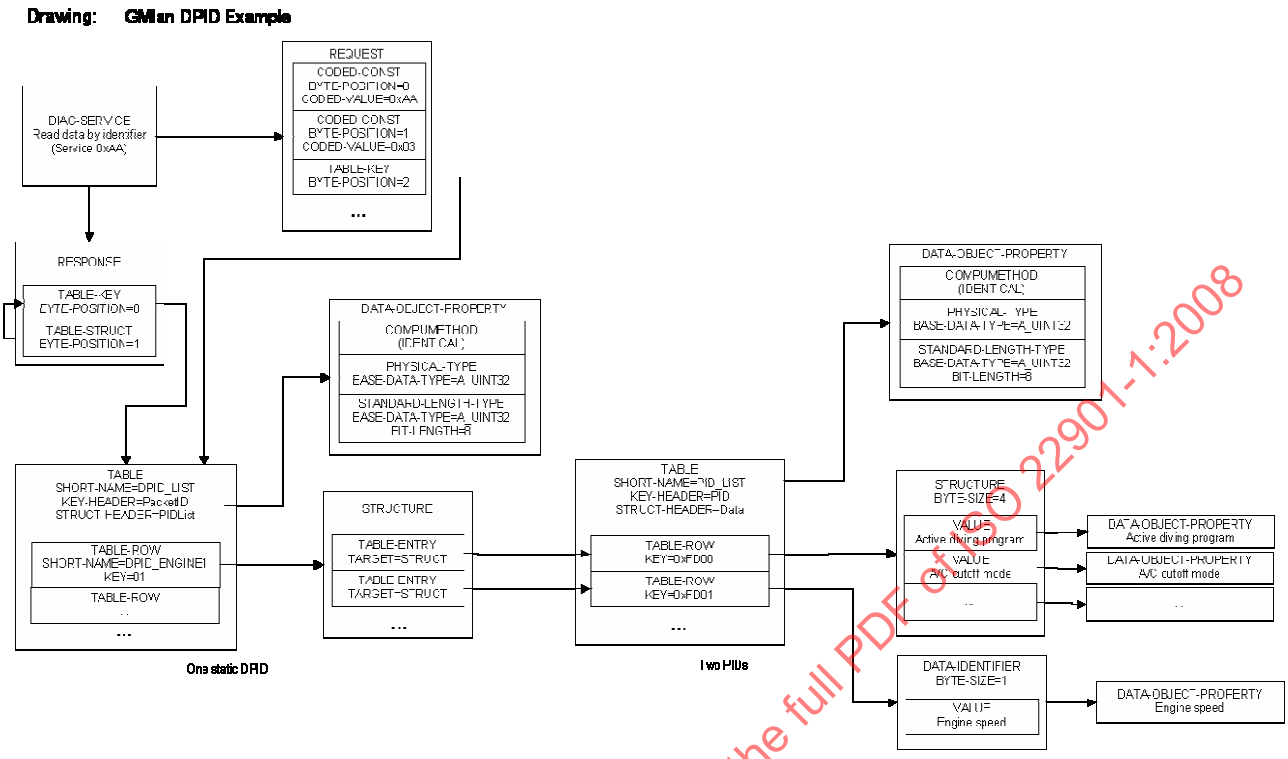


Figure 112 — Use of TABLE-ENTRY (GMlan DPID Example)

Each TABLE-ROW can refer to STATE-TRANSITIONs, PRE-CONDITIONs and FUNCT-CLASSes. The use of these references complies with the use at DIAG-SERVICEs. If a TABLE-ROW defines state transitions and/or pre-conditions, this definition is added to the possible definition at the appropriate DIAG-SERVICE. That means that the allowed states for the execution of the DIAG-SERVICE with a concrete TABLE-ROW are the sum of all pre-conditions defined at the DIAG-SERVICE and at the TABLE-ROW. The same applies to the state transitions. All STATE-TRANSITIONs at the DIAG-SERVICE and at the TABLE-ROW are put together. If the DIAG-SERVICE is executed with the TABLE-ROW all STATE-TRANSITIONs are evaluated. The STATE-TRANSITION with the SOURCE state that is equal to the current state is applied. If the DIAG-SERVICE is executed successfully, the state of the ECU changes from the specified SOURCE to the specified TARGET state.

Similar to DIAG-COMMs also TABLE-ROWs may specify the attributes IS-FINAL and IS-MANDATORY. If IS-FINAL="true", the TABLE-ROW shall not be changed in a derived layer. If a TABLE contains such TABLE-ROWs, the TABLE with the same SHORT-NAME in the lower layer shall not define TABLE-ROWs with the same KEYs. Instead of that, it shall import the original TABLE-ROWs if necessary.

In case if IS-MANDATORY=true, the TABLE-ROW shall be available in the derived layer. If a TABLE contains such TABLE-ROWs, a TABLE with the same SHORT-NAME in the lower layer shall contain TABLE-ROWs (or TABLE-ROW-REFs) with the same KEYs. Furthermore, such a TABLE cannot be completely eliminated from the inheritance by means of NOT-INHERITED-TABLE, but it can be overridden or inherited from another layer. The value of IS-MANDATORY shall not be changed in an overridden TABLE-ROW from "true" to "false", i.e. a mandatory TABLE-ROW cannot be made optional.

Both attributes IS-FINAL and IS-MANDATORY are applied to TABLE-ROWs only not to the referenced STRUCTURE or DATA-OBJECT-PROP which can be overridden. The D-server ignores the attributes IS-MANDATORY and IS-FINAL.

Figure 113 — Use of IS-FINAL and IS-MANDATORY shows the use of IS-FINALE and IS-MANDATORY attributes:

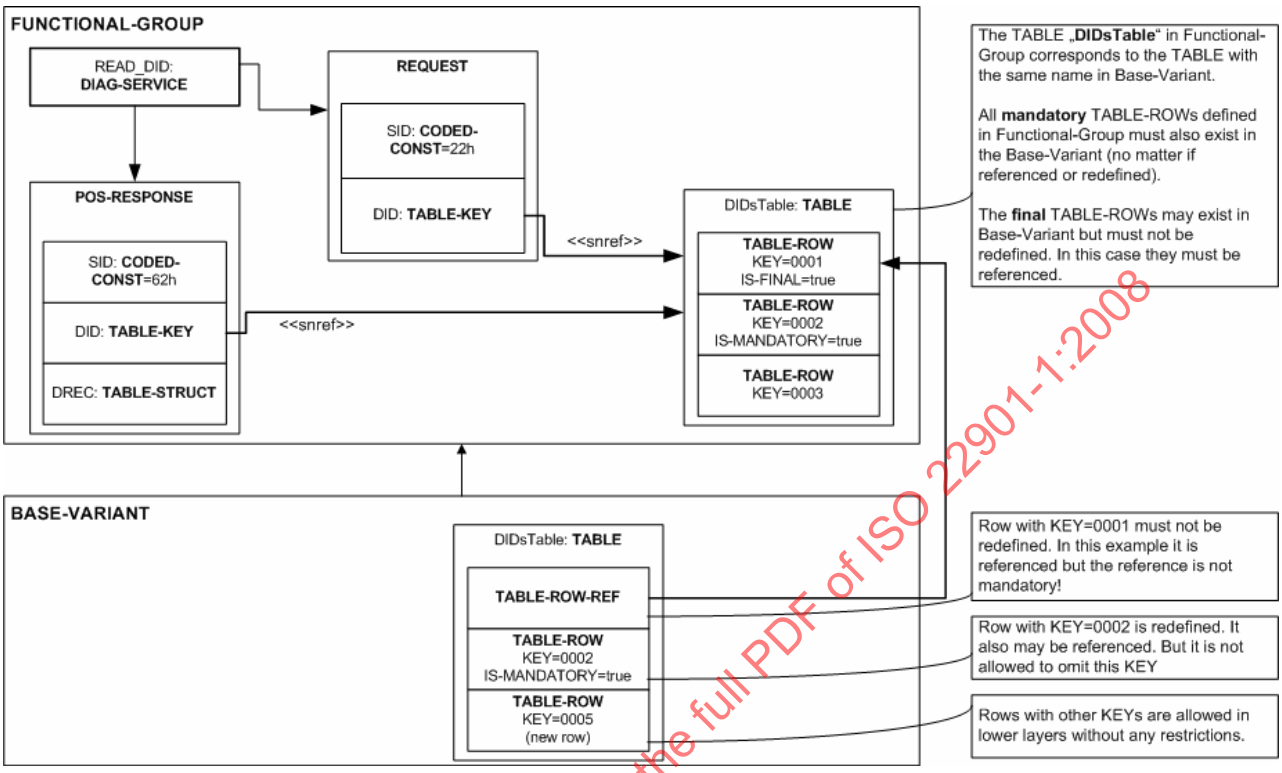


Figure 113 — Use of IS-FINAL and IS-MANDATORY

7.3.6.12 Cascading tables

Figure 114 — Use of cascading tables shows the use of cascading tables. There is the main table with DataIdentifiers (DIDs_TAB), which contains all DIDs that can be written with the UDS service 0x2E. In the simple case each DID is defined by a number (KEY) and the appropriate data record (STRUCTURE) with one or more simple parameters. The DID 0x1000 is such a DID. It refers to a STRUCTURE with the only parameter “Timestamp”. There is also a complex DID in this figure: 0x1001. The first byte (ControlParameter) of its data record decides about the remaining structure. If its value is equal to 0x00, only the threshold value is sent to the ECU (STRUCTURE DID_1001_00). In case of 0x01 one more parameter is needed: its name is “Time”. In this case the parameters of the STRUCTURE “DID_1001_01” are to be set by the user.

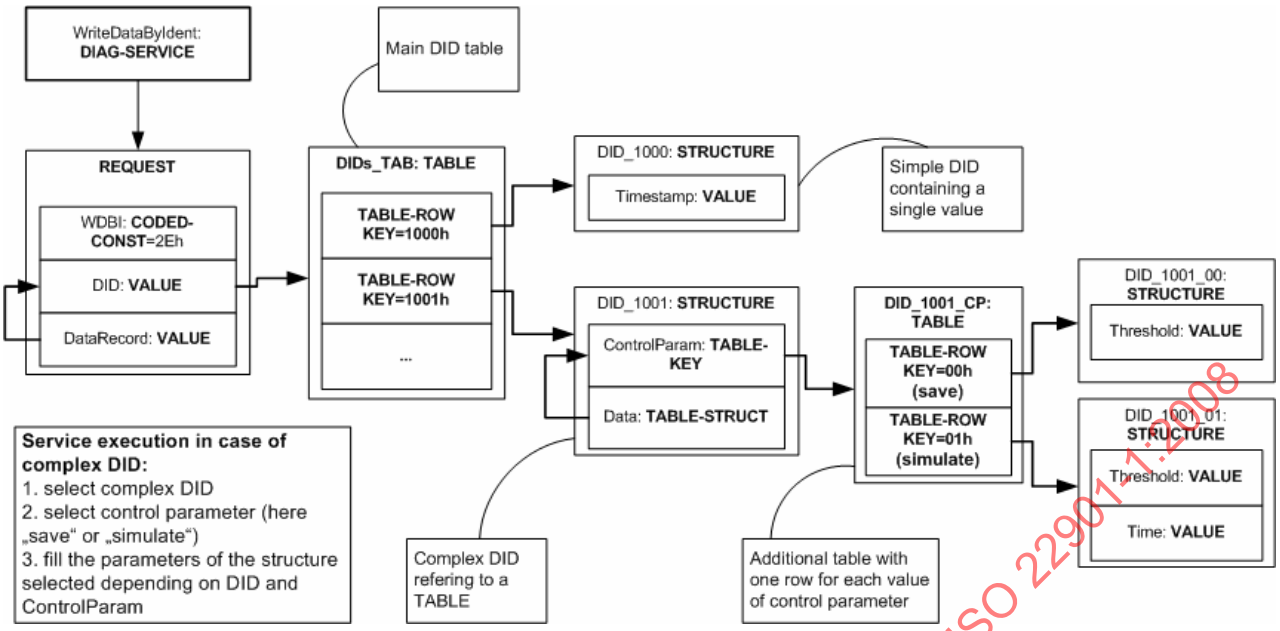


Figure 114 — Use of cascading tables

7.3.7 Diagnostic variable

7.3.7.1 General

In the measurement and calibration (MC) use case, the tester has direct access to the ECU memory to read and write data. The ECU provides internal variables to simplify this memory access. An ECU-internal variable maps a logical variable identifier (also called “label”) to the memory address of the variable value. This mapping is part of the ECU software; thus, an ECU-internal variable is called “SW-variable”.

In the diagnostic use case, another communication paradigm is preferred: diagnostic data is typically transferred via diagnostic services or diagnostic jobs that read/write data from/to the ECU. The ECU memory is not accessed directly.

A D-server should be capable to utilize SW-variables from the MC use case also in the diagnostic use case. For this reason the notion of diagnostic variables (DIAG-VARIABLEs) is introduced in ODX. A DIAG-VARIABLE emulates a SW-variable by calling appropriate diagnostic services for operations like reading or writing a value from/to the ECU.

An ODX conformant diagnostic system makes it possible to communicate with an ECU via both DIAG-VARIABLEs and diagnostic services. The user can chose the communication paradigm that he is more familiar with.

Figure 115 — UML representation of diagnostic variable layer illustrates the modelling.

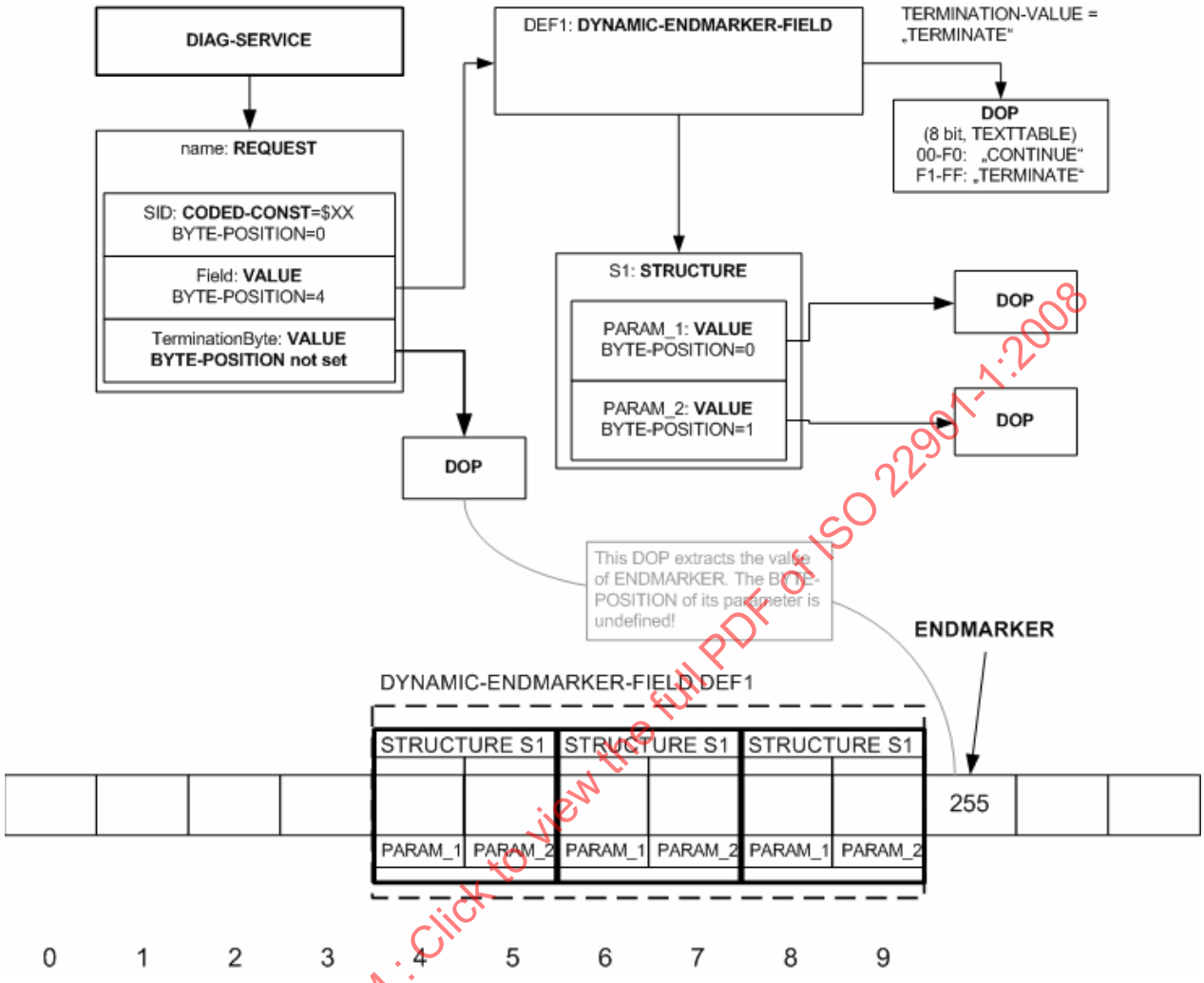


Figure 115 — UML representation of diagnostic variable layer

7.3.7.2 Use cases

The following use cases apply:

— functional testing of ECUs during development: after setting a value in ECU memory the corresponding DIAG-VARIABLE is read by a diagnostic system while the corresponding SW-variable is read by an MC system; the received values are compared to detect functional errors of the ECU;

— list of all SW-variables which can be accessed via diagnostic services;

— clustering of related services: DIAG-VARIABLEs contain information which services are related (RELATION-TYPE Element) to each other and deal with the same variable, e.g. “read”, “write”, and “reset to default”.

See Figure 116 — UML representation of diagnostic variable.

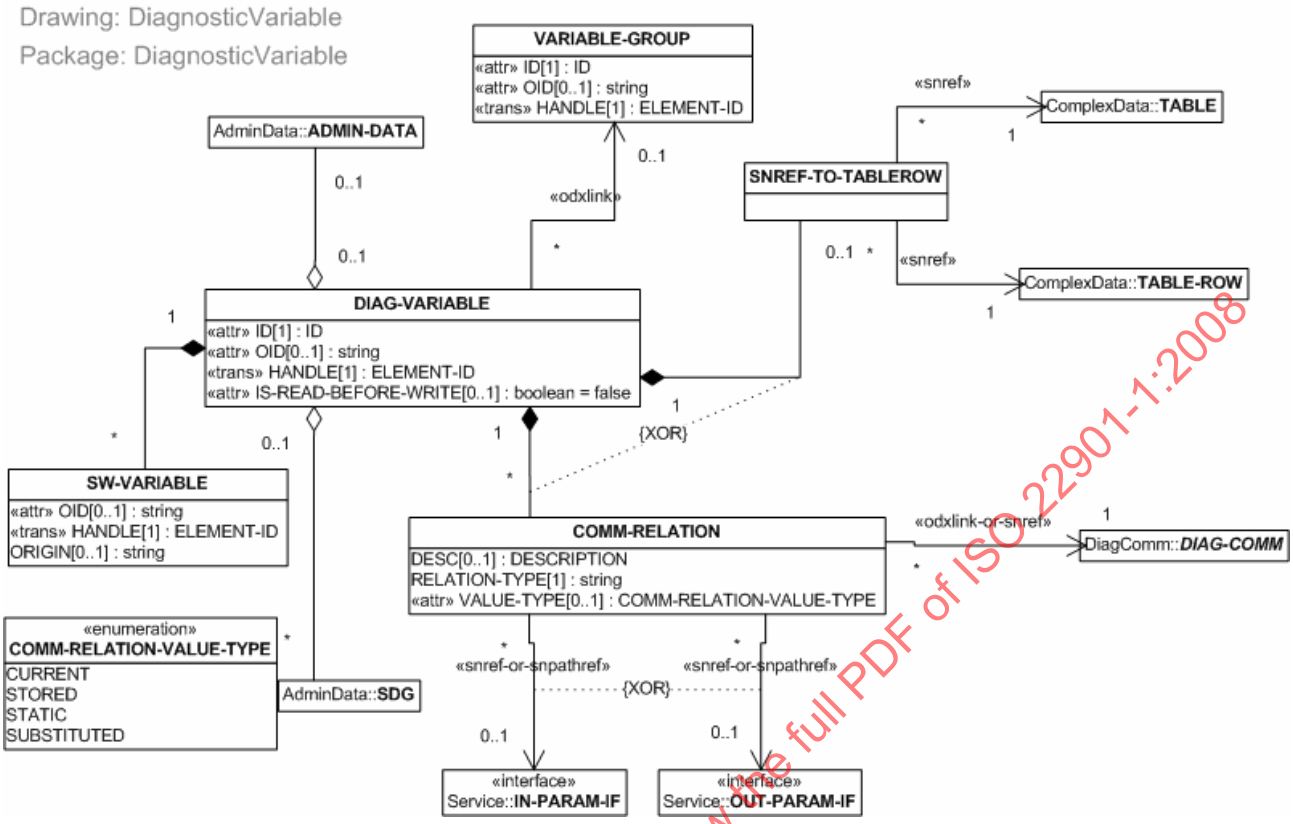


Figure 116 — UML representation of diagnostic variable

7.3.7.3 Structure of DIAG-VARIABLE

A DIAG-VARIABLE typically depends on a number of DIAG-COMM objects (diagnostic services or jobs).

This dependency can be expressed either by placing the appropriate DIAG-COMM objects into the COMM-RELATION-TYPE element with the adequate value at RELATION-TYPE (1) or by referencing a TABLE and TABLE-ROW object using SHORT-NAME references (2).

a) For example, to read the value of a variable a service is needed that requests the corresponding data from the ECU. A COMM-RELATION object describes the relation between the DIAG-VARIABLE and a DIAG-COMM object; each COMM-RELATION refers to one DIAG-COMM object. Depending on the type of relation there may be a reference to a parameter in the PDU, which contains the variable. The RELATION-TYPE describes the type of the operation that the referenced DIAG-COMM performs on the DIAG-VARIABLE. Valid values are READ, WRITE, RETURNCONTROL etc. A DIAG-VARIABLE shall not have multiple COMM-RELATIONs with the same RELATION-TYPE string. Modelling the dependency this way provides no possibility to set the corresponding PARAM object (DIAG-SERVICE) and INPUT-PARAM object (SINGLE-ECU-JOB) in order to define e.g. which data identifier to read. This approach leads to inflated ODX data, because for each DIAG-VARIABLE at least one DIAG-COMM object shall be defined with the appropriate value set for the parameter, e.g. by using the PHYSICAL-DEFAULT-VALUE element inside a VALUE parameter.

b) A second possibility was introduced to describe the dependency between a DIAG-VARIABLE and a DIAG-COMM. Therefore, the new element SNREF-TO-TABLEROW has been added to the ODX model. This new element consists of two child elements, TABLE-SNREF and TABLE-ROW-SNREF, which reference a TABLE and a TABLE-ROW by SHORT-NAME respectively. This is necessary to target a specific TABLE-ROW unambiguously. In addition, the TABLE class has a child element TABLE-DIAG-COMM-CONNECTORS to connect one or more DIAG-COMM objects to a TABLE. The new class

references a DIAG-COMM. The corresponding SEMANTIC can be used to define the kind of DIAG-COMM referenced, i.e. a READ and a WRITE service for data identifiers. In DIAG-VARIABLE objects it is only allowed to use either the COMM-RELATION mechanism or the SNREF-TO-TABLEROW mechanism. It is not allowed to use both or none of them.

Operations like reading or writing the value of a DIAG-VARIABLE imply a data transfer from the ECU to the D-server or vice versa. In this case, when using the COMM-RELATION element the service parameter that is used for the data transfer is specified by one of the attributes IN-PARAM-IF-SNREF and OUT-PARAM-IF-SNREF/IN-PARAM-IF-SNREF denotes a PARAM of the referenced DIAG-COMM's REQUEST that is used for writing data to the ECU. To read data from the ECU the OUT-PARAM-IF-SNREF is set to the name of the corresponding PARAM of the RESPONSE of the DIAG-COMM. If no data needs to be transferred between the D-server and the ECU, no service parameter needs to be selected. The D-server performs such an operation for example when it returns the control of the DIAG-VARIABLE to the ECU.

When using the SNREF-TO-TABLEROW mechanism, no PARAM-IF-SNREF needs to be specified, because by targeting a certain TABLE-ROW element inside a TABLE the DIAG-COMM to use and the PARAM to set are clearly specified.

A DIAG-VARIABLE corresponds to the TABLE that directly (not via <<odxlink>>) contains the TABLE-ROW that is referenced by the DIAG-VARIABLE. Other TABLEs are ignored, even if they reference the same TABLE-ROW element.

A D-server evaluates the referenced TABLE-ROW and by doing this, retrieves the KEY of the TABLE-ROW. The value of the KEY is the value each TABLE-KEY PARAM of the referenced DIAG-COMM's REQUEST shall be set with.

EXAMPLE 1 Use of SNREF-TO-TABLEROW class:

```
<BASE-VARIANT ID="BV_ID_1">
  <SHORT-NAME>BV</SHORT-NAME>
  <LONG-NAME>Base Variant</LONG-NAME>
  <DIAG-DATA-DICTIONARY-SPEC>
    <DATA-OBJECT-PROPS>
      <DATA-OBJECT-PROP ID="DOP_1">
        <SHORT-NAME>KEYDOP</SHORT-NAME>
        <COMPU-METHOD>
          <CATEGORY>TEXTTABLE</CATEGORY>
          <COMPU-INTERNAL-TO-PHYS>
            <COMPU-SCALES>
              <COMPU-SCALE>
                <LOWER-LIMIT>65321</LOWER-LIMIT>
                <UPPER-LIMIT>65321</UPPER-LIMIT>
                <COMPU-CONST>
                  <VT>Engine RPM</VT>
                </COMPU-CONST>
              </COMPU-SCALE>
            </COMPU-SCALES>
          </COMPU-INTERNAL-TO-PHYS>
        </COMPU-METHOD>
        <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
          <BIT-LENGTH>16</BIT-LENGTH>
        </DIAG-CODED-TYPE>
        <PHYSICAL-TYPE BASE-DATA-TYPE="A_UNICODE2STRING"/>
      </DATA-OBJECT-PROP>
      <DATA-OBJECT-PROP ID="DOP_2">
        <SHORT-NAME>DOP_2</SHORT-NAME>
        <COMPU-METHOD>
          <CATEGORY>LINEAR</CATEGORY>
          <COMPU-INTERNAL-TO-PHYS>
```

```
          <COMPU-SCALES>
            <COMPU-SCALE>
              <LOWER-LIMIT>0</LOWER-LIMIT>
              <UPPER-LIMIT>65535</UPPER-LIMIT>
              <COMPU-RATIONAL-COEFFS>
                <COMPU-NUMERATOR>
                  <V>800</V>
                  <V>0.01</V>
                </COMPU-NUMERATOR>
              </COMPU-RATIONAL-COEFFS>
            </COMPU-SCALE>
          </COMPU-SCALES>
        </COMPU-INTERNAL-TO-PHYS>
      </COMPU-METHOD>
      <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
        <BIT-LENGTH>16</BIT-LENGTH>
      </DIAG-CODED-TYPE>
      <PHYSICAL-TYPE BASE-DATA-TYPE="A_FLOAT64">
        <PRECISION>3</PRECISION>
      </PHYSICAL-TYPE>
    </DATA-OBJECT-PROP>
  </DATA-OBJECT-PROPS>
  <STRUCTURES>
    <STRUCTURE ID="STRUCTURE_ID_1">
      <SHORT-NAME>STRUCTURE_Engine_RPM</SHORT-NAME>
      <PARAMS>
        <PARAM xsi:type="VALUE">
          <SHORT-NAME>Engine_RPM</SHORT-NAME>
          <BYTE-POSITION>0</BYTE-POSITION>
          <DOP-SNREF SHORT-NAME="DOP_2"/>
        </PARAM>
      </PARAMS>
    </STRUCTURE>
  </STRUCTURES>
  <TABLES>
    <TABLE ID="TABLE_ID_1">
      <SHORT-NAME>TABLE</SHORT-NAME>
      <KEY-DOP-REF ID-REF="DOP_1" DOCREF="BV" DOCTYPE="LAYER"/>
      <TABLE-ROW ID="TABLE_ROW_ID_1">
        <SHORT-NAME>TABLE_ROW</SHORT-NAME>
        <KEY>Engine RPM</KEY>
        <STRUCTURE-REF ID-REF="STRUCTURE_ID_1" DOCREF="BV" DOCTYPE="LAYER"/>
      </TABLE-ROW>
      <TABLE-DIAG-COMM-CONNECTORS>
        <TABLE-DIAG-COMM-CONNECTOR>
          <SEMANTIC>READ</SEMANTIC>
          <DIAG-COMM-REF ID-REF="DS_1" DOCREF="BV" DOCTYPE="LAYER"/>
        </TABLE-DIAG-COMM-CONNECTOR>
      </TABLE-DIAG-COMM-CONNECTORS>
    </TABLE>
  </TABLES>
</DIAG-DATA-DICTIONARY-SPEC>
<DIAG-COMMS>
  <DIAG-SERVICE ID="DS_1">
    <SHORT-NAME>RDBI</SHORT-NAME>
    <REQUEST-REF ID-REF="REQ_1" DOCREF="BV" DOCTYPE="LAYER"/>
    <POS-RESPONSE-REFS>
      <POS-RESPONSE-REF ID-REF="POS_RE_1" DOCREF="BV" DOCTYPE="LAYER"/>
```

```
      </POS-RESPONSE-REFS>
    </DIAG-SERVICE>
  </DIAG-COMMS>
  <REQUESTS>
    <REQUEST ID="REQUE_1">
      <SHORT-NAME>REQ_1</SHORT-NAME>
      <PARAMS>
        <PARAM xsi:type="CODED-CONST" SEMANTIC="SERVICE-ID">
          <SHORT-NAME>ServiceID</SHORT-NAME>
          <BYTE-POSITION>0</BYTE-POSITION>
          <CODED-VALUE>34</CODED-VALUE>
          <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
            <BIT-LENGTH>8</BIT-LENGTH>
          </DIAG-CODED-TYPE>
        </PARAM>
        <PARAM xsi:type="TABLE-KEY" SEMANTIC="LOCAL-ID" ID="TABLE_KEY_REQUE">
          <SHORT-NAME>Key</SHORT-NAME>
          <BYTE-POSITION>1</BYTE-POSITION>
          <TABLE-REF ID-REF="TABLE_ID_1" DOCREF="BV" DOCTYPE="LAYER"/>
        </PARAM>
      </PARAMS>
    </REQUEST>
  </REQUESTS>
  <POS-RESPONSES>
    <POS-RESPONSE ID="POS_RE_1">
      <SHORT-NAME>POS_RE_1</SHORT-NAME>
      <PARAMS>
        <PARAM xsi:type="CODED-CONST" SEMANTIC="SERVICE-ID">
          <SHORT-NAME>ServiceID</SHORT-NAME>
          <BYTE-POSITION>0</BYTE-POSITION>
          <CODED-VALUE>98</CODED-VALUE>
          <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
            <BIT-LENGTH>8</BIT-LENGTH>
          </DIAG-CODED-TYPE>
        </PARAM>
        <PARAM xsi:type="TABLE-KEY" SEMANTIC="LOCAL-ID" ID="TABLE_KEY_RESPO">
          <SHORT-NAME>Key</SHORT-NAME>
          <BYTE-POSITION>1</BYTE-POSITION>
          <TABLE-REF ID-REF="TABLE_ID_1" DOCREF="BV" DOCTYPE="LAYER"/>
        </PARAM>
        <PARAM xsi:type="TABLE-STRUCT" SEMANTIC="DATA">
          <SHORT-NAME>Data</SHORT-NAME>
          <BYTE-POSITION>3</BYTE-POSITION>
          <TABLE-KEY-REF ID-REF="TABLE_KEY_RESPO" DOCREF="BV" DOCTYPE="LAYER"/>
        </PARAM>
      </PARAMS>
    </POS-RESPONSE>
  </POS-RESPONSES>
  <DIAG-VARIABLES>
    <DIAG-VARIABLE ID="DV_ID_1">
      <SHORT-NAME>DV</SHORT-NAME>
      <SNREF-TO-TABLEROW>
        <TABLE-SNREF SHORT-NAME="TABLE"/>
        <TABLE-ROW-SNREF SHORT-NAME="TABLE_ROW"/>
      </SNREF-TO-TABLEROW>
    </DIAG-VARIABLE>
  </DIAG-VARIABLES>
</BASE-VARIANT>
```

Figure 117 — SNREF-TO-TABLEROW mechanism visualizes the SNREF-TO-TABLEROW mechanism and how to use it.

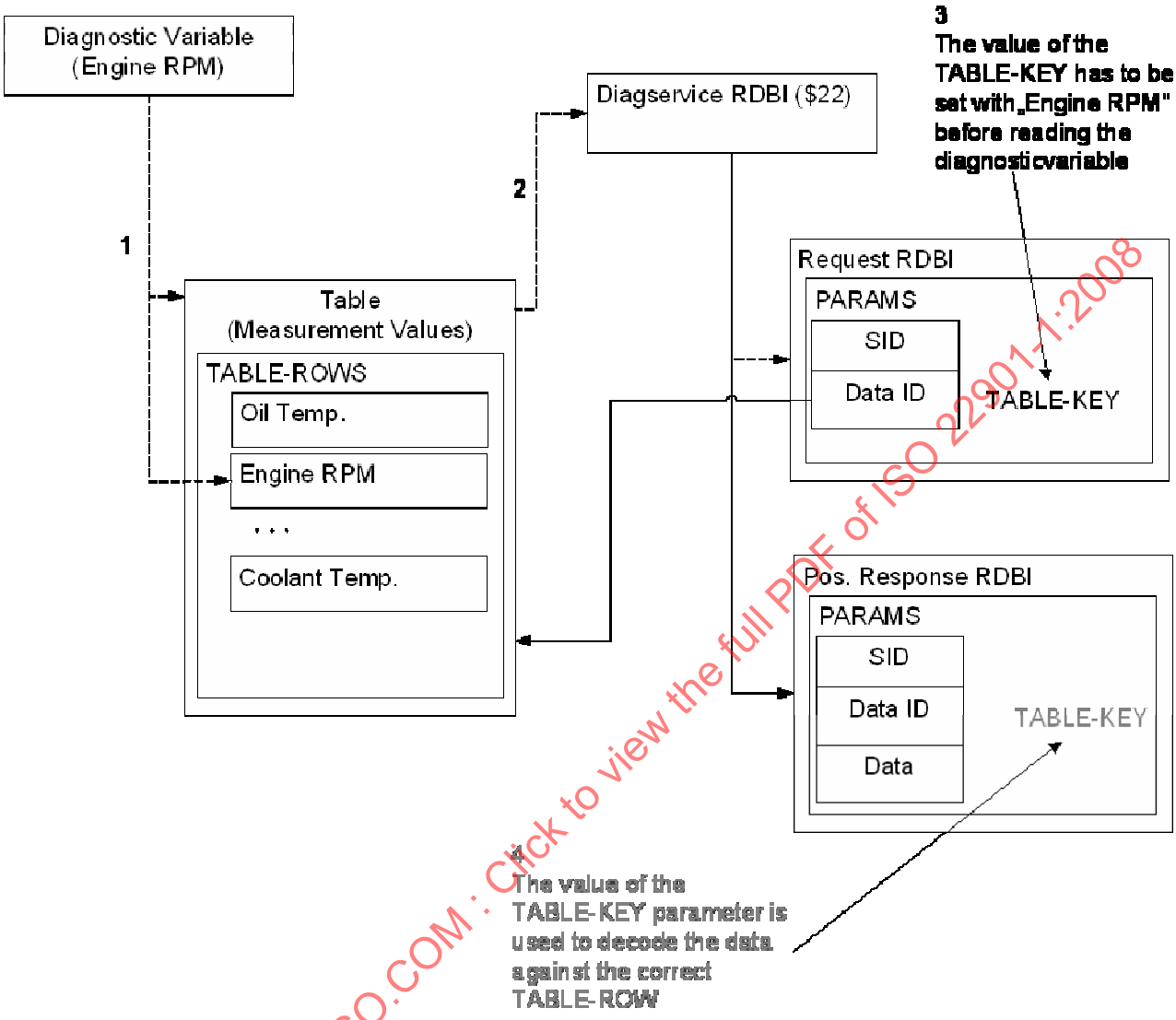


Figure 117 — SNREF-TO-TABLEROW mechanism

If a DIAG-VARIABLE shall not be written without reading it beforehand, IS-READ-BEFORE-WRITE attribute shall be set to “true”. For example, if a DIAG-VARIABLE represents a data set and a single item in this set is to be changed, the whole set shall be read and only then the item can be changed and the variable can be written. By default the value “false” is assumed for this attribute.

Typically a DIAG-VARIABLE has a simple data type like integer or string. If the COMM-RELATION mechanism is used, the conversion between the coded and the physical representation for such a DIAG-VARIABLE is done by the DATA-OBJECT-PROP that is associated with the PARAM specified by IN-PARAM-IF-SNREF and OUT-PARAM-IF-SNREF. Both shall have identical definitions of PHYSICAL-TYPE. PARAMs with complex DOPs (e.g. STATIC-FIELD or MUX) are not allowed to be referenced.

When using the SNREF-TO-TABLEROW option, the conversion is entirely handled by the specified data related to the referenced TABLE-ROW, e.g. the PARAM objects that are contained in the referenced STRUCTURE (STRUCTURE-REF in TABLE-ROW) define the conversion between the coded and physical representation.

The VALUE-TYPE defines how the D-server should handle the transfer of DIAG-VARIABLE values between the ECU and the tester, as described below.

— CURRENT: The value should be retrieved from the tester/ECU every time when the DIAG-VARIABLE is accessed. The D-server sends/receives the actual value. This is the default if no VALUE-TYPE is specified.

— MEASUREMENT: The values describe measurement services, tables, etc.

— STORED: When a changed value is retrieved from the tester, the D-server should store this value for further use. The stored value is sent to the ECU.

— STATIC: The value should be retrieved once from the tester/ECU when the DIAG-VARIABLE is accessed for the first time. After that, the value cannot be changed anymore. An example for a static value is an ECU-internal identifier.

— SUBSTITUTED: These values are neither current nor stored but values delivered by the ECU as substituted (or intermediate) values.

A SW-VARIABLE object describes a SW-variable that is available as a DIAG-VARIABLE, too. The identifier of the SW-variable specified in the ECU software is used as SHORT-NAME. In addition, the ORIGIN of the SW-variable, i.e. the ECU software release containing the SW-variable, can be defined.

A DIAG-VARIABLE may be associated with an arbitrary number of SW-VARIABLEs indicating an existing mapping between the represented SW-variables and the DIAG-VARIABLE. One SW-VARIABLE object exists for each SW-variable within specific ECU software that is mapped to the DIAG-VARIABLE. That way, many SW-VARIABLEs may be associated with the same DIAG-VARIABLE. If no SW-VARIABLE is given, the DIAG-VARIABLE is only available in the diagnostic use case.

A DIAG-VARIABLE can refer to a VARIABLE-GROUP. Each VARIABLE-GROUP denotes a category of DIAG-VARIABLEs, e.g. measurement variables, calibration parameters, or compound variables. The introduction of VARIABLE-GROUPs is a means to handle large numbers of DIAG-VARIABLEs.

EXAMPLE 2 Mapping of two SW-variables (defined by suppliers A and B) to a DIAG-VARIABLE (defined by OEM C):

```
<ECU-VARIANT ID = "Engine_ECU_Variant">
    <SHORT-NAME>Engine_ECU_Variant</SHORT-NAME>
    <DIAG-VARIABLES>
        <DIAG-VARIABLE ID = "DV_PHY_TMOT" IS-READ-BEFORE-WRITE = "false">
            <SHORT-NAME>DV_PHY_TMOT</SHORT-NAME>
            <VARIABLE-GROUP-REF ID-REF = "VG_MEAS"/>
            <SW-VARIABLES>
                <SW-VARIABLE>
                    <SHORT-NAME>tmot</SHORT-NAME>
                    <ORIGIN>A</ORIGIN>
                </SW-VARIABLE>
                <SW-VARIABLE>
                    <SHORT-NAME>TMOTK</SHORT-NAME>
                    <ORIGIN>B</ORIGIN>
                </SW-VARIABLE>
            </SW-VARIABLES>
            <COMM-RELATIONS>
                <COMM-RELATION VALUE-TYPE = "CURRENT">
                    <RELATION-TYPE>READ</RELATION-TYPE>
                    <DIAG-COMM-SNREF SHORT-NAME = "DS_ReadMotorTemperature"/>
                    <OUT-PARAM-IF-SNREF SHORT-NAME = "STAT_PHY_TMOT"/>
                </COMM-RELATION>
                <COMM-RELATION VALUE-TYPE = "CURRENT">
                    <RELATION-TYPE>WRITE</RELATION-TYPE>
```

```
                    <DIAG-COMM-SNREF SHORT-NAME = "DS_WriteMotorTemperature"/>
                    <IN-PARAM-IF-SNREF SHORT-NAME = "ARG_PHY_TMOT"/>
                </COMM-RELATION>
                <COMM-RELATION VALUE-TYPE = "CURRENT">
                    <RELATION-TYPE>RETURNCONTROL</RELATION-TYPE>
                    <DIAG-COMM-SNREF SHORT-NAME = "DS_WriteMotorTemperature_END"/>
                </COMM-RELATION>
            </COMM-RELATIONS>
        </DIAG-VARIABLE>
    </DIAG-VARIABLES>
    <VARIABLE-GROUPS>
        <VARIABLE-GROUP ID = "VG_MEAS">
            <SHORT-NAME>VG_MEAS</SHORT-NAME>
            <LONG-NAME>measurement variable group</LONG-NAME>
        </VARIABLE-GROUP>
    </VARIABLE-GROUPS>
</ECU-VARIANT>
```

7.3.7.4 Inheritance of DIAG-VARIABLEs

DIAG-VARIABLEs are inherited like diagnostic services or jobs, they can be reused or overridden at lower layers. The inheritance of a DIAG-VARIABLE from the PARENT layer can be eliminated via the NOT-INHERITED-VARIABLE class. For the following use scenarios, we assume that the DIAG-VARIABLE is defined in a BASE-VARIANT instance.

If the DIAG-VARIABLE is to be reused in the ECU-VARIANT instance, no further change is necessary. In case that one of the referenced DIAG-COMMs is overridden in the same ECU-VARIANT instance the association defined by the corresponding COMM-RELATION is overridden; the COMM-RELATION refers to the new DIAG-COMM. The overriding DIAG-COMM shall have identical SHORT-NAMEs of the PARAMS that are used by DIAG-VARIABLE.

The overriding of a DIAG-VARIABLE is required when a COMM-RELATION or an associated SW-VARIABLE is changed. In this case the complete DIAG-VARIABLE is overridden including its COMM-RELATIONs. As a consequence, all references to services or TABLEs and to parameters shall be specified again for the ECU-VARIANT even if they have not changed from the BASE-VARIANT.

7.3.8 Dynamically defined messages

With some protocols (e.g. ISO 14230 or ISO 14229-1) it is possible to dynamically define data records at run time, which contain values from other records identified by a local identifier (LOCAL-ID), a common identifier (COMMON-ID) or an address in the ECU memory, etc. The new data records are identified by a DYN-ID (dynamically defined identifier). The object DYN-DEFINED-SPEC lists all supported dynamically defined identifiers of the ECU (SUPPORTED-DYN-ID) and references all TABLEs, which contain information about existing data records and can be used for creation of new DYN-ID records. The supported DYN-IDs and the TABLEs are grouped by the definition mode. Therefore the class DYN-ID-DEF-MODE-INFO was introduced as a wrapper for one group of DYN-IDs and the appropriate TABLEs. Its member DEF-MODE indicates the definition mode: by LOCAL-ID, COMMON-ID, etc. Each instance of DYN-ID-DEF-MODE-INFO refers to three DIAG-COMMs used to define new DYN-IDs, to read data using a DYN-ID and to clear a DYN-ID defined before. The DIAGNOSTIC-CLASS of these DIAG-COMMs shall have the values DYN-DEF-MESSAGE, READ-DYN-DEF-MESSAGE or CLEAR-DYN-DEF-MESSAGE respectively. Within one DYN-DEFINED-SPEC there should not exist multiple DYN-ID-DEF-MODE-INFO instances with the same DEF-MODE value.

The DYN-DEFINED-SPEC shall be defined in a BASE-VARIANT or ECU-VARIANT. There exists only one object of that class within each DIAG-LAYER because a DYN-DEFINED-SPEC has no SHORT-NAME. The DIAG-COMMs referenced by DYN-ID-DEF-MODE-INFO have to carry the appropriate DIAGNOSTIC-CLASS=“DYN-DEF-MESSAGE”, “READ-DYN-DEF-MESSAGE” or “CLEAR-DYN-DEF-MESSAGE”. All TABLEs used for creation of new DYN-ID records shall be referenced from this DYN-DEFINED-SPEC. If an ECU-VARIANT does not have an own DYN-DEFINED-SPEC, the DYN-DEFINED-SPEC of the parent BASE-

VARIANT is valid. If the ECU-VARIANT defines its own DYN-DEFINED-SPEC, it completely replaces the one of the BASE-VARIANT. See 7.4.2 to get an example for use of DYN-DEFINED-SPEC. The definition of an empty DYN-DEFINED-SPEC at the ECU-VARIANT means that this ECU-VARIANT does not support the mechanism of the DYN-DEFINED-SPEC independent of the BASE-VARIANT.

See Figure 118 — UML representation of DYN-DEFINED-SPEC.

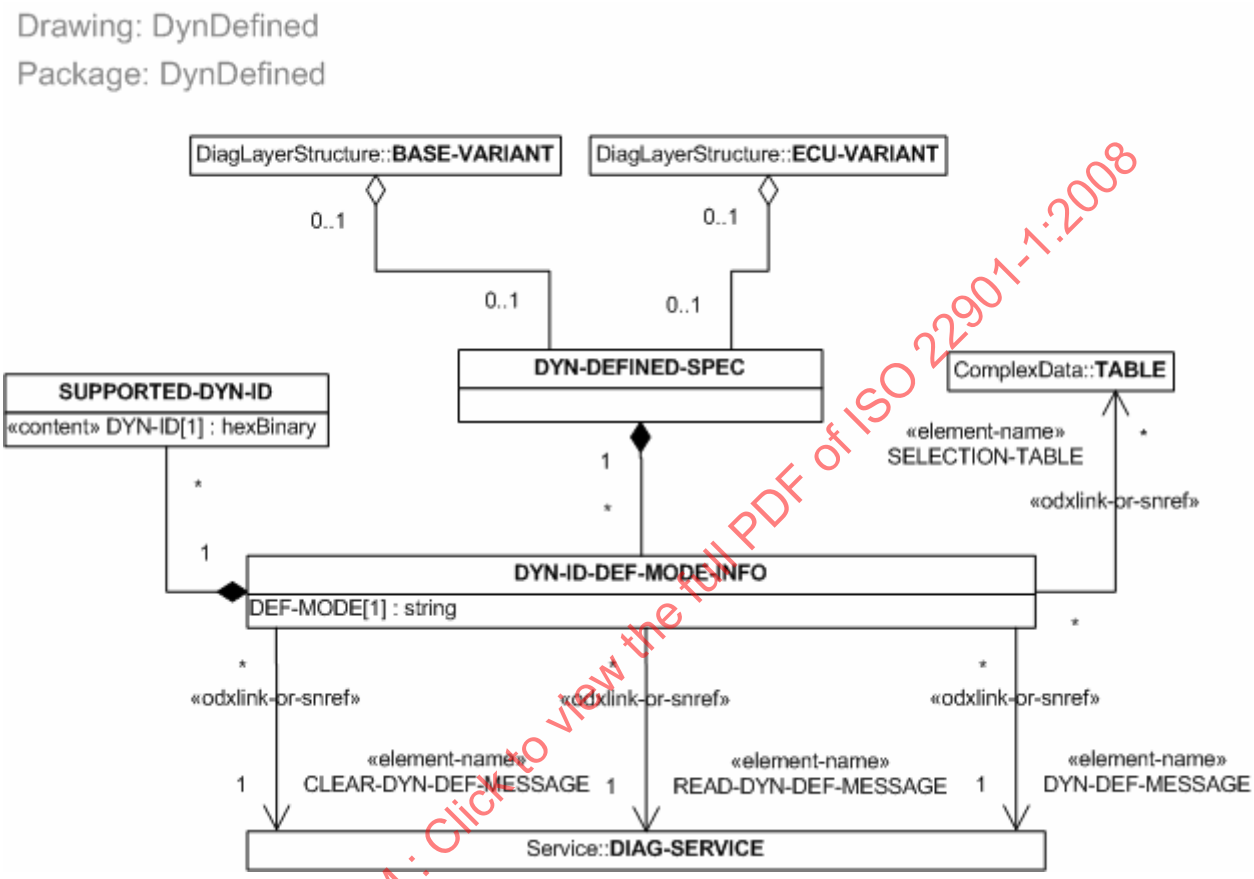


Figure 118 — UML representation of DYN-DEFINED-SPEC

7.3.9 Session and security handling

The session and security sub model covers two aspects:

a) to describe possible state transitions resulting from the execution of a DIAG-COMM;

b) to describe preconditions for the execution of a DIAG-COMM.

Figure 119 — UML representation of session security and STATE-CHART shows the data model for the two features.

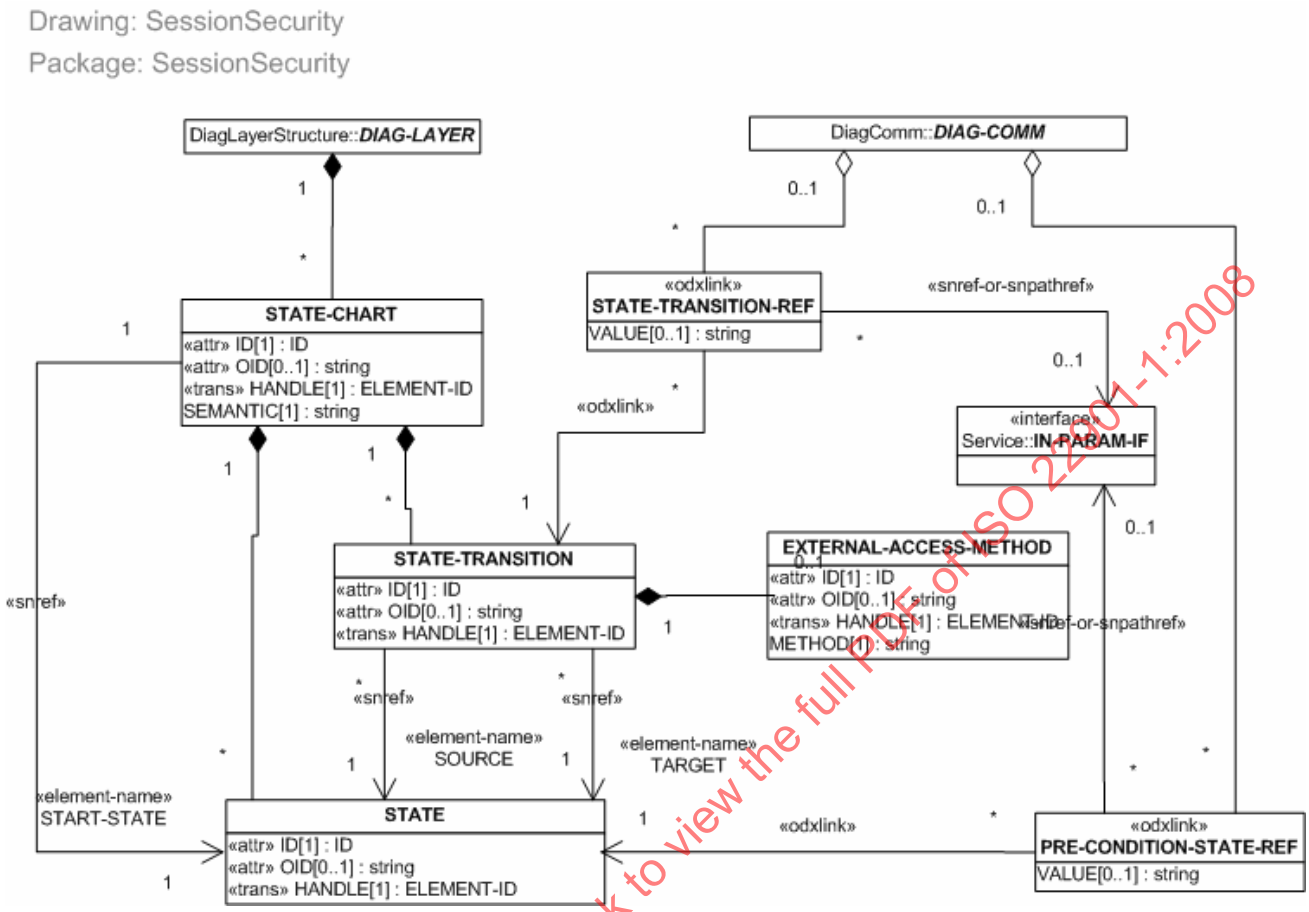


Figure 119 — UML representation of session security and STATE-CHART

Both the session and the security handling are modelled within one state machine model. The element STATE-CHART stores the possible STATEs of an ECU. SEMANTIC defines the type of the STATE-CHART. It has the predefined values “SESSION” and “SECURITY”. SEMANTIC is extendible by the user. The start state of the state machine is defined by the START-STATE snref at STATE-CHART. A state transition within one state machine is modelled by the STATE-TRANSITION element at STATE-CHART. Each STATE-TRANSITION references one source (SOURCE) and one target state (TARGET). A STATE-TRANSITION contains an optional EXTERNAL-ACCESS-METHOD element.

The EXTERNAL-ACCESS-METHOD makes it possible to specify an OEM-specific method necessary to perform the STATE-TRANSITION. For security-critical applications this link can be used. The attribute METHOD specifies which method is to be used. The values or METHOD need to be defined OEM-specifically and the semantics of every method shall be implemented into an OEM-specific extension of a D-server.

STATE-TRANSITION-REFs and PRE-CONDITION-STATE-REFs can be used at DIAG-COMMs. If STATE-TRANSITION-REF is used, this has the following semantic: If the DIAG-COMM is successfully executed, the state of the ECU changes from the specified SOURCE to the specified TARGET state. A DIAG-COMM is executable in all SOURCE states defined in its referenced STATE-TRANSITIONs. Self transitions can be described by STATE-TRANSITIONs with identical SOURCE and TARGET states.

PRE-CONDITION-STATE-REF can be used to define the allowed states for the execution of the DIAG-COMM in cases where the execution of the DIAG-COMM does not result in a state-transition of the ECU but its execution is bound to the condition that the ECU already is in a certain state. If a STATE-TRANSITION-REF is used, there may but does not have to be a PRE-CONDITION-STATE-REF for the source states of the

referenced transitions. If neither STATE-TRANSITION-REF nor PRE-CONDITION-STATE-REF are used, the DIAG-COMM is executable in all states and the execution does not result in a state transition. If STATE-TRANSITION-REFs and/or PRE-CONDITION-STATE-REFs are used, the DIAG-COMM is executable in the SOURCE states of the referenced STATE-TRANSITIONs and in the referenced preconditions' STATEs but no other states.

The optional VALUE together with the also optional IN-PARAM-IF snref at STATE-TRANSITION-REF and PRE-CONDITION-STATE-REF can be used if the STATE-TRANSITIONs and pre-condition STATEs are dependent on the values of the referenced PARAMs.

An example for the handling of sessions and security is accomplished in Annex G.

7.3.10 Vehicle information

7.3.10.1 Vehicle identification data

The vehicle information specification serves two purposes:

a) to identify a vehicle through the fixing of a set of values (see data model in Figure 120);

b) to give a D-server access to the vehicle by describing the vehicle topology (see data model in Figure 121).

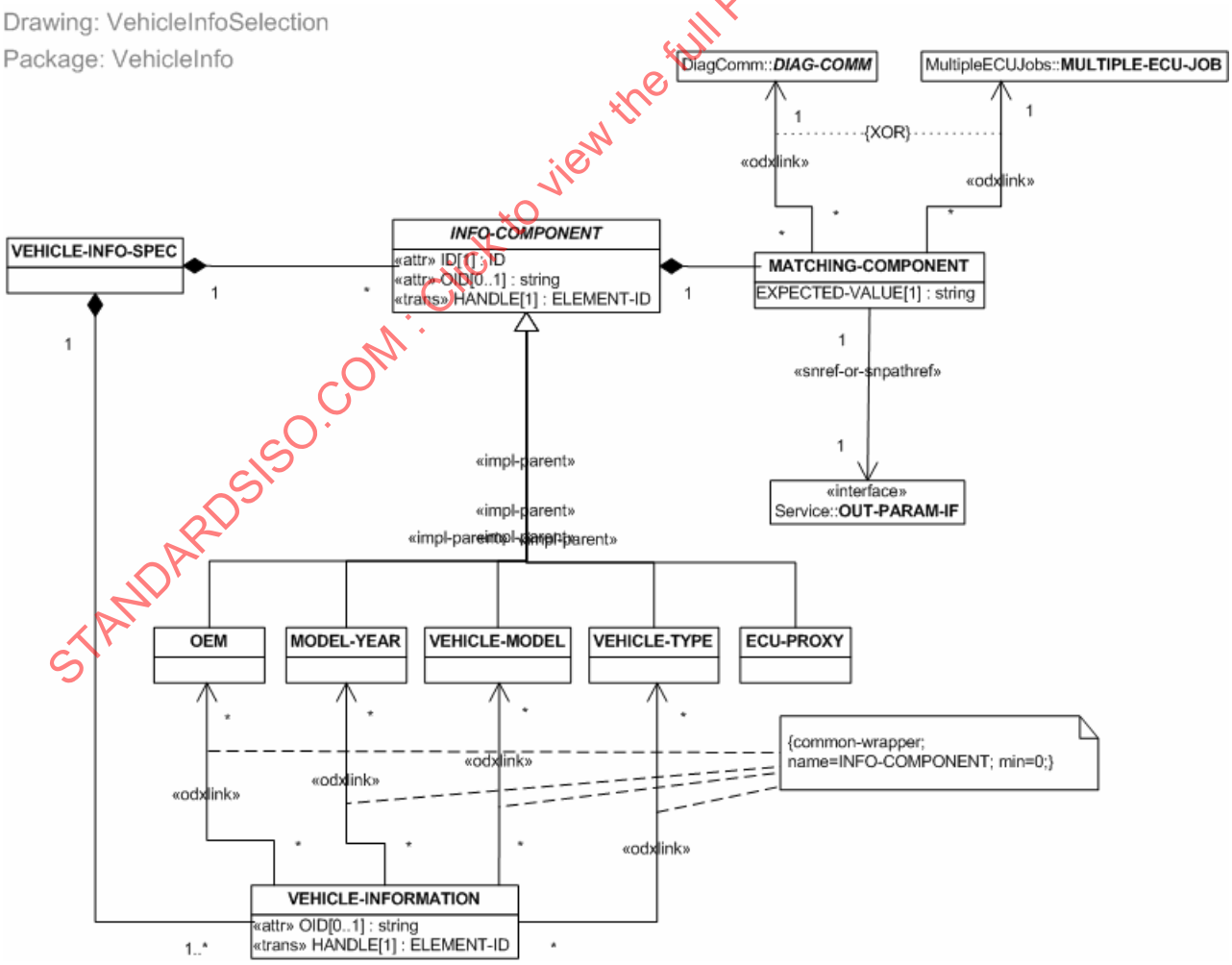


Figure 120 — UML representation of vehicle information selection

The vehicle identification data model consists of a set of INFO-COMPONENTs. An INFO-COMPONENT describes one aspect of a vehicle that narrows down the set of possible vehicle topologies (VEHICLE-INFORMATION). Currently, the following INFO-COMPONENTS have been modelled: the OEM (e.g. “Volkswagen”), the MODEL-YEAR (e.g. “2003”), the VEHICLE-MODEL (e.g. “Golf V”) and the VEHICLE-TYPE (e.g. “passenger car”). A special kind of INFO-COMPONENT is the so-called ECU-PROXY. It is used to establish an alternative description for a family of built-in ECUs for external presentation. A vehicle may have multiple possible engines with different engine control ECUs. Within the BASE-VARIANTs only internal names for these ECUs are used (e.g. ZXH20). If, for vehicle identification purposes, it is necessary for a service technician to enter some information about the engine manually, because it cannot be read out automatically, he will only be able to enter the externally known name of the engine (e.g. “1.9 TDI”). This is made possible through an ECU-PROXY.

If all INFO-COMPONENTS are identified (a value has been assigned to them), ideally one or possibly a small set of possible vehicle topology descriptions (instances of VEHICLE-INFORMATION) has been determined. This is modelled through the references from the VEHICLE-INFORMATION class to the different INFO-COMPONENTs. ECU-PROXYs are referenced indirectly through the LOGICAL-LINKs contained within a VEHICLE-INFORMATION specification.

The values for the INFO-COMPONENTs may be established in two different ways:

— by interactive data entry, e.g. by the service technician, or

— by communication with the vehicle.

For the latter case, every INFO-COMPONENT may contain an arbitrary number of MATCHING-COMPONENTs. These contain references to output or response parameters of jobs or diagnostic services, respectively. An expected value is specified for every MATCHING-COMPONENT. If this expected value is matched by the output or response parameter at runtime, a specific INFO-COMPONENT is found. Between the different MATCHING-COMPONENTs of one INFO-COMPONENT a logical AND-constraint is assumed. This means a specific INFO-COMPONENT is established, if all contained MATCHING-COMPONENTs actually matched.

7.3.10.2 Vehicle topology

The vehicle topology data model’s main element is the VEHICLE-INFORMATION. One instance of VEHICLE-INFORMATION describes the vehicle topology of one designated set of vehicles, e.g. all vehicles of one model within one model year. A VEHICLE-INFORMATION contains information about the VEHICLE-CONNECTOR, the available PHYSICAL-VEHICLE-LINKs and the available LOGICAL-LINKs.

See Figure 121 — UML representation of vehicle information for further detail.

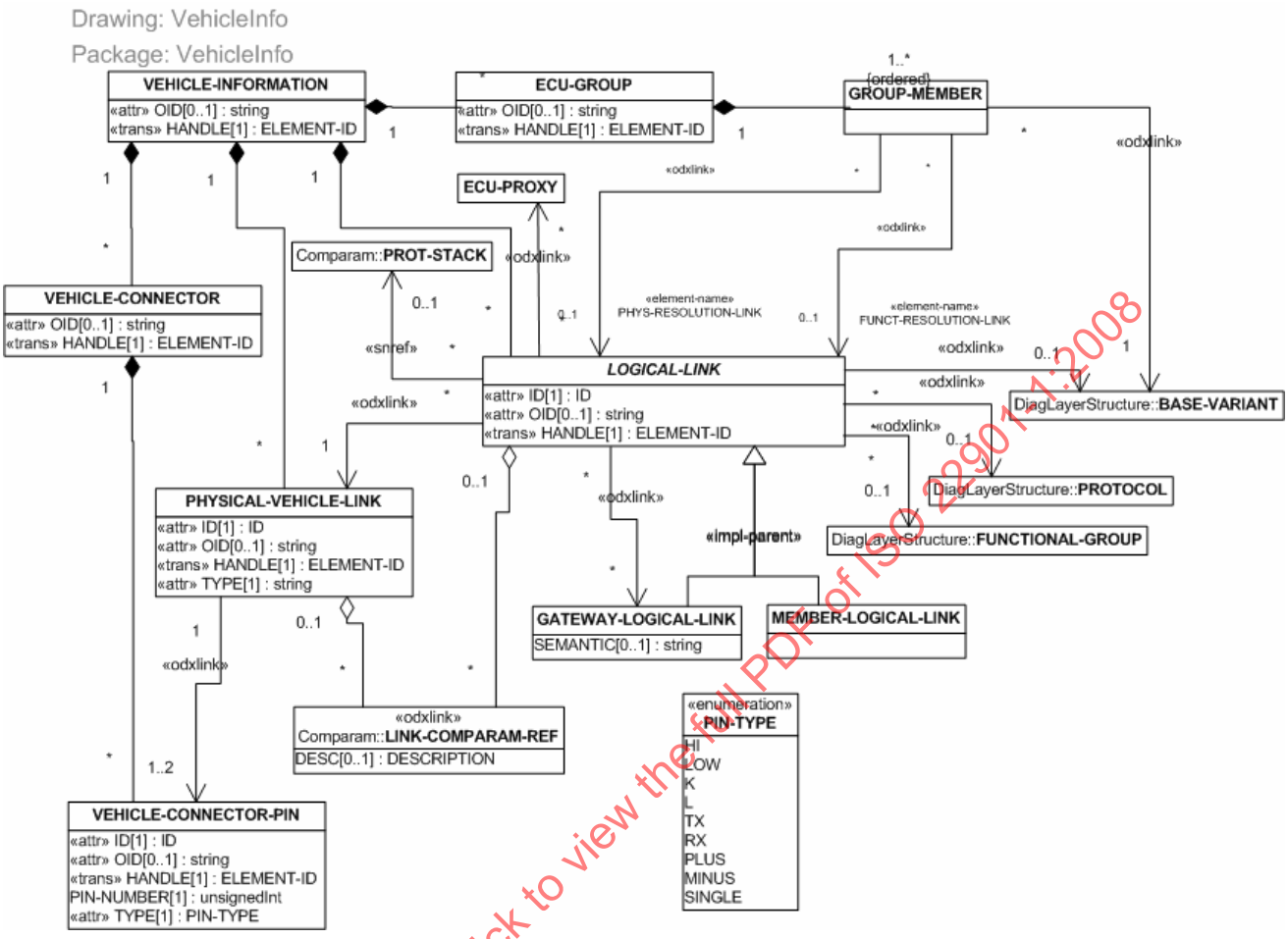


Figure 121 — UML representation of vehicle information

A VEHICLE-CONNECTOR contains a description of its pins (VEHICLE-CONNECTOR-PIN). Every pin has a number (PIN-NUMBER). This number shall be unique within one VEHICLE-CONNECTOR. A PHYSICAL-VEHICLE-LINK has a TYPE, which describes the physical communication layer usable on that link (e.g. K-LINE or CAN) and identifies the VEHICLE-CONNECTOR-PIN that shall be used in order to communicate via this PHYSICAL-VEHICLE-LINK.

A LOGICAL-LINK is the representation of one access path to a protocol usable for ECU communication, a functional group of ECUs or an ECU base variant. Multiple references to DIAG-LAYERs shall be used to specify the complete access information to an ECU. As it is possible for a BASE-VARIANT to implement multiple protocols, one PROTOCOL shall explicitly be referenced by a logical link to determine which protocol is to be used on this access path to the ECU. A LOGICAL-LINK can either reference a BASE-VARIANT alone or a PROTOCOL alone or a PROTOCOL and a BASE-VARIANT or a PROTOCOL and a FUNCTIONAL-GROUP. A reference to a BASE-VARIANT and PROTOCOL has the implicit meaning, that all FUNCTIONAL-GROUPS that inherit from that PROTOCOL and to that BASE-VARIANT are valid.

Two subclasses of LOGICAL-LINK exist. One subclass to identify a logical link to a gateway ECU (GATEWAY-LOGICAL-LINK) and one subclass to identify a regular ECU that may be accessed either directly or through a gateway (MEMBER-LOGICAL-LINK). Every logical link may reference other logical links. The referenced target identifies a gateway through which the ECU represented by the reference source can be accessed. By letting a GATEWAY-LOGICAL-LINK reference another GATEWAY-LOGICAL-LINK, a hierarchy of gateways can be specified.

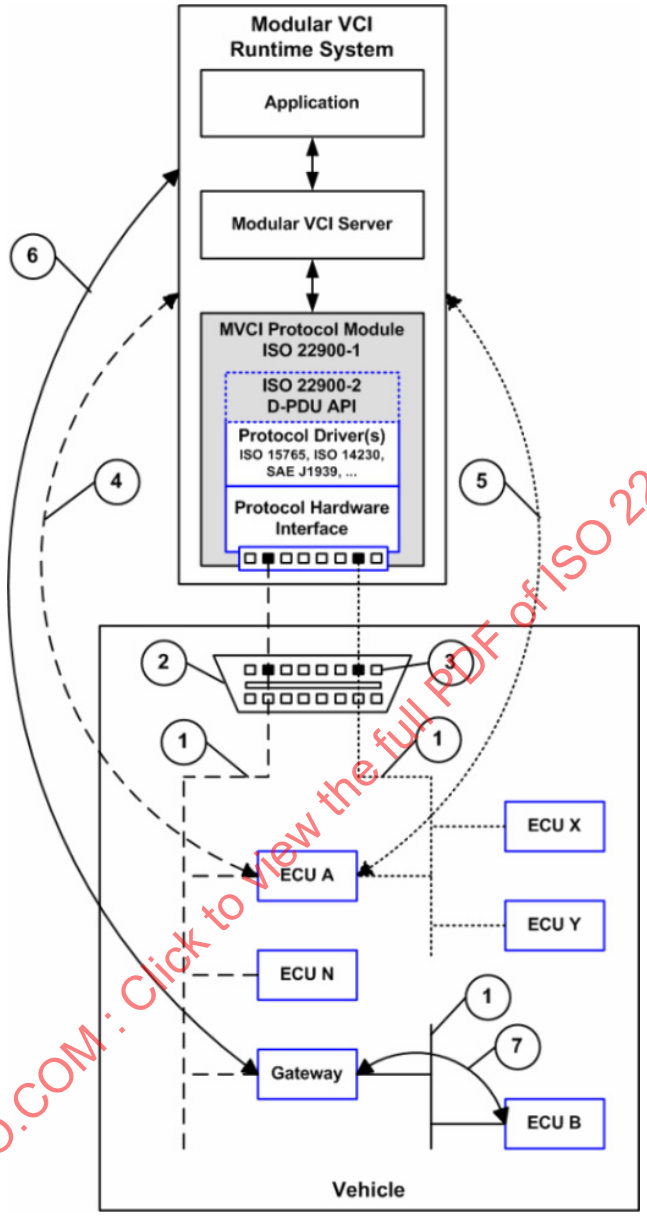
The optional reference to a PROT-STACK indicates which set of protocol specific communication parameters are used for a certain LOGICAL-LINK. LOGICAL-LINKs and PHYSICAL-VEHICLE-LINKs may both reference COMPARAMs. COMPARAMs at the physical vehicle link describe baud rates, timings etc. that are dependent on the physical capabilities of the hardware link (e.g. the CAN bus or the K-Line). COMPARAMs at the logical link are used to define communication parameters dependent on the built-in of an ECU into the vehicle (e.g. behind a gateway or not).

An ECU-GROUP makes it possible to group different BASE-VARIANTs together that are to be seen as alternatives within the VEHICLE-INFORMATION. Consider a VEHICLE-INFORMATION element describing a complete vehicle model and this vehicle model has an alternative built-in of ECUs serving the same purpose (e.g. multiple different radios). These ECUs even can have the same diagnostic address. In this case a separate logical link for each one of these radios would be contained in the VEHICLE-INFORMATION. However, since only one of the radios can actually occur in a given vehicle, all these logical links are grouped in an ECU-GROUP. Every GROUP-MEMBER of an ECU-GROUP therefore references a BASE-VARIANT and a LOGICAL-LINK since the same BASE-VARIANT may be accessible through multiple LOGICAL-LINKs. Every ECU-GROUP within one VEHICLE-INFO-SPEC needs to be resolved, before functional communication can consider overridden response messages. The references from GROUP-MEMBER to LOGICAL-LINK named PHYS-RESOLUTION-LINK-REF and FUNCT-LOGICAL-LINK-REF indicate whether this resolution is done by physical or functional addressing. For base variant identification the MATCHING-PARAMETERs of the BASE-VARIANT-PATTERNs at each corresponding BASE-VARIANT are used (see also 7.3.2.8 concerning base variant identification). Either a PHYS-RESOLUTION-LINK or a FUNCT-RESOLUTION-LINK shall exist for every GROUP-MEMBER. The FUNCT-RESOLUTION-LINK points to a LOGICAL-LINK that references a FUNCTIONAL-GROUP and no BASE-VARIANT. The PHYS-RESOLUTION-LINK points to a LOGICAL-LINK that references a BASE-VARIANT.

7.3.10.3 Runtime behaviour

Currently, the D-server is only able to use the vehicle topology data and not the vehicle identification data. Communication with ECUs within the vehicle is established by opening a logical link. The data of the location (e.g. a base variant) referenced by the logical link is then loaded into the system and the location's defined DIAG-COMMs can be executed.

See Figure 122 — Vehicle topology example at runtime behaviour.



Key

1 PHYSICAL-VEHICLE-LINK

2 VEHICLE-CONNECTOR

3 VEHICLE-CONNECTOR-PIN

4 LOGICAL-LINK 1 to ECU A

5 LOGICAL-LINK 2 to ECU A

6 LOGICAL-LINK to ECU B

7 GATEWAY-LOGICAL-LINK-REF(ERENCE) to ECU B

Figure 122 — Vehicle topology example

7.3.11 Multiple ECU jobs

The MULTIPLE-ECU-JOB-SPEC is one kind of ODX-CATEGORY. An arbitrary number of such jobs may be contained in one MULTIPLE-ECU-JOB-SPEC. See Figure 123 — UML representation of multiple ECU jobs.

Figure 123 — UML representation of multiple ECU jobs

In contrast to SINGLE-ECU-JOBs (see 7.3.5.7), the code of MULTIPLE-ECU-JOBs contains calls to DIAG-COMMs of more than one ECU. Consequently, a MULTIPLE-ECU-JOB cannot be assigned to the diagnostic data specification of one dedicated ECU (e.g. a BASE-VARIANT). At execution time, a MULTIPLE-ECU-JOB is not bound to one logical link (as a SINGLE-ECU-JOB is) within the D-server. Rather, a MULTIPLE-ECU-JOB may open and close logical links deliberately and communicate to multiple ECUs simultaneously. An application domain for a MULTIPLE-ECU-JOB is the identification of a vehicle, where the job communicates to multiple ECU base variants to identify the vehicle model, make and year.

Multiple ECU jobs are an alternative way to communicate with multiple ECUs next to functional services. However, they do not use functional addressing.

As MULTIPLE-ECU-JOBs cannot be assigned to one ECU, they cannot be included in a DIAG-LAYER specification. In fact, they are a separate ODX-CATEGORY. In addition to ADMIN-DATA and COMPANY-DATA, which are inherited from ODX-CATEGORY, the MULTIPLE-ECU-JOB-SPEC class also, has an aggregation to a DIAG-DATA-DICTIONARY-SPEC. This is needed to specify the data structures for the input and output parameter structures of the contained MULTIPLE-ECU-JOBs. Furthermore, new functional classes (FUNCT-CLASS) can be defined within a MULTIPLE-ECU-JOB-SPEC. In addition, multiple ECU-SHARED-DATA layers may be imported via IMPORT-REF. Only elements within these ECU-SHARED-DATA or within the MULTIPLE-ECU-JOB-SPEC itself can be target of <<odxlink>> originated in the MULTIPLE-ECU-JOB-SPEC.

The class MULTIPLE-ECU-JOB is similar to a SINGLE-ECU-JOB and thus aggregates ADMIN-DATA, INPUT-PARAMs, OUTPUT-PARAMs as well as NEG-OUTPUT-PARAMs, AUDIENCE and SDG classes. A MULTIPLE-ECU-JOB may also reference a set of functional classes (FUNCT-CLASS). However, a MULTIPLE-ECU-JOB is extended with an association to one or multiple DIAG-LAYERs. Within an instance, the references to DIAG-LAYERs are used to specify data of which DIAG-LAYERs is needed for the execution of the MULTIPLE-ECU-JOB. Thus, if a MULTIPLE-ECU-JOBs code calls a diagnostic service of a BASE-VARIANT called Engine_Control_Module, the data specification for this MULTIPLE-ECU-JOB shall contain a reference to the Engine_Control_Module BASE-VARIANT diagnostic layer. The consistency between the code of the MULTIPLE-ECU-JOB and the specified parameter and DIAG-LAYER reference data shall be maintained manually or by a separate analysis tool. A tool compliant with the standard does not have to check this consistency.

EXAMPLE The following example shows the structure of a MULTIPLE-ECU-JOB-SPEC diagnostic data specification. The only contained MULTIPLE-ECU-JOB returns a vehicle's model, make and year. To perform this vehicle identification, the job needs to retrieve data from two ECUs. Their respective base variant data specifications are referenced within the DIAG-LAYER-REFs part. As vehicle identification is a task mainly performed at the dealerships and service bays, the audience is restricted to aftermarket and aftersales.

```
<ODX xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="odx.xsd" MODEL-
VERSION="2.2.0">
    <MULTIPLE-ECU-JOB-SPEC ID="ID_73103450">
        <SHORT-NAME>MEJS</SHORT-NAME>
        <MULTIPLE-ECU-JOBS>
            <MULTIPLE-ECU-JOB ID="ID_73101" SEMANTIC="IDENTIFICATION">
                <SHORT-NAME>Vehicle_Identification</SHORT-NAME>
                <PROG-CODES>
                    <PROG-CODE>
                        <CODE-FILE>Vehicle_Ident</CODE-FILE>
```

```
                    <SYNTAX>CLASS</SYNTAX>
                    <REVISION>1.3.4</REVISION>
                </PROG-CODE>
            </PROG-CODES>
            <OUTPUT-PARAMS>
                <OUTPUT-PARAM ID="ID_73107">
                    <SHORT-NAME>Vehicle_Model</SHORT-NAME>
                    <LONG-NAME>Name of vehicle model</LONG-NAME>
                    <DOP-BASE-REF ID-REF="ID_73102"/>
                </OUTPUT-PARAM>
                <OUTPUT-PARAM ID="ID_73108">
                    <SHORT-NAME>Vehicle_Manufacturer</SHORT-NAME>
                    <LONG-NAME>Name of vehicle manufacturer</LONG-NAME>
                    <DOP-BASE-REF ID-REF="ID_73103"/>
                </OUTPUT-PARAM>
                <OUTPUT-PARAM ID="ID_73109">
                    <SHORT-NAME>Vehicle_Year</SHORT-NAME>
                    <LONG-NAME>Year vehicle has been manufactured</LONG-NAME>
                    <DOP-BASE-REF ID-REF="ID_73104"/>
                </OUTPUT-PARAM>
            </OUTPUT-PARAMS>
            <DIAG-LAYER-REFS>
                <DIAG-LAYER-REF ID-REF="ID_73105" DOCREF="ECM" DOCTYPE="CONTAINER" REVISION="2.1.3"/>
                <DIAG-LAYER-REF ID-REF="ID_73106" DOCREF="BCM" DOCTYPE="LAYER" REVISION="4.1.5"/>
            </DIAG-LAYER-REFS>
            <AUDIENCE IS-SUPPLIER="false" IS-DEVELOPMENT ="false" IS-MANUFACTURING="false"/>
        </MULTIPLE-ECU-JOB>
    </MULTIPLE-ECU-JOBS>
</MULTIPLE-ECU-JOB-SPEC>
</ODX>
```

7.3.12 Data types

There are a few type definitions that are used throughout the data model to ensure consistent structures.

See Figure 124 — UML representation of basic data types.

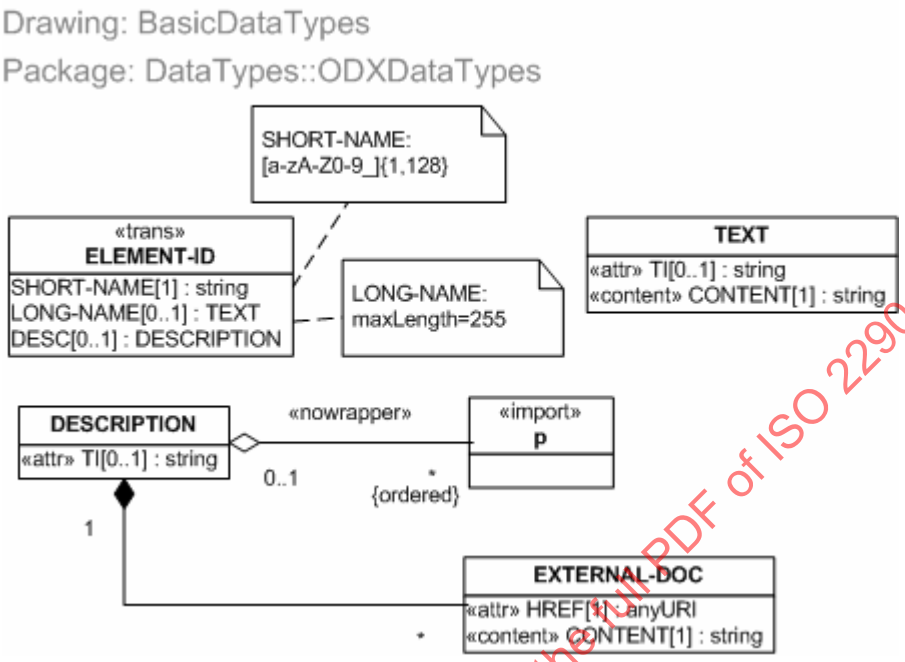


Figure 124 — UML representation of basic data types

The data type ELEMENT-ID is a representation of the three elements SHORT-NAME, LONG-NAME and DESC (see also 7.1.1).

The DESCRIPTION data type is used for all DESC elements and provides a method to add formatted documentation to the corresponding data element. The sub-element EXTERNAL-DOC can be used to refer to any kind of external document. With the HREF attribute, the document is addressed and the CONTENT may provide a description or meta information. The external document can be a file or anything else. The manner in which to resolve this element is not specified by the document. It can be done in a system specific manner and is not a mandatory feature of an ODX compliant tool.

The data type p represents the <p> tag from XHTML. Only a subset of what is defined with the XHTML recommendation can be used to capture formatted documentation.

The TEXT data type is being used at all places where an identifier needs to be assigned to a string e.g. to support internationalisation.

The EMPTY data type is being used to denote an empty content model within the XML implementation.

7.3.13 References

7.3.13.1 Overview

Some parts of an ODX document like UNITs or DOPs may be needed more than once. Therefore they can be reused by reference. The referenced part is called “link target”; the ODX element that references the link target is called “link source”. There are two types of references: odxlinks, SHORT-NAME references and short-name-path references.

The following descriptions are based on the implementation of the data model in XML.

7.3.13.2 References via odxlink

As a convention, odxlink elements are named by appending the suffix “–REF” to the name of the referenced element type, e.g. a UNIT-REF links to a UNIT. Link elements normally have no sub-elements and no text as content. The attributes DOCTYPE, DOCREF, ID-REF and REVISION are used to specify the link target.

ODX data may be assembled by composing so-called “document fragments”. A document fragment represents a logical part of the ODX document. An odxlink usually references the link-target via the document fragment the link-target is located in. The document fragment is defined by the two attributes DOCTYPE and DOCREF. See Figure 125 — UML representation of structure of an odxlink for further detail.

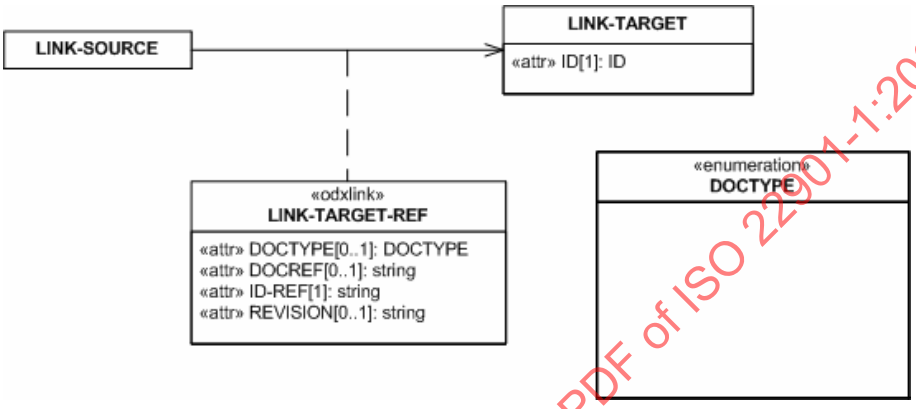


Figure 125 — UML representation of structure of an odxlink

Generally speaking, odxlinks are resolved in two steps:

a) find the document fragment that contains the link target;

b) retrieve the link target from the document fragment.

Step a) uses the attributes DOCTYPE and DOCREF. These two attributes may be defined only jointly. The DOCREF attribute refers to the SHORT-NAME of the ODX-CATEGORY or the DIAG-LAYER. If DOCREF and DOCTYPE are not set, the link target exists in the local ODX-CATEGORY where also the link source exists (internal odxlink).

In step b), the ID of the referenced element is specified by the ID-REF attribute. The optional REVISION attribute can be used to link to a specific version of the document fragment. An ODX conformant tool may ignore this attribute while resolving odxlinks.

The examples below show some use cases for odxlinks.

EXAMPLE 1 Linking a PROTOCOL layer.

Container-based linking: Figure 126 — Linking a PROTOCOL layer (via DIAG-LAYER-CONTAINER) describes container-based linking.

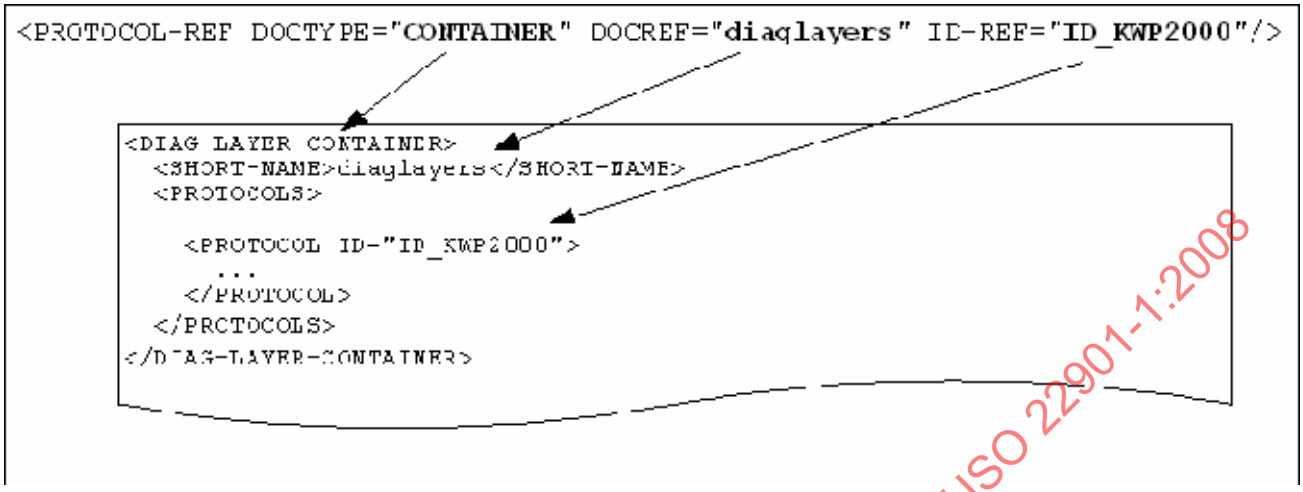


Figure 126 — Linking a PROTOCOL layer (via DIAG-LAYER-CONTAINER)

Layer-based linking: DOCREF refers to the SHORT-NAME of the document fragment while ID-REF refers to the ID. Figure 127 — Linking a PROTOCOL layer (via DIAG-LAYER) describes layer-based linking via DIAG-LAYER.

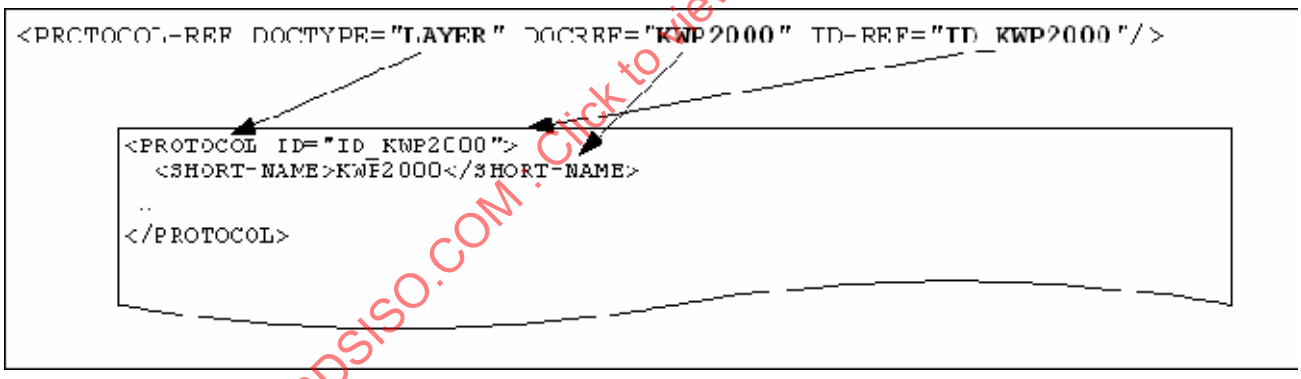


Figure 127 — Linking a PROTOCOL layer (via DIAG-LAYER)

EXAMPLE 2 Linking a UNIT.

Container-based linking: Figure 128 — Linking a UNIT (via DIAG-LAYER-CONTAINER) describes container-based linking via DIAG-LAYER-CONTAINER.

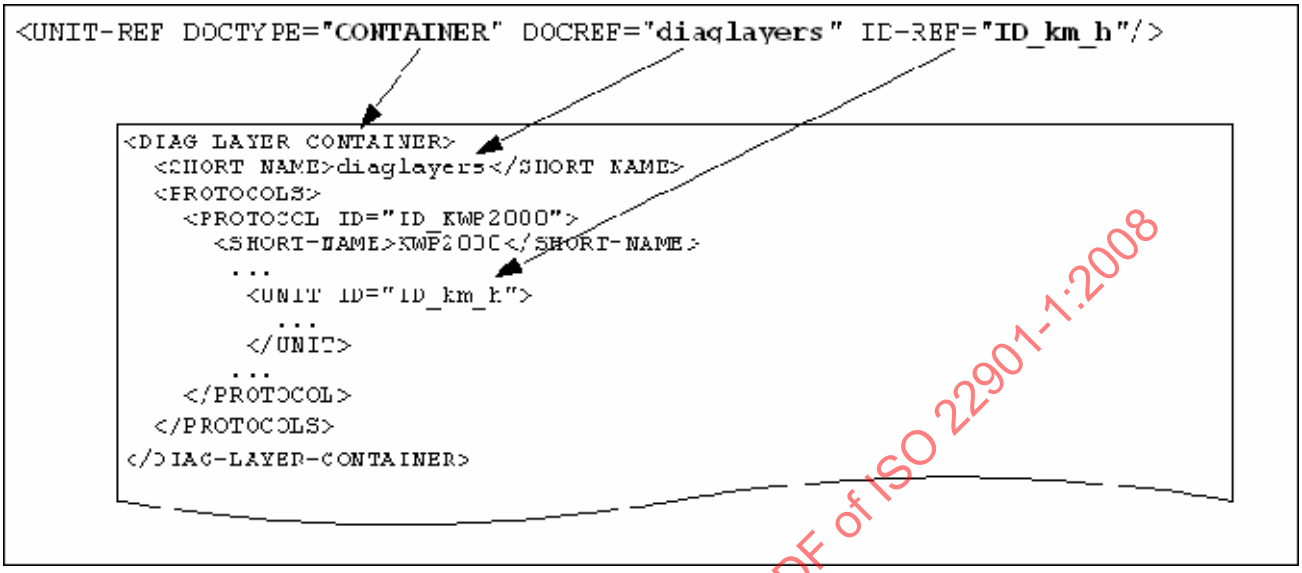


Figure 128 — Linking a UNIT (via DIAG-LAYER-CONTAINER)

Layer-based linking: Figure 129 — Linking a UNIT (via DIAG-LAYER) describes layer-based linking via DIAG-LAYER.

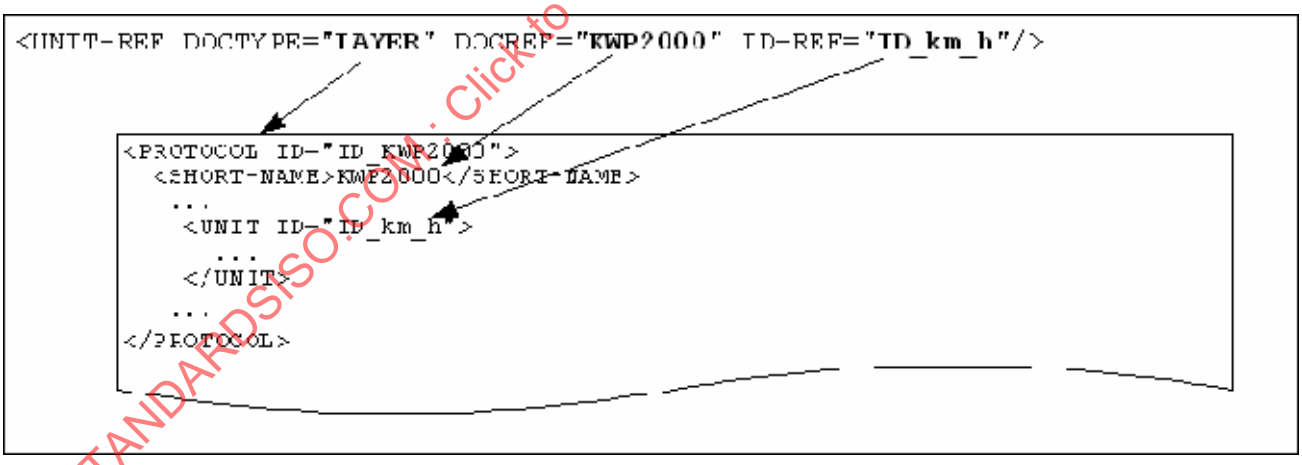


Figure 129 — Linking a UNIT (via DIAG-LAYER)

EXAMPLE 3 Internal versus external linking:

```
<DIAG-LAYER-CONTAINER>
  <SHORT-NAME>diaglayers</SHORT-NAME>
  <PROTOCOLS>
    <PROTOCOL ID="ISO14230">
      <SHORT-NAME>ISO14230</SHORT-NAME>
      ...
      <UNIT ID="ID_km_h">...</UNIT>
      ...
      <UNIT-REF DOCTYPE="CONTAINER" DOCREF="diaglayers" ID-REF="ID_km_h"/> (1)
      <UNIT-REF DOCTYPE="LAYER" DOCREF="ISO14230" ID-REF="ID_km_h"/>       (2)
      <UNIT-REF ID-REF="ID_km_h"/>                                         (3)
      ...
    </PROTOCOL>
  <PROTOCOLS>
  <ECU-SHARED-DATAS>
    <ECU-SHARED-DATA>
      ...
      <UNIT-REF DOCTYPE="CONTAINER" DOCREF="diaglayers" ID-REF="ID_km_h"/>  (4)
      <UNIT-REF ID-REF="ID_km_h"/>                                          (5)
            ...
    </ECU-SHARED-DATA>
  </ECU-SHARED-DATAS>
</DIAG-LAYER-CONTAINER>

<DIAG-LAYER-CONTAINER>
  <SHORT-NAME>someOtherContainer</SHORT-NAME>
  <ECU-SHARED-DATAS>
    <ECU-SHARED-DATA>
      ...
      <UNIT-REF DOCTYPE="LAYER" DOCREF="ISO14230" ID-REF="ID_km_h"/>        (6)
      <UNIT-REF DOCTYPE="CONTAINER" DOCREF="diaglayers" ID-REF="ID_km_h"/>  (7)
      ...
    </ECU-SHARED-DATA>
  </ECU-SHARED-DATA>
</DIAG-LAYER-CONTAINER>
```

The links **(1)** to **(5)** are internal links.

The links **(6)** and **(7)** are external links.

7.3.13.3 References via short name path

Parameters form a recursive hierarchy. Parameters that reference COMPLEX-DOPs contain themselves parameters that again can have a complex inner structure. Short-name-paths usually reference single parameters of complex or simple structure within this hierarchy. They may reference multiple parameters with the same name only when they are used for matching a parameter value. There, a FIELD parameter is handled in a special way.

Figure 130 — Structure of a short-name-path reference describes further detail.

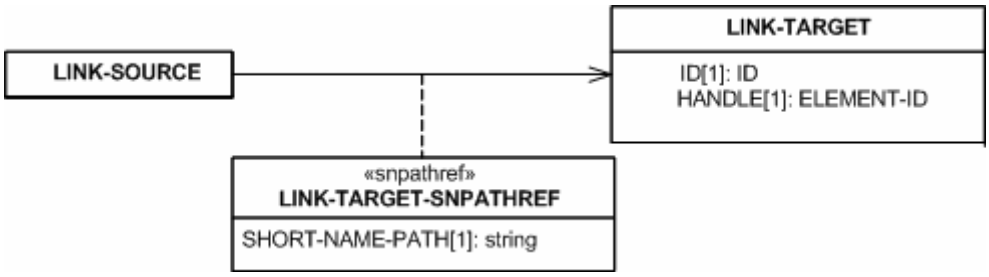


Figure 130 — Structure of a short-name-path reference

As a convention, short-name-path-references are named by appending the suffix "–SNPATHREF" to the name of the referenced element type, e.g. OUT-PARAM-IF-SNPATHREF links to a output parameter (OUT-PARAM-IF). The path itself is stored in the XML-attribute SHORT-NAME-PATH as depicted above.

All short-name-paths target parameters. In addition to the short-name-path a reference to a DIAG-COMM or MULTIPLE-ECU-JOB is always given. The initial context in which a short-name-path is to be resolved is defined along with the source object of the «snref-or-snpathref». For example, a MATCHING-PARAMETER element references a DIAG-COMM via «snref». The parameter referenced by the MATCHING-PARAMETER is searched in the PARAMeters of the concrete response received for that DIAG-COMM or in its OUTPUT-PARAMs if the DIAG-COMM is a SINGLE-ECU-JOB.

The path itself adheres to the following syntax:

```
path         =  step { stepsequence }  ;
step         =  SHORT-NAME             ;
stepsequence =  separator step         ;
separator    =  "."                    ;
```

Each step is resolved in order, first to last. There are two algorithms for resolving a short-name-path: The *Single-Param-Algorithm* and the *Multiple-Param-Algorithm*. The first always selects a single parameter, or signals an error. The second may also select a set of parameters. In the following subclauses both algorithms are described in detail. For the second one only its difference to the first is explained. Again, which algorithm to use is defined along with the source object of the «snref-or-snpathref».

When resolving a short-name-path the value of the IS-VISIBLE attribute is ignored. Therefore, all parameters are visible to the resolution algorithm.

EXAMPLE Referencing a response PARAM via short-name-path:

```
<OUT-PARAM-IF-SNPATHREF SHORT-NAME-PATH="sn_start.sn_minute.sn_display"/>
```

7.3.13.3.1 Single-Param-Algorithm

The algorithm works on runtime-objects that need not have a unique representation in the ODX data model. Figure 131 shows an example of such a response. The response consists of two complex parameters short-named "sn_start" and "sn_end". They each, in turn, consist of three parameters short-named "sn_hour", "sn_minute", and "sn_second". Of these, the first two are again structured and each contains parameters short-named "sn_value" and "sn_display". The structure of these parameters is described in ODX. But, while

the actual response contains 4 parameters named “sn_value”, in the ODX-document only a single parameter with that SHORT-NAME can be found.

EXAMPLE ODX description of a response and its structures:

```
<POS-RESPONSE ID="id_getSpanResp">
  <SHORT-NAME>sn_getSpan</SHORT-NAME>
  <PARAMS>
    <PARAM xsi:type="VALUE">
       <SHORT-NAME>sn_start</SHORT-NAME>
       <DOP-SNREF SHORT-NAME="sn_time"/>
    </PARAM>
    <PARAM xsi:type="VALUE">
       <SHORT-NAME>sn_end</SHORT-NAME>
       <DOP-SNREF SHORT-NAME="sn_time"/>
    </PARAM>
  </PARAMS>
</POS-RESPONSE>

<STRUCTURE ID="id_display">
  <SHORT-NAME>sn_display</SHORT-NAME>
  <BYTE-SIZE>2</BYTE-SIZE>
  <PARAMS>
     <PARAM xsi:type="VALUE">
        <SHORT-NAME>sn_value</SHORT-NAME>
        <BYTE-POSITION>0</BYTE-POSITION>
        <DOP-REF ID-REF="id_byte"/>
     </PARAM>
     <PARAM xsi:type="VALUE">
        <SHORT-NAME>sn_display</SHORT-NAME>
        <BYTE-POSITION>1</BYTE-POSITION>
        <DOP-REF ID-REF="id_byte"/>
     </PARAM>
  </PARAMS>
</STRUCTURE>
```

```
<STRUCTURE ID="id_time">
  <SHORT-NAME>sn_time</SHORT-NAME>
  <BYTE-SIZE>5</BYTE-SIZE>
  <PARAMS>
     <PARAM xsi:type="VALUE">
        <SHORT-NAME>sn_hour</SHORT-NAME>
        <BYTE-POSITION>0</BYTE-POSITION>
        <DOP-REF ID-REF="id_display"/>
     </PARAM>
     <PARAM xsi:type="VALUE">
        <SHORT-NAME>sn_minute</SHORT-NAME>
        <BYTE-POSITION>2</BYTE-POSITION>
        <DOP-REF ID-REF="id_display"/>
     </PARAM>
     <PARAM  xsi:type="VALUE">
        <SHORT-NAME>sn_second</SHORT-NAME>
        <BYTE-POSITION>4</BYTE-POSITION>
        <DOP-REF ID-REF="id_byte"/>
     </PARAM>
  </PARAMS>
</STRUCTURE>
```


Figure 131 — Response example for Single-Param-Algorithm illustrates the above description.

sn_start					sn_end				
sn_hour		sn_minute		sn_second	sn_hour		sn_minute		sn_second
sn_value	sn_display	sn_value	sn_display		sn_value	sn_display	sn_value	sn_display	
7	19	7	7	54	8	20	34	34	12

Figure 131 — Response example for Single-Param-Algorithm

As an example for the algorithm, take the above path “sn_start.sn_minute.sn_display”.

a) The first step shall be resolved given the set of parameters determined by the source object of the short-name-path. This set of elements is defined for each source object type separately throughout this part of ISO 22901. Each of these elements always carries an unique SHORT-NAME. This is guaranteed by the

element set definition and the SHORT-NAME boundaries defined in 7.3.13.5. The first step selects the element from the given set that has the same SHORT-NAME value as the step. In the example the set contains the PARAM elements with SHORT-NAME “sn_start” and “sn_end”. The first step is “sn_start”, thus the first parameter spanning the first 5 bytes is selected.

b) Resolve the next step, if any, otherwise stop. Every step but the first shall be resolved from the currently selected parameter and the value of the current step. For the second step in the example, the currently selected parameter short-named “sn_start” is currently selected and the value of the current step is “sn_minute”. The resolution of a step depends on the type of parameter according to Table 15 — Resolution algorithm for Single-Param-Algorithm.

Table 15 — Resolution algorithm for Single-Param-Algorithm

Type of parameter		Resolution algorithm
CODED-CONST		signal error
DYNAMIC		signal error
LENGTH-KEY		signal error
MATCHING-REQUEST-PARAM		signal error
NRC-CONST		signal error
PHYS-CONST		signal error
RESERVED		signal error
SYSTEM		signal error
TABLE-ENTRY		The TABLE-ENTRY references a TABLE-ROW. Resolution depends on the value of ROW-FRAGMENT.
	ROW-FRAGMENT	
	KEY	signal error
	STRUCT	If the TABLE-ROW references a DATA-OBJECT-PROP signal an error. If the TABLE-ROW references a STRUCTURE, that STRUCTURE contains an ordered set of PARAM objects. From this set select the PARAM with the same SHORT-NAME as the step.
TABLE-KEY		signal error
TABLE-STRUCT		The TABLE-STRUCT is linked to a TABLE-KEY parameter. The value of this TABLE-KEY parameter shall determine a TABLE-ROW as described in 7.3.6.11 (for the dynamic case, its value shall have been set), otherwise signal an error. If the TABLE-ROW references a DATA-OBJECT-PROP signal an error. If the TABLE-ROW references a STRUCTURE, that STRUCTURE contains an ordered set of PARAM objects. From this set select the PARAM with the same SHORT-NAME as the step.
VALUE		The VALUE references a DOP-BASE. Resolution depends on the type of the DOP-BASE object:
	DOP	
	DATA-OBJECT-PROP	signal error

Table 15 (*continued*)

Type of parameter		Resolution algorithm
	DTC-DOP	signal error
	MUX	The MUX references a STRUCTURE[19]. That STRUCTURE contains an ordered set of PARAM objects. From this set select the PARAM with the same SHORT-NAME as the step.
	BASIC-STRUCTURE	The BASIC-STRUCTURE contains an ordered set of PARAM objects. From this set select the PARAM with the same SHORT-NAME as the step.
	ENV-DATA-DESC	The ENV-DATA-DESC references an ENV-DATA object.[20] The ENV-DATA contains an ordered set of PARAM objects. From this set select the PARAM with the same SHORT-NAME as the step.
	FIELD	signal error

If no error has been signalled, the currently selected parameter has changed. With that changed status, repeat step b).

The target of the short-name-path is the currently selected element.

In the rare case, that a short-name-path shall be resolved on data base objects within D-Server (for example within a SUB-COMPONENT element), the runtime object that is targeted by the short-name-path will correspond to the returned data base object. While on the runtime side a short-name-path references a single parameter, this need not be the case for the data base side. For example, in the structure depicted in Figure 131 the path “sn_start.sn_minute.sn_display” will be resolved on the data base side to the definition of the parameter with the name “sn_display”. Yet, that data base object might be used by the run time parameters referenced by “sn_start.sn_hour.sn_display”, “sn_end.sn_hour.sn_display”, and “sn_end.sn_minute.sn_display” as well. On the data base side, these four parameters might not be distinguished.

7.3.13.3.2 Multiple-Param-Algorithm

The Multiple-Param-Algorithm handles FIELDs differently than the Single-Param-Algorithm. It may select a single parameter, a set of parameters that all have the same SHORT-NAME, or signal an error.

As an example consider the following response: The parameter short-named “sn_log” is a FIELD PARAMeter. Its repeatedElement is a basic structure that, in turn, consists of a PARAMeter short-named “sn_result” and a PARAMeter “sn_second”. In the example response the repeatedElement is repeated thrice. The PARAMeter short-named “sn_result” is complex and consist of two PARAMeters short-named “sn_value” and “sn_display”. As an example consider the path “sn_log.sn_result.sn_value”. It will select all parameters short-named “sn_value” within the FIELD short-named “sn_log”.

EXAMPLE ODX description of a response and its structures:

```
<POS-RESPONSE ID="id_getLog">
  <SHORT-NAME>sn_getLog</SHORT-NAME>
  <PARAMS>
    <PARAM xsi:type="VALUE">
       <SHORT-NAME>sn_log</SHORT-NAME>
       <DOP-SNREF SHORT-NAME="sn_logField"/>
    </PARAM>
```

```
<STRUCTURE ID="id_logEntry">
  <SHORT-NAME>sn_display</SHORT-NAME>
  <BYTE-SIZE>2</BYTE-SIZE>
  <PARAMS>
     <PARAM xsi:type="VALUE">
        <SHORT-NAME>sn_result</SHORT-NAME>
        <BYTE-POSITION>0</BYTE-POSITION>
```

19) A MUX shall only occur in a RESPONSE. Here, the MUX shall select a STRUCTURE with the actual data. Otherwise, the RESPONSE as a whole would already be in error.

20) An ENV-DATA-DESC shall only occur in a RESPONSE. Again, the current data shall already select one ENV-DATA object, or the RESPONSE as a whole would be in error.

```
  </PARAMS>
</POS-RESPONSE>

<END-OF-PDU-FIELD ID="id_logField">
        <SHORT-NAME>sn_logField</SHORT-NAME>
        <BASIC-STRUCTURE-REF ID-REF = "id_logEntry"/>
        <MAX-NUMBER-OF-ITEMS>6</MAX-NUMBER-OF-ITEMS>
        <MIN-NUMBER-OF-ITEMS>1</MIN-NUMBER-OF-ITEMS>
</END-OF-PDU-FIELD>
```

```
      <DOP-REF ID-REF="id_display"/>
    </PARAM>
    <PARAM xsi:type="VALUE">
      <SHORT-NAME>sn_second</SHORT-NAME>
      <BYTE-POSITION>1</BYTE-POSITION>
      <DOP-REF ID-REF="id_byte"/>
    </PARAM>
  </PARAMS>
</STRUCTURE>
```

See Figure 132 — Response example for Multiple-Param-Algorithm for further detail regarding above example.

sn_log								
sn_result		sn_second	sn_result		sn_second	sn_result		sn_second
sn_value	sn_display		sn_value	sn_display		sn_value	sn_display	
7	66	19	20	54	22	20	27	27

Figure 132 — Response example for Multiple-Param-Algorithm

EXAMPLE Referencing a response PARAM via short-name-path:

```
<OUT-PARAM-IF-SNPATHREF SHORT-NAME-PATH="sn_log.sn_result.sn_value"/>
```

As an example, consider the path “sn_log.sn_result.sn_value”. It will select all parameters short-named “sn_value” within the FIELD short-named “sn_log”.

a) The first step is resolved exactly as the in the Single-Param-Algorithm. It selects a single element. In the example, the PARAMeter short-named “sn_log” is selected.

b) Resolve the next step, if any, otherwise stop. Input to the next step is a *set* of selected PARAMeters. The set may also contain only a single element. Resolution is performed for all parameters in the set simultaneously. The resulting selections are united into a single set, resulting in either a new set for the next step or in an error if resolving the step for any element signalled an error.

For each element in the set, resolution is done as in the Single-Param-Algorithm with one exception: if the DOP of a VALUE PARAMeter is FIELD.

Table 16 — Resolution algorithm for Multiple-Param-Algorithm

Type of parameter		Resolution algorithm
VALUE		The VALUE references a DOP-BASE. Resolution depends on the type of the DOP-BASE object:
	DOP	
	FIELD	The repeatedElement of the FIELD is a BASIC-STRUCTURE. It occurs repeatedly in the PDU. That BASIC-STRUCTURE contains an ordered set of PARAM objects. From this repeated set select all repeated PARAMs with the same SHORT-NAME as the step.

Thus, in the example, the second step will select all three PARAMeters short-named “sn_result”. They will be used as input for the third and last step. The third step will, for each of these PARAMeters, select the PARAMeter short-named “sn_value” and unite them into a result set.

Repeat step b).

The target of the short-name-path is the currently selected set of PARAMeters.

7.3.13.4 References via SHORT-NAME

Some elements of an ODX document like DIAG-SERVICE can be referenced via SHORT-NAME. Parameters can for example reference a DOP via odxlink or SHORT-NAME (stereotype in the UML drawings «odxlink-or-snref»). In case of SHORT-NAME references the value inheritance and overriding mechanism shall be observed (see 7.3.2.4). See Figure 133 — Structure of a SNREF link for further detail.

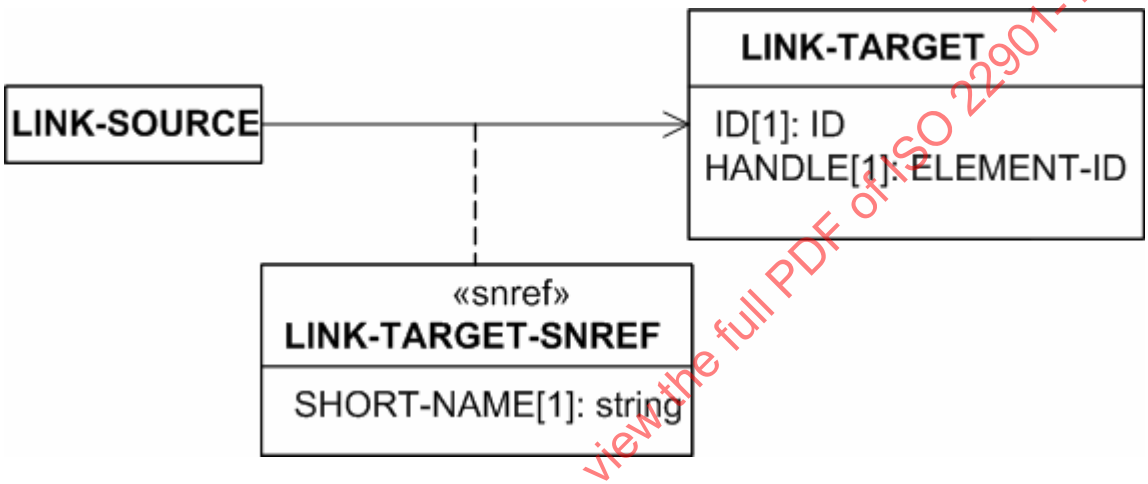


Figure 133 — Structure of a SNREF link

As a convention, SHORT-NAME references are named by appending the suffix “–SNREF” to the name of the referenced element type, e.g. PARAM-SNREF links to a parameter (PARAM).

EXAMPLE 1 Referencing a response PARAM via SHORT-NAME:

```
<OUT-PARAM-IF-SNREF SHORT-NAME="ReadDataByAddress"/>
```

EXAMPLE 2 Referencing a DIAG-SERVICE via SHORT-NAME:

```
<DIAG-COMM-SNREF SHORT-NAME="DS_ReadDataByAddress"/>
```

7.3.13.5 Boundary for uniqueness of SHORT-NAME

According to the reference mechanism via SHORT-NAME and the value inheritance mechanism adequate boundaries shall be defined for unique SHORT-NAMEs. Table 17 — shows the boundary definitions of the short names at several positions. The term “global” means unique within one ODX database. It may be the scope of ODX data of a particular company as a whole. A group of elements in one table row means that all the short names of these elements together should be unique in the given boundary. Table 17 defines the SHORT-NAME namespaces on the XML elements generated from the UML model and not for all UML elements. Therefore, for example, child classes that inherit from a parent class with an <<impl-parent>> stereotype do not appear within this table.

EXAMPLE SESSION and DATABLOCK are necessarily unique within ECU-MEM, but it is permitted that a SESSION has the same SHORT-NAME value as a DATABLOCK because they belong to different name spaces. Would they share the same name space a SESSION and a DATABLOCK with equal SHORT-NAME shall not exist within one ECU-MEM.

Table 17 — Boundary for uniqueness of SHORT-NAME

Name Space (Base Class)	Elements within Name Space	Boundary
ADDITIONAL-AUDIENCE	ADDITIONAL-AUDIENCE	DIAG-LAYER, MULTIPLE-ECU-JOB-SPEC, FLASH, ECU-CONFIG, FUNCTION-DICITIONARY
BASE-COMPARAM	COMPARAM, COMPLEX-COMPARAM	PROT-STACK
BASE-FUNCTION-NODE	FUNCTION-NODE, FUNCTION-NODE-GROUP	FUNCTION-DICTIONARY
CASE	CASE, DEFAULT-CASE	MUX
CHECKSUM	CHECKSUM	SESSION
COMPANY-DATA	COMPANY-DATA	ODX-CATEGORY DIAG-LAYER
CONFIG-DATA	CONFIG-DATA	ECU-CONFIG
CONFIG-ITEM	CONFIG-ID-ITEM, DATA-ID-ITEM, OPTION-ITEM, SYSTEM-ITEM	CONFIG-RECORD
CONFIG-RECORD	CONFIG-RECORD	CONFIG-DATA
DATA-RECORD	DATA-RECORD	CONFIG-RECORD
DATABLOCK	DATABLOCK	ECU-MEM
DIAG-COMM	DIAG-SERVICE, SINGLE-ECU-JOB	DIAG-LAYER
DIAG-LAYER	PROTOCOL, FUNCTIONAL-GROUP, BASE-VARIANT, ECU-VARIANT, ECU-SHARED-DATA	Global
DIAG-OBJECT-CONNECTOR	DIAG-OBJECT-CONNECTOR	COMPONENT-CONNECTOR
DIAG-VARIABLE	DIAG-VARIABLE	DIAG-LAYER
DOP-BASE	DATA-OBJECT-PROP, DTC-DOP, MUX, ENV-DATA, STRUCTURE, ENV-DATA-DESC, STATIC-FIELD, DYNAMIC-ENDMARKER-FIELD, DYNAMIC-LENGTH-FIELD, END-OF-PDU-FIELD	DIAG-DATA-DICTIONARY-SPEC, CONFIG-DATA-DICTIONARY-SPEC, COMPARAM-SUBSET
DTC	DTC	DTC-DOP
DTC-CONNECTOR	DTC-CONNECTOR	SUB-COMPONENT, COMPONENT-CONNECTOR
ENV-DATA-CONNECTOR	ENV-DATA-CONNECTOR	SUB-COMPONENT, COMPONENT-CONNECTOR
ECU-GROUP	ECU-GROUP	VEHICLE-INFORMATION
ECU-MEM-CONNECTOR	ECU-MEM-CONNECTOR	FLASH
ECU-MEM	ECU-MEM	FLASH
EXPECTED-IDENT	EXPECTED-IDENT	SESSION
EXTERNAL-ACCESS-METHOD	EXTERNAL-ACCESS-METHOD	STATE-TRANSITION
FLASH-CLASS	FLASH-CLASS	ECU-MEM-CONNECTOR
FLASHDATA	EXTERN-FLASHDATA, INTERN-FLASHDATA	ECU-MEM
FUNCT-CLASS	FUNCT-CLASS	DIAG-LAYER, MULTIPLE-ECU-JOB-SPEC
	FUNCTION-OUT-PARAM, FUNCTION-IN-PARAM	BASE-FUNCTION-NODE

Table 17 (*continued*)

Name Space (Base Class)	Elements within Name Space	Boundary
INFO-COMPONENT	ECU-PROXY, MODEL-YEAR, OEM, VEHICLE-MODEL, VEHICLE-TYPE	VEHICLE-INFO-SPEC
LIBRARY	LIBRARY	DIAG-LAYER
LOGICAL-LINK	GATEWAY-LOGICAL-LINK, MEMBER-LOGICAL-LINK	VEHICLE-INFORMATION
MULTIPLE-ECU-JOB	MULTIPLE-ECU-JOB	Global
ODX-CATEGORY	COMPARAM-SPEC, MULTIPLE-ECU-JOB-SPEC, VEHICLE-INFO-SPEC, FLASH, DIAG-LAYER-CONTAINER, ECU-CONFIG, COMPARAM-SUBSET, FUNCTION-DICTIONARY	Global
OWN-IDENT	OWN-IDENT	DATABLOCK, SESSION-DESC
PARAM	CODED-CONST, DYNAMIC, LENGTH-KEY, MATCHING-REQUEST-PARAM, NRC-CONST,PHYS-CONST, RESERVED, SYSTEM, TABLE-ENTRY, TABLE-KEY, TABLE-STRUCT, VALUE, INPUT-PARAM, OUTPUT-PARAM, NEG-OUTPUT-PARAM	BASIC-STRUCTURE, REQUEST[21], RESPONSE, SINGLE-ECU-JOB, MULTIPLE-ECU-JOB
PHYSICAL-DIMENSION	PHYSICAL-DIMENSION	UNIT-SPEC
PHYSICAL-VEHICLE-LINK	PHYSICAL-VEHICLE-LINK	VEHICLE-INFORMATION
PHYS-MEM	PHYS-MEM	ECU-MEM
PHYS-SEGMENT	ADDRDEF-PHYS-SEGMENT, SIZEDEF-PHYS-SEGMENT	PHYS-MEM
PROT-STACK	PROT-STACK	COMPARAM-SPEC
REQUEST	REQUEST	DIAG-LAYER
RESPONSE	NEG-RESPONSE, POS-RESPONSE, GLOBAL-NEG-RESPONSE	DIAG-LAYER
SDG-CAPTION	SDG-CAPTION	SDG
SEGMENT	SEGMENT	DATABLOCK
SESSION	SESSION	ECU-MEM
SESSION-DESC	SESSION-DESC	ECU-MEM-CONNECTOR
STATE	STATE	STATE-CHART
STATE-CHART	STATE-CHART	DIAG-LAYER
STATE-TRANSITION	STATE-TRANSITION	STATE-CHART
SUB-COMPONENT	SUB-COMPONENT	DIAG-LAYER
SUB-COMPONENT-PARAM-CONNECTOR	SUB-COMPONENT-PARAM-CONNECTOR	SUB-COMPONENT
SW-VARIABLE	SW-VARIABLE	SW-VARIABLE
TABLE	TABLE	DIAG-DATA-DICTIONARY-SPEC
TABLE-ROW	TABLE-ROW	TABLE

21) This means the PARAM elements contained *directly* in this element. It does not include all PARAM elements reachable from included structures, fields and so on.

Table 17 (*continued*)

Name Space (Base Class)	Elements within Name Space	Boundary
TABLE-ROW-CONNECTOR	TABLE-ROW-CONNECTOR	SUB-COMPONENT, COMPONENT-CONNECTOR
TEAM-MEMBER	TEAM-MEMBER	COMPANY-DATA
UNIT-GROUP	UNIT-GROUP	UNIT-SPEC
UNIT	UNIT	UNIT-SPEC
VARIABLE-GROUP	VARIABLE-GROUP	DIAG-LAYER
VEHICLE-CONNECTOR	VEHICLE-CONNECTOR	VEHICLE-INFORMATION
VEHICLE-CONNECTOR-PIN	VEHICLE-CONNECTOR-PIN	VEHICLE-CONNECTOR
VEHICLE-INFORMATION	VEHICLE-INFORMATION	VEHICLE-INFO-SPEC
XDOC	XDOC	RELATED-DOC

7.4 Usage scenarios (diagnostic)

7.4.1 Diagnostic service description

As an example for this usage scenario the service InputOutputControlByLocalIdentifier from ISO 14230 is used. The request of this service has the following structure as shown in Figure 134 — Request of service 0x30 in ISO 14230:

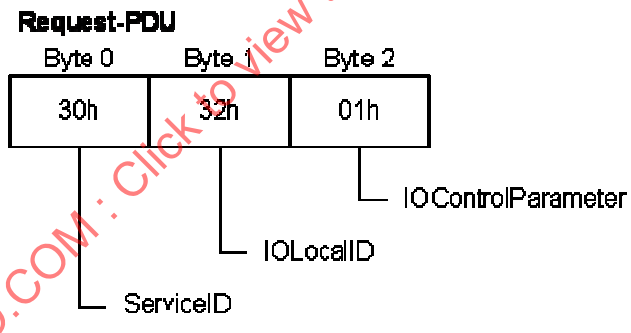


Figure 134 — Request of service 0x30 in ISO 14230

There are three parameters in the request:

— ServiceID (0x30 = InputOutputControlByLocalIdentifier);

— IOLocalID (0x32 = “Desired Idle Adjustment”);

— IOControlParameter (0x01 = reportCurrentState).

As a response, the following PDU is possible as shown in Figure 135 — Response of service 0x30 in ISO 14230.

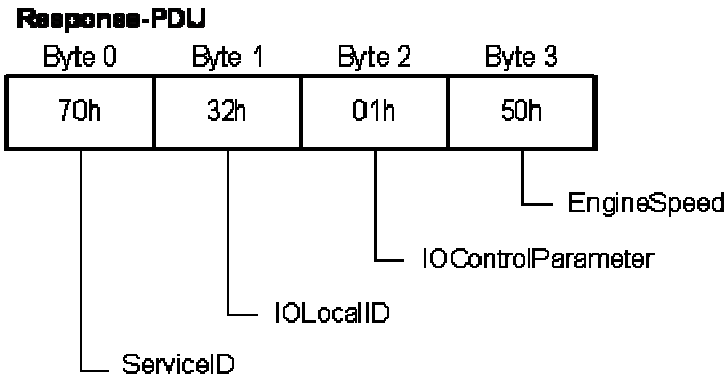


Figure 135 — Response of service 0x30 in ISO 14230

The first parameter of the response is the Service Id + 0x40 = 0x70. The next two ones are the replication of the request parameters at the same positions. The last byte indicates the coded value of the current engine speed. The physical value is calculated by multiplying the coded one with 10. Thus the engine speed is equal 800 r/min (0x50 = 80; 80 * 10 = 800).

The diagnostic service is defined in ODX as follows:

```
<DIAG-SERVICE ID = "DS_IOCBLID" SEMANTIC = "CONTROL">
  <SHORT-NAME>DS_IOCBLID</SHORT-NAME>
  <AUDIENCE/>
  <REQUEST-REF ID-REF = "REQ_IOCBLID"/>
  <POS-RESPONSE-REFS>
    <POS-RESPONSE-REF ID-REF = "RESP_IOCBLID"/>
  </POS-RESPONSE-REFS>
</DIAG-SERVICE>
```

whereby the request “REQ_IOCBLID” and the response “RESP_IOCBLID” look like this:

```
<REQUEST ID = "REQ_IOCBLID">
  <SHORT-NAME>REQ_IOCBLID</SHORT-NAME>
  <PARAMS>
    <PARAM SEMANTIC = "SERVICE-ID" xsi:type = "CODED-CONST">
      <SHORT-NAME>ServiceID</SHORT-NAME>
      <BYTE-POSITION>0</BYTE-POSITION>
      <CODED-VALUE>48</CODED-VALUE>
      <DIAG-CODED-TYPE xsi:type = "STANDARD-LENGTH-TYPE" BASE-DATA-TYPE = "A_UINT32">
         <BIT-LENGTH>8</BIT-LENGTH>
       </DIAG-CODED-TYPE>
     </PARAM>
     <PARAM xsi:type = "CODED-CONST">
      <SHORT-NAME>IOLocalID</SHORT-NAME>
       <BYTE-POSITION>1</BYTE-POSITION>
       <CODED-VALUE>50</CODED-VALUE>
       <DIAG-CODED-TYPE xsi:type = "STANDARD-LENGTH-TYPE" BASE-DATA-TYPE = "A_UINT32">
          <BIT-LENGTH>8</BIT-LENGTH>
       </DIAG-CODED-TYPE>
    </PARAM>
    <PARAM xsi:type = "CODED-CONST">
      <SHORT-NAME>IOControlParameter</SHORT-NAME>
```

```
      <BYTE-POSITION>2</BYTE-POSITION>
      <CODED-VALUE>01</CODED-VALUE>
      <DIAG-CODED-TYPE xsi:type = "STANDARD-LENGTH-TYPE" BASE-DATA-TYPE = "A_UINT32">
        <BIT-LENGTH>8</BIT-LENGTH>
      </DIAG-CODED-TYPE>
    </PARAM>
  </PARAMS>
</REQUEST>

<POS-RESPONSE ID = "RESP_IOCBLID">
  <SHORT-NAME>RESP_IOCBLID</SHORT-NAME>
  <PARAMS>
    <PARAM xsi:type = "CODED-CONST" SEMANTIC = "SERVICE-ID">
      <SHORT-NAME>ServiceID</SHORT-NAME>
      <BYTE-POSITION>0</BYTE-POSITION>
      <CODED-VALUE>112</CODED-VALUE>
      <DIAG-CODED-TYPE xsi:type = "STANDARD-LENGTH-TYPE" BASE-DATA-TYPE = "A_UINT32">
        <BIT-LENGTH>8</BIT-LENGTH>
      </DIAG-CODED-TYPE>
    </PARAM>
    <PARAM xsi:type = "MATCHING-REQUEST-PARAM">
      <SHORT-NAME>IOLocalID</SHORT-NAME>
      <BYTE-POSITION>1</BYTE-POSITION>
      <REQUEST-BYTE-POS>1</REQUEST-BYTE-POS>
       <BYTE-LENGTH>1</BYTE-LENGTH>
    </PARAM>
    <PARAM xsi:type = "MATCHING-REQUEST-PARAM">
      <SHORT-NAME>IOControlParameter</SHORT-NAME>
      <BYTE-POSITION>2</BYTE-POSITION>
      <REQUEST-BYTE-POS>2</REQUEST-BYTE-POS>
      <BYTE-LENGTH>1</BYTE-LENGTH>
    </PARAM>
    <PARAM xsi:type = "VALUE">
      <SHORT-NAME>EngineSpeed</SHORT-NAME>
      <BYTE-POSITION>3</BYTE-POSITION>
      <DOP-REF ID-REF = "DOP_EngineSpeed"/>
    </PARAM>
  </PARAMS>
</POS-RESPONSE>
```

In the response, the second and the third parameters are declared as “MATCHING-REQUEST-PARAM” parameters. Therefore, they refer to the appropriate byte position in the request by the element REQUEST-BYTE-POS. The fourth parameter is the proper result of the request. It contains a link to a DOP, which is responsible for conversion.

This DOP and the associated UNIT are defined as follows:

```
<DATA-OBJECT-PROP ID = "DOP_EngineSpeed">
  <SHORT-NAME>DOP_EngineSpeed</SHORT-NAME>
  <COMPU-METHOD>
    <CATEGORY>LINEAR</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
      <COMPU-SCALES>
        <COMPU-SCALE>
          <COMPU-RATIONAL-COEFFS>
              <COMPU-NUMERATOR>
              <V>0</V>
```

```
              <V>10</V>
            </COMPU-NUMERATOR>
            <COMPU-DENOMINATOR>
              <V>1</V>
            </COMPU-DENOMINATOR>
          </COMPU-RATIONAL-COEFFS>
        </COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
  </COMPU-METHOD>
  <DIAG-CODED-TYPE BASE-DATA-TYPE = "A_UINT32" xsi:type = "STANDARD-LENGTH-TYPE">
    <BIT-LENGTH>8</BIT-LENGTH>
  </DIAG-CODED-TYPE>
  <PHYSICAL-TYPE BASE-DATA-TYPE = "A_UINT32" DISPLAY-RADIX = "DEC"/>
  <UNIT-REF ID-REF = "Unit_RPerMin"/>
</DATA-OBJECT-PROP>


<UNIT ID = "Unit_RPerMin">
  <SHORT-NAME>Unit_RPerMin</SHORT-NAME>
  <DISPLAY-NAME>r/min</DISPLAY-NAME>
  <FACTOR-SI-TO-UNIT>60</FACTOR-SI-TO-UNIT>
  <PHYSICAL-DIMENSION-REF ID-REF = "PD_RPerSec"/>
</UNIT>


<PHYSICAL-DIMENSION ID = "PD_RPerSec">
  <SHORT-NAME>PD_RPerSec</SHORT-NAME>
  <TIME-EXP>-1</TIME-EXP>
</PHYSICAL-DIMENSION>
```

The DOP's COMPU-METHOD is of type “LINEAR” which is indicated by the element CATEGORY. The numerator is defined by two V elements (<V>0</V><V>10</V>).

This leads to the following equation:

PhysicalValue = 0 + CodedValue * 10

The referenced UNIT object provides the units of the computed value (r/min). It is derived from the physical dimension “PHYS-DIM-rPerSec” with the time exponent value of -1. As the SI unit for time is second a factor of 60 is defined to get from rounds per second to rounds per minute.

7.4.2 Dynamically defined messages

7.4.2.1 General

With some protocols (e.g. ISO 14230 or ISO 14229-1), it is possible to dynamically define data records at run time, which contain values from other records identified by a local identifier (LID), a common identifier (CID) or an address in the ECU memory, etc. These new records are identified by the dynamic identifier (DYN-ID). In ISO 14230 specification, this identifier is called “dynamically defined local identifier” (DDLID).

Figure 136 — Creation of a new data record in ISO 14230 shows the process of creating a new data record with ISO 14230 by referencing values from other records identified by LIDs. The record with LID=01h consists of three values whereby the other one (LID=0x02) contains two values. One value from the first record and one value from second one are grouped into the new record with DYN-ID=0xF0.

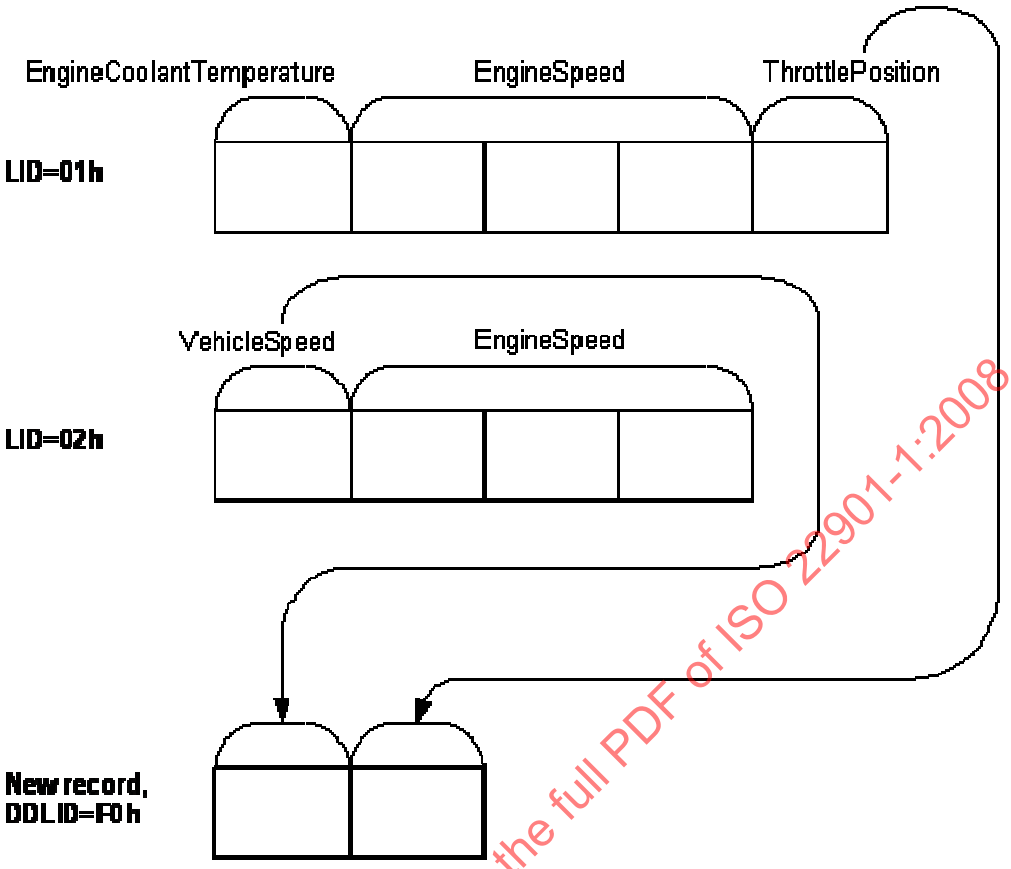


Figure 136 — Creation of a new data record in ISO 14230

Figure 137 — Example of DYN-DEFINED-SPEC shows an example of a DYN-DEFINED-SPEC object. For creating a new data record in the definition mode “LOCAL-ID” the table TAB_LIDs and one of the DYN-IDs 0xF0 or 0xF1 are to be used. To combine COMMON-IDs to one DYN-ID the table TAB_CIDs and one of the DYN-IDs 0xF2 or 0xF3 are used. The table with SEMANTIC=“LOCAL-ID” holds the description of all records identified by a local identifier and the table with SEMANTIC=“COMMON-ID” defines data records identified by a common identifier. In the first table, TABLE-ROW with KEY=0x01 refers to the structure LID_01, i.e. the data record with LID=0x01 is given by the structure LID_01. The latter consists of three values (parameters): EngineCoolantTemperature, EngineSpeed and ThrottlePosition. Other table rows are interpreted in the same way.

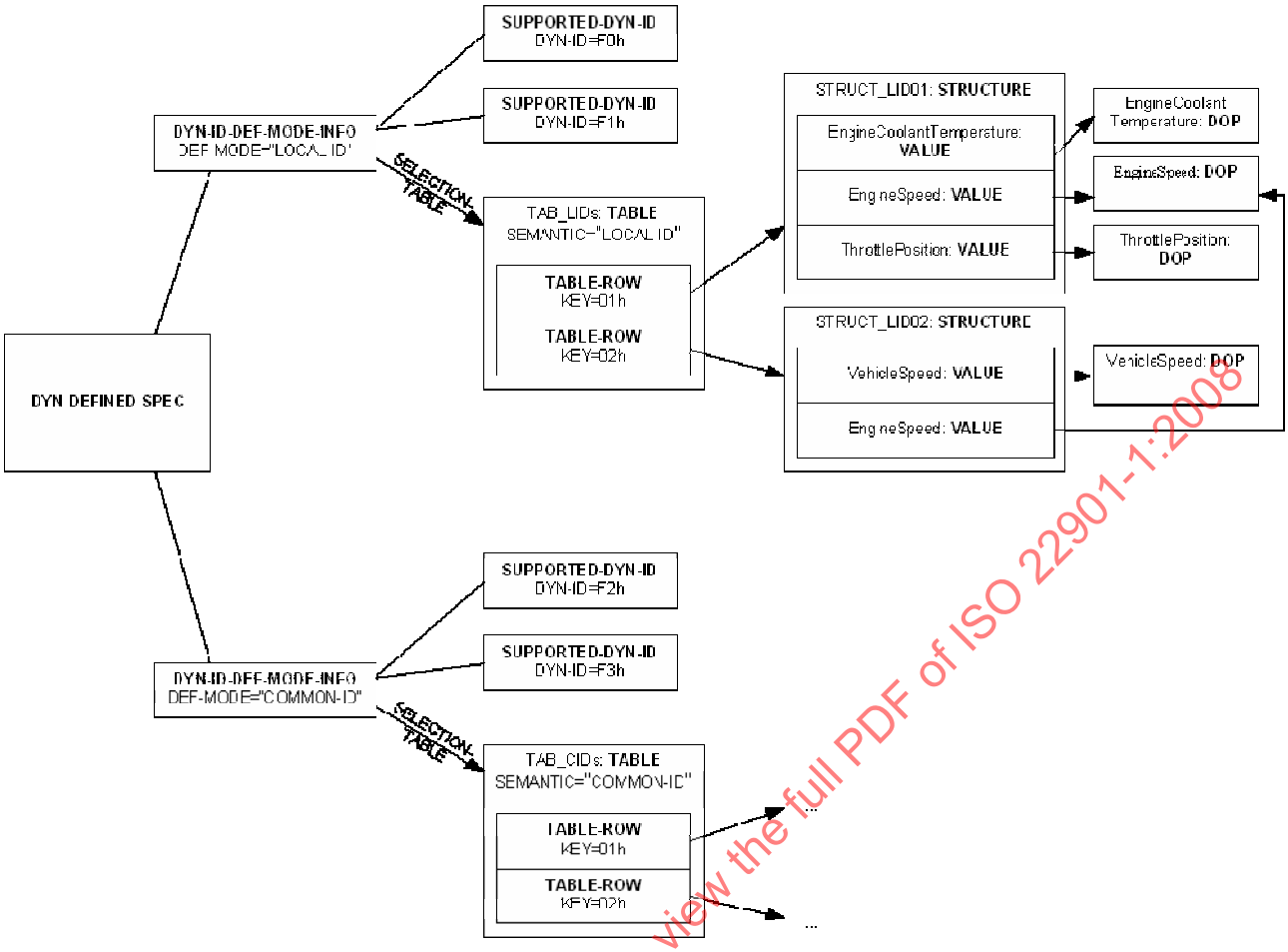


Figure 137 — Example of DYN-DEFINED-SPEC

The member DIAGNOSTIC-CLASS of DIAG-COMM is used to indicate the D-server that the service/job defines a new DYN-ID (DIAGNOSTIC-CLASS="DYN-DEF-MESSAGE"), that the service/job reads data using the DYN-ID defined before (DIAGNOSTIC-CLASS="READ-DYN-DEF-MESSAGE") or that the service/job clears information about the DYN-ID definitions (DIAGNOSTIC-CLASS="CLEAR-DYN-DEF-MESSAGE"). In the following examples the ISO 14230 service with ID=0x21 has the DIAGNOSTIC-CLASS "READ-DYN-DEF-MESSAGE". There are also two services with ID=2Ch. One of them has the DIAGNOSTIC-CLASS "DYN-DEF-MESSAGE" and the other one "CLEAR-DYN-DEF-MESSAGE". A complete implementation of these services can be found in Clause E.2.

7.4.2.2 Dynamic case

In the first example, the user constructs the DDLID (assisted by the application) at run time.

Figure 138 shows an example for the process of defining of DYN-IDs with the service 0x2C. The request of the service contains the parameters ServiceID (=0x2C) and DDLID (it is filled with the new DYN-ID by the user at run time). The last parameter references an END-OF-PDU-FIELD, which references the structure that is used to define an item for the new record. This structure is repeated for each new item in the defined DYN-ID record. The user fills the values in this structure according to the used protocol (in this example, it is ISO 14230).

If a DYN-DEF-MESSAGE is to be sent to the ECU, the application can offer the user the choice of the following items:

a) one of the definition modes;

b) one of the supported DYN-IDs in the selected definition mode;

c) a specific row of the TABLE that belongs to the selected definition mode;

d) concrete values of the structures referenced by the selected row (in this example, these are ThrottlePosition and VehicleSpeed).

This information is used to fill the structure referenced from the END-OF-PDU-FIELD. Alternatively, an advanced user can fill this structure manually, e.g. if the application does not provide an assistant for assembling of DYN-ID records. In Figure 138, this structure is called “STRUCT_DDLIDItem” and it is filled two times meaning that two values are combined to one record with DYN-ID=F0h.

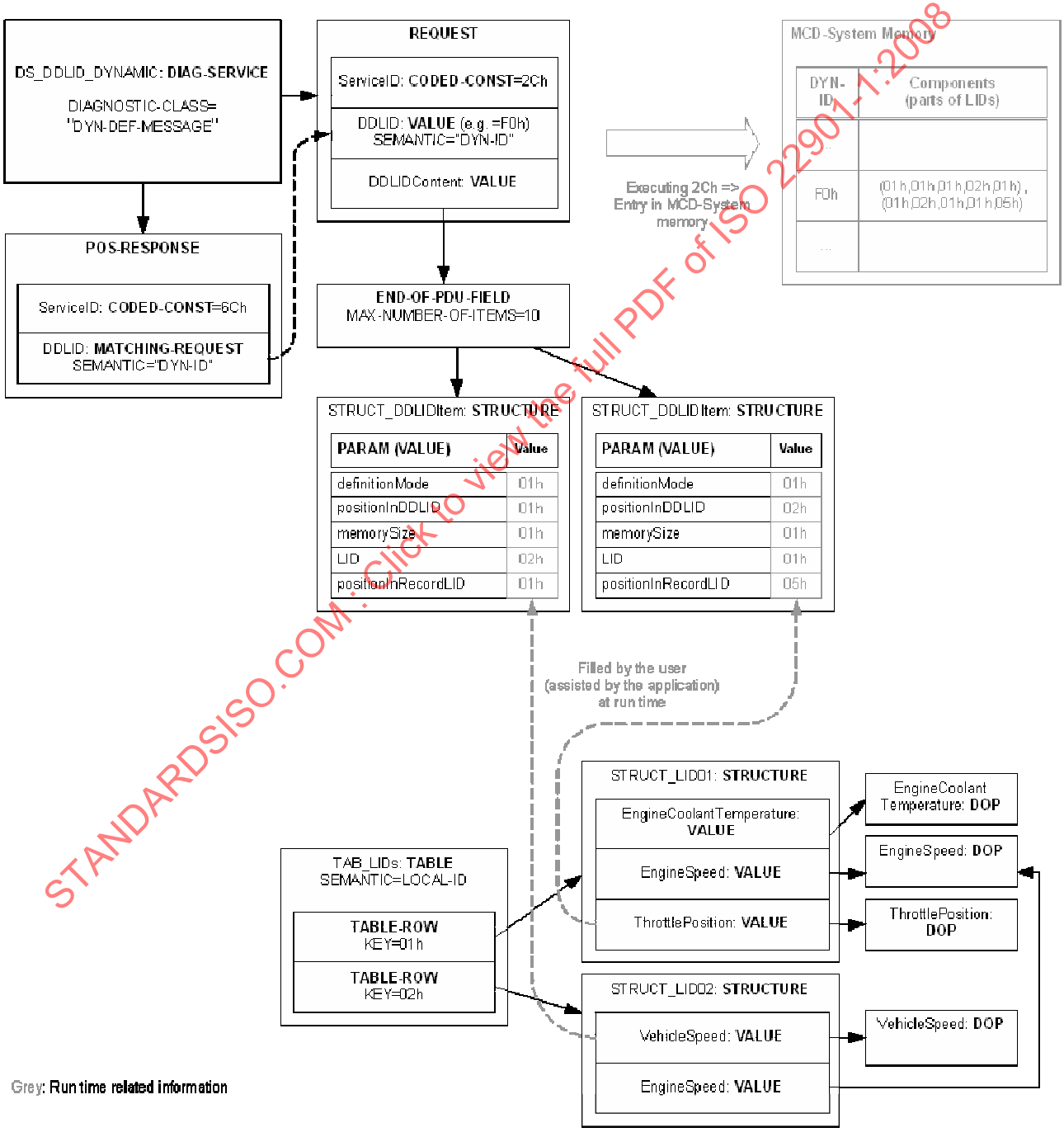


Figure 138 — Example of DYN-DEF-MESSAGE, dynamic case

After executing the DYN-DEF-MESSAGE service, the D-server shall store the following information:

— DYN-ID (in ISO 14230: DDLID);

— information about components of the DYN-ID record depending on the used protocol (in ISO 14230 definition mode, position in LID, memory size, LID and position in DDLID).

This information is sufficient to interpret the DYN-ID record at a later time.

Now the defined DYN-ID record can be read by the READ-DYN-DEF-MESSAGE service (0x21 in ISO 14230) as shown in Figure 139 — Example of READ-DYN-DEF-MESSAGE, static and dynamic cases.

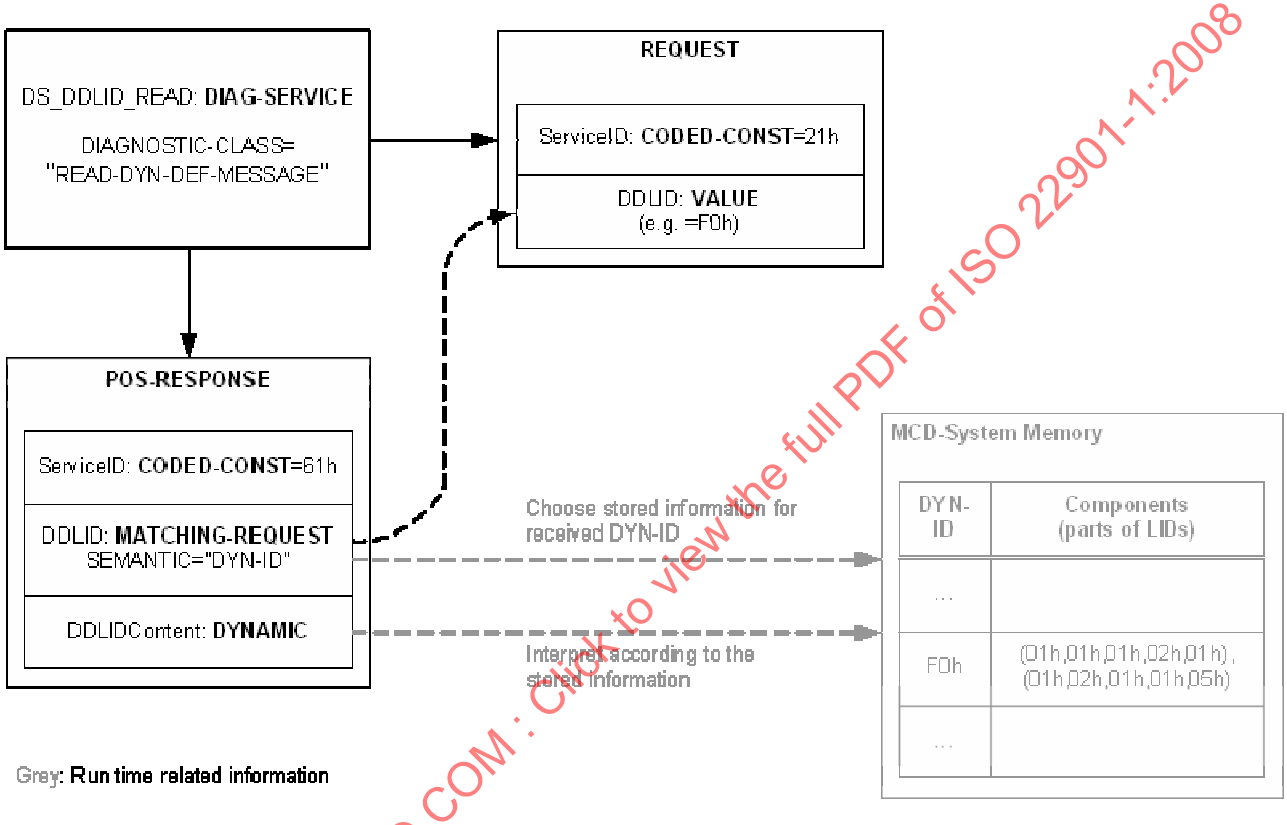


Figure 139 — Example of READ-DYN-DEF-MESSAGE, static and dynamic cases

In the response a parameter with SEMANTIC=“DYN-ID” indicates that the entry with DYN-ID=F0h, stored in the memory, should be chosen for the following interpretation of data. The next parameter (of type DYNAMIC) has no reference to any DOP, because its interpretation is only known at run time and the data stored in the memory is to be used for it.

The D-server does not have any information about the TABLE used for definition of the DYN-ID record (this information has only the application). The D-server has only the information about the content of the DYN-ID record. In this example, the following information is available for each component of the DYN-ID record:

a) SEMANTIC of the TABLE (derived from the parameter “definitionMode” in the structure STRUCT_DDLIDItem);

b) identifier (parameter LID in the structure STRUCT_DDLIDItem);

c) the remaining parameters used for the definition of the DYN-ID record (in this example, the remaining parameters of the structure STRUCT_DDLIDItem).

All TABLEs listed in DYN-DEFINED-SPEC with the given SEMANTIC are searched for the row with the key that is equal to the given LID. The first row found is then used to interpret the received data in the way it was defined by the DYN-DEF-MESSAGE service.

In this way the read service can be coded in ODX without knowing which LID's (or CID's etc.) the user will group during run time. The dynamic part of the service shall be implemented by the D-server and is dependant on the diagnosis protocol.

7.4.2.3 Static case

The author of an ODX instance can also assign a set of data to a DYN-ID defining a DYN-DEF-MESSAGE service with fixed values in the structure that represent a data item. If *n* is the number of items grouped to a DYN-ID record, *n* structures are referenced from *n* parameters of the request (in Figure 140 *n*=2). Each structure has parameters of type CODED-CONST. The parameter count can be different in different protocols. In ISO 14230, there are five ones: definitionMode, positionInDDLID, memorySize, LID and positionInRecordLID.

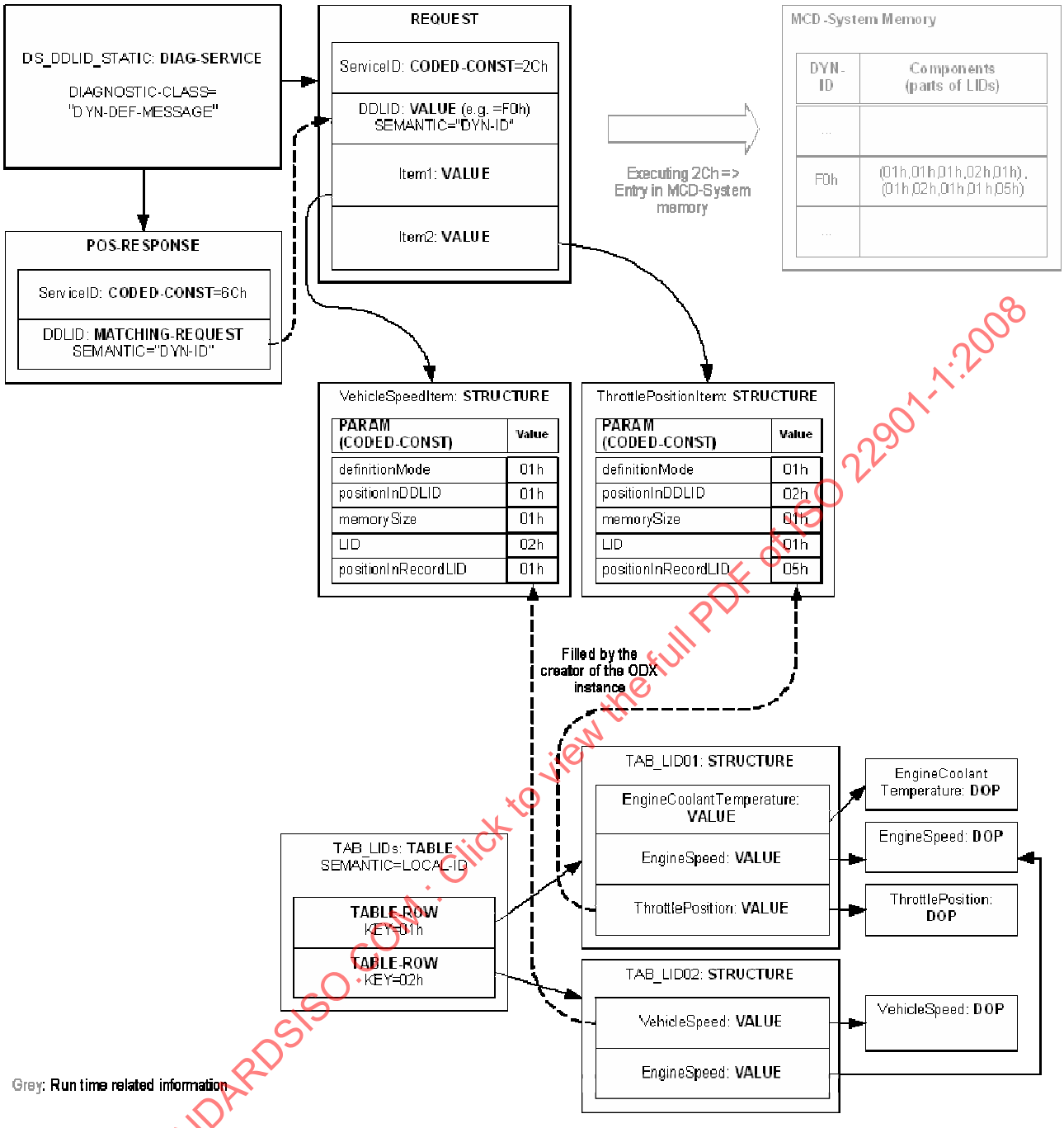


Figure 140 — Example of DYN-DEF-MESSAGE, static case

The execution of the service occurs in the same manner as in the dynamic case. Information about DYN-ID and its content is stored in the memory again. The READ-DYN-DEF-MESSAGE service can be used without any changes to read this DYN-ID record. From the point of view of the D-server there is no difference between the dynamic and the static case.

7.4.3 Variant identification

ODX can manage different variants of an ECU. Only one ECU variant (either BASE-VARIANT or ECU-VARIANT) may be active at one time during runtime. This is necessary, since they are all identified via the same physical address. Before a variant identification session is started, the user normally selects a BASE-VARIANT. Then, the identification of an ECU-VARIANT is performed by the execution of DIAG-COMMs and evaluating their responses. Depending on the protocol and the ECU implementation one or several DIAG-

COMMs are executed. A variant is identified by comparing one or more results returned by the ECU with the variant identification data in the ECU-VARIANT instances.

Variant identification of an already identified ECU-VARIANT is also possible. In order to ensure that this repeated variant identification leads to identical results the following restrictions to DIAG-COMMs with DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION shall be fulfilled: Changing the behaviour of the DIAG-COMMs on ECU-VARIANT level is not allowed. That means that neither the definition nor overriding of the DIAG-COMMs itself is possible. Also, overwriting any of the elements in the transitive closure of a DIAG-COMM's REQUEST, POS-RESPONSE, NEG-RESPONSE is forbidden. Some examples for the consequences of these restrictions are:

a) a DIAG-SERVICE having DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION shall not have the flag IS-CYCLIC=true;

b) a DIAG-COMM with DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION shall neither be defined nor overridden on ECU-VARIANT level;

c) any element of the transitive closure of a DIAG-COMM's REQUEST, POS-RESPONSE, NEG-RESPONSE with DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION shall not be overridden in ECU-VARIANTs.

The SHORT-NAME of the response parameter used for variant identification shall not be part of any valid GLOBAL-NEG-RESPONSEs in the BASE-VARIANT and all ECU-VARIANTs. See Figure 141 — UML representation and structure of variant identification.

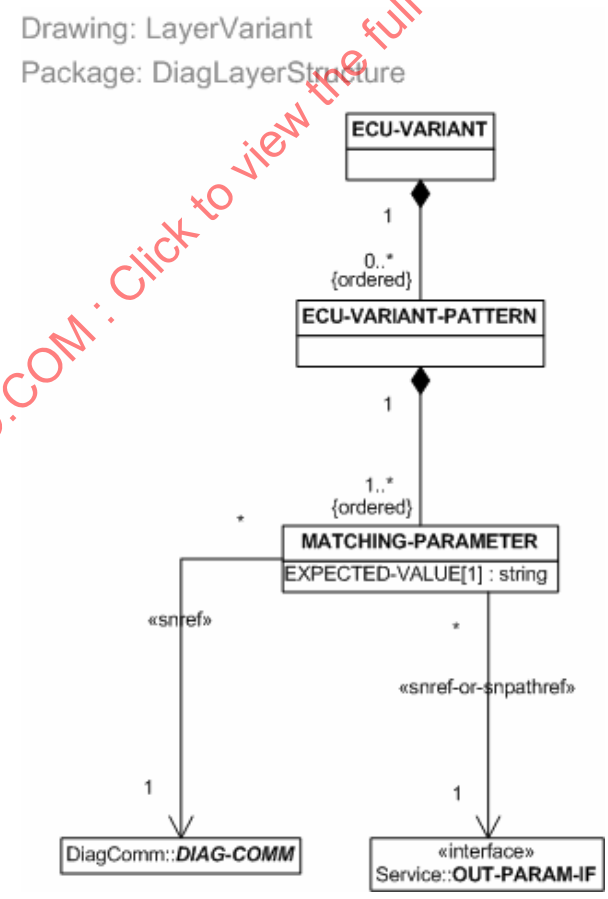


Figure 141 — UML representation and structure of variant identification

ECU-VARIANT-PATTERNs are organized in a list corresponding to the different ECU-VARIANTs that can be described via the ODX document. More than one ECU-VARIANT-PATTERN might be necessary if different ECU samples (with different identifications) do not differ regarding their diagnostic behaviour and are defined as one ECU-VARIANT in the ODX container. Every ECU-VARIANT-PATTERN contains a list of MATCHING-PARAMETER with values identifying the current ECU. All ECU-VARIANT-PATTERN are disjoint, each of them identifies one ECU-VARIANT. The different MATCHING-PARAMETER might be determined via more than one DIAG-COMM. The ECU-VARIANT-PATTERNs are only valid for one ECU-Variant layer. The MATCHING-PARAMETER element references a DIAG-COMM via snref and one of its OUT-PARAM-IF via snref or snpathref. In case the DIAG-COMM is a DIAG-SERVICE and it contains more than one response (positive or negative), the snref or snpathref shall be resolvable in at least one POS-RESPONSE or NEG-RESPONSE.

A parameter of a GLOBAL-NEG-RESPONSE shall not be used for variant identification. This parameter shall also refer to a DATA-OBJECT-PROP. If the reference is not resolvable in the actual response (either no parameter with the pathname can be found or the referenced parameter is not a DATA-OBJECT-PROP) the variant is not successfully matched. The starting point for resolving the short-name-path is in case of a DIAG-SERVICE the PARAMS element within the RESPONSE of the referenced DIAG-SERVICE. In case of a SINGLE-ECU-JOB it is the OUTPUT-PARAMS element within the SINGLE-ECU-JOB. The short-name-path is resolved using the Multiple-Param-Algorithm (see 7.3.13.3.2).

The variant identification algorithm works in the following way: All variants belonging to the same base variant are stored in an ordered list depending on the rising alphabetical order of their SHORT-NAMEs (this ensures deterministic behaviour between different implementations of a D-server). The first ECU-VARIANT's patterns are checked first, again in the order of their appearance within the ECU-VARIANT.

A MATCHING-PARAMETER matches if the EXPECTED-VALUE and the physical value of the response parameter referenced with the OUT-PARAM-IF-SNREF are equal. If the referenced PARAM lies within a FIELD the meaning is the following: The comparison is done for every FIELD-result repeatedly. The matching is successful, if one of the result values of the FIELD results matches with the MATCHING-PARAMETER. The same holds true for the set of parameters identified with the multiple param algorithm for OUT-PARAM-IF-SNPATHREF. The MATCHING-PARAMETER matches if the physical value of at least one of these parameters equals the EXPECTED-VALUE. An ECU-VARIANT-PATTERN matches, if all its MATCHING-PARAMETERs match. A variant is detected if at least one ECU-VARIANT-PATTERN matches. As soon as one ECU-VARIANT matched, the variant identification terminates. The variant is selected. If no variant could be identified, an exception occurs.

EXAMPLE Defining a set of ECU-VARIANT-PATTERN:

DIAG-COMM DS_IdentificationRead

Request: 0x1A, 0x86

Response: 0x5A, 0x86, 0x16, 0x80, 0x03

Pattern 1 ExpectedValue = Supplier1

ExpectedValue = 32769

Pattern 2 ExpectedValue = Supplier2 (CodedValue = 0x16)

ExpectedValue = 32771 (CodedValue = 0x8003)

ECU Variant2 using Pattern 2:

```
<ECU-VARIANT ID = "Variant2">
  <SHORT-NAME>Variant2</SHORT-NAME>
  <ECU-VARIANT-PATTERNS>
    <ECU-VARIANT-PATTERN>
      <MATCHING-PARAMETERS>
        <MATCHING-PARAMETER>
```

```
          <EXPECTED-VALUE>Supplier2</EXPECTED-VALUE>
          <DIAG-COMM-SNREF SHORT-NAME = "DS_IdentificationRead"/>
            <OUT-PARAM-IF-SNREF SHORT-NAME = "SupplierIdentification"/>
        </MATCHING-PARAMETER>
        <MATCHING-PARAMETER>
          <EXPECTED-VALUE>32771</EXPECTED-VALUE>
          <DIAG-COMM-SNREF SHORT-NAME = "DS_IdentificationRead"/>
             <OUT-PARAM-IF-SNREF SHORT-NAME = "DiagnosticVersion"/>
        </MATCHING-PARAMETER>
      </MATCHING-PARAMETERS>
    </ECU-VARIANT-PATTERN>
  </ECU-VARIANT-PATTERNS>
  <PARENT-REFS>
    <PARENT-REF ID-REF = "ECU_xyz" xsi:type = "BASE-VARIANT-REF"/>
  </PARENT-REFS>
</ECU-VARIANT>
```

Positive Response of DIAG-COMM “IdentificationRead”:

```
<POS-RESPONSE ID = "RESP_IdentificationRead">
  <SHORT-NAME>RESP_IdentificationRead</SHORT-NAME>
  <PARAMS>
    <PARAM SEMANTIC = "DATA" xsi:type = "VALUE">
      <SHORT-NAME>SupplierIdentification</SHORT-NAME>
      <BYTE-POSITION>2</BYTE-POSITION>
      <DOP-REF ID-REF = "DOP_SupplierList"/>
    </PARAM>
    <PARAM SEMANTIC = "DATA" xsi:type = "VALUE">
      <SHORT-NAME>DiagnosticVersion</SHORT-NAME>
      <BYTE-POSITION>3</BYTE-POSITION>
        <DOP-REF ID-REF = "DOP_2ByteHexDump"/>
    </PARAM>
  </PARAMS>
</POS-RESPONSE>
```

DOP SupplierIdentification:

```
<DATA-OBJECT-PROP ID="DOP_SupplierList">
  <SHORT-NAME>DOP_SupplierList</SHORT-NAME>
  <COMPU-METHOD>
    <CATEGORY>TEXTTABLE</CATEGORY>
    <COMPU-INTERNAL-TO-PHYS>
      <COMPU-SCALES>
        <COMPU-SCALE>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">21</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">21</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>Supplier1</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
        <COMPU-SCALE>
          <LOWER-LIMIT INTERVAL-TYPE="CLOSED">22</LOWER-LIMIT>
          <UPPER-LIMIT INTERVAL-TYPE="CLOSED">22</UPPER-LIMIT>
          <COMPU-CONST>
            <VT>Supplier2</VT>
          </COMPU-CONST>
        </COMPU-SCALE>
```

```
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
  </COMPU-METHOD>
  <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
    <BIT-LENGTH>8</BIT-LENGTH>
  </DIAG-CODED-TYPE>
  <PHYSICAL-TYPE BASE-DATA-TYPE="A_UNICODE2STRING"/>
</DATA-OBJECT-PROP>
```

DOP DiagnosticVersion:

```
<DATA-OBJECT-PROP ID = "DOP_2ByteHexDump">
  <SHORT-NAME>DOP_2ByteHexDump</SHORT-NAME>
  <COMPU-METHOD>
    <CATEGORY>IDENTICAL</CATEGORY>
  </COMPU-METHOD>
  <DIAG-CODED-TYPE BASE-DATA-TYPE = "A_UINT32" xsi:type = "STANDARD-LENGTH-TYPE">
    <BIT-LENGTH>16</BIT-LENGTH>
  </DIAG-CODED-TYPE>
  <PHYSICAL-TYPE BASE-DATA-TYPE = "A_UINT32" DISPLAY-RADIX = "HEX"/>
</DATA-OBJECT-PROP>
```

7.4.4 Base variant identification scenario

7.4.4.1 Overview

This subclause (7.4.4) summarizes how data for a base variant identification scenario can be authored and what the runtime semantics of these data structures are. Base variant identification is applicable to all situations where an alternative built-in of ECUs needs to be resolved. Alternative ECUs means that there is a set of possible ECUs exactly one of which is actually built into a vehicle. Typical examples are different engine controllers for different engines or different radio ECUs from the low-end to the high-end or ECUs performing the identical function in a vehicle, but are supplied from different vendors. Each ECU in such a set shall correspond to a BASE-VARAINT.

7.4.4.2 ODX data structures for BASE-VARIANT identification scenario

To group alternative BASE-VARIANTs together, the ECU-GROUP element shall be used. Through the GROUP-MEMBER class it references BASE-VARIANTs. For every reference it can also be specified which LOGICAL-LINK is to be used when resolving this variant. Resolution via functional addressing and resolution via physical addressing are both supported. Consequently, two possible references to LOGICAL-LINKs are supported: A physical resolution link and a functional resolution link.

The following constraints apply:

a) one of the two link types shall be defined for every GROUP-MEMBER: the specification of both kinds of links is equally possible;

b) the FUNCT-RESOLUTION-LINK points to a LOGICAL-LINK that references a FUNCTIONAL-GROUP and no BASE-VARIANT;

c) the PHYS-RESOLUTION-LINK points to a LOGICAL-LINK that references a BASE-VARIANT.

To define how a particular BASE-VARIANT within the group can be identified by communication with the vehicle, every BASE-VARIANT may contain a BASE-VARIANT-PATTERN (see Figure 45). A BASE-VARIANT-PATTERN contains a set of MATCHING-BASE-VARIANT-PARAMETERs, which contains an EXPECTED-VALUE. This EXPECTED-VALUE is matched against the content of a referenced DIAG-COMM's

parameter referenced through OUT-PARAM-IF. The OUT-PARAM-IF is referenced either via SHORT-NAME or via short-name-path. The starting point for resolving the short-name-path is, in case of the DIAG-COMM being a DIAG-SERVICE, the PARAMS element representation of the received RESPONSE at run time. In case of it being a SINGLE-ECU-JOB, the starting point is the OUTPUT-PARAMS element representation of the jobs execution result. The short-name-path is resolved using the Multiple-Param-Algorithm (see 7.3.13.3.2).

A MATCHING-BASE-VARIANT-PARAMETER matches if the EXPECTED-VALUE and the physical value of the response parameter referenced with the OUT-PARAM-IF-SNREF are equal. If the referenced PARAM lies within a FIELD the meaning is the following: The comparison is done for every FIELD-result repeatedly. The matching is successful, if one of the result values of the FIELD results matches with the MATCHING-BASE-VARIANT-PARAMETER. The same holds true for the set of parameters identified with the multiple param algorithm for OUT-PARAM-IF-SNPATHREF. The MATCHING-BASE-VARIANT-PARAMETER matches if the physical value of at least one of these parameters equals the EXPECTED-VALUE.

If all MATCHING-BASE-VARIANT-PARAMETERs of one BASE-VARIANT-PATTERN are considered as matched, the whole pattern is considered as matched and the BASE-VARIANT is identified.

Every MATCHING-BASE-VARIANT-PARAMETER also has an attribute that is to be used to determine whether the DIAG-COMM to retrieve the actual value from the vehicle is to be executed with physical or functional addressing (USE-PHYSICAL-ADDRESSING), with the default being physical addressing. If the referenced DIAG-COMM only supports one addressing mode, the value of this attribute shall be ignored.

The following constraints apply:

— If the BASE-VARIANT-PATTERN of BASE-VARIANT has at least one MATCHING-BASE-VARIANT-PARAMETER with USE-PHYSICAL-ADDRESSING = false, the BASE-VARIANT shall have a FUNCTIONAL-RESOLUTION-LINK specified in every ECU-GROUP it appears in.

— If the BASE-VARIANT-PATTERN of BASE-VARIANT has at least one MATCHING-BASE-VARIANT-PARAMETER with USE-PHYSICAL-ADDRESSING = true or USE-PHYSICAL-ADDRESSING is not specified, the BASE-VARIANT shall have a PHYS-RESOLUTION-LINK specified in every ECU-GROUP it appears in.

Together with the referenced OUT-PARAM an implicit statement is made as to which kind of response is expected (also refer to variant identification, where this mechanism also applies). For example, if the OUT-PARAM-IF references the negative response code of a negative response and at runtime a positive response is received, the MATCHING-BASE-VARIANT-PARAMETER is considered to not be matched.

7.4.4.3 Sequence of events for BASE-VARIANT identification

To explain the semantics of base variant identification in a more detailed manner, the defined data is to be interpreted as described below.

a) The BASE-VARIANTs are iterated in their order within the ECU-GROUP.

b) For the current BASE-VARIANT, iterate through the MATCHING-BASE-VARIANT-PARAMETERs in the order they are defined in the BASE-VARIANT-PATTERN:

   1) as soon as one MATCHING-BASE-VARIANT-PARAMETER is not matched, the BASE-VARIANT is considered as not identified; continue with a);

   2) as soon as all MATCHING-BASE-VARIANT-PARAMETERs within the BASE-VARIANT-PATTERN of the BASE-VARIANT under investigation matched, that BASE-VARIANT is considered identified.

7.4.4.4 ODX example

7.4.4.4.1 The following example shows a DIAG-LAYER-CONTAINER with one FUNCTIONAL-GROUP and two BASE-VARIANTs inheriting from it. Both BASE-VARIANTs contain a BASE-VARIANT-PATTERN which

matches the Software Version Number against an expected value. The DIAG-SERVICE to read the identification value is authored on the level of the FUNCTIONAL-GROUP. It is inherited as-is by both BASE-VARIANTs. As it has an ADDRESSING mode of FUNCTIONAL-OR-PHYSICAL it can be used for both functional and physical base variant identification. In the given example BV1 shall be resolved functionally, while BV2 shall be resolved physically (cf. value of USE-PHYSICAL-ADDRESSING element).

```
<ODX xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="odx.xsd" MODEL-
VERSION="2.2.0">
    <DIAG-LAYER-CONTAINER ID="DLC_10423">
        <SHORT-NAME>DLC_A</SHORT-NAME>
        <FUNCTIONAL-GROUPS>
            <FUNCTIONAL-GROUP ID="FG_10424">
                <SHORT-NAME>FG</SHORT-NAME>
                <DIAG-DATA-DICTIONARY-SPEC>
                    <DATA-OBJECT-PROPS>
                        <DATA-OBJECT-PROP ID="DOP_11081">
                            <SHORT-NAME>HexByteField</SHORT-NAME>
                            <COMPU-METHOD>
                                <CATEGORY>IDENTICAL</CATEGORY>
                            </COMPU-METHOD>
                            <DIAG-CODED-TYPE BASE-DATA-TYPE="A_BYTEFIELD" TERMINATION="END-OF-PDU" xsi:type="MIN-
MAX-LENGTH-TYPE">
                                <MAX-LENGTH>16</MAX-LENGTH>
                                <MIN-LENGTH>1</MIN-LENGTH>
                            </DIAG-CODED-TYPE>
                            <PHYSICAL-TYPE BASE-DATA-TYPE="A_BYTEFIELD"/>
                        </DATA-OBJECT-PROP>
                    </DATA-OBJECT-PROPS>
                </DIAG-DATA-DICTIONARY-SPEC>
                <DIAG-COMMS>
                    <DIAG-SERVICE ID="DS_11251" ADDRESSING="FUNCTIONAL-OR-PHYSICAL">
                        <SHORT-NAME>DS_ReadSWVersionNumber</SHORT-NAME>
                        <REQUEST-REF ID-REF="REQ_10471"/>
                        <POS-RESPONSE-REFS>
                            <POS-RESPONSE-REF ID-REF="PR_11131"/>
                        </POS-RESPONSE-REFS>
                    </DIAG-SERVICE>
                </DIAG-COMMS>
                <REQUESTS>
                    <REQUEST ID="REQ_10471">
                        <SHORT-NAME>REQ_ReadSWVersionNumber</SHORT-NAME>
                        <PARAMS>
                            <PARAM xsi:type="CODED-CONST">
                                <SHORT-NAME>PARAM_SID</SHORT-NAME>
                                <BYTE-POSITION>0</BYTE-POSITION>
                                <BIT-POSITION>0</BIT-POSITION>
                                <CODED-VALUE>34</CODED-VALUE>
                                <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
                                    <BIT-LENGTH>8</BIT-LENGTH>
                                </DIAG-CODED-TYPE>
                            </PARAM>
                            <PARAM xsi:type="CODED-CONST">
                                <SHORT-NAME>DID_SWVersionNumber</SHORT-NAME>
                                <BYTE-POSITION>1</BYTE-POSITION>
                                <BIT-POSITION>0</BIT-POSITION>
                                <CODED-VALUE>18193</CODED-VALUE>
                                <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
                                    <BIT-LENGTH>16</BIT-LENGTH>
```

```
                    </DIAG-CODED-TYPE>
                </PARAM>
            </PARAMS>
        </REQUEST>
    </REQUESTS>
    <POS-RESPONSES>
        <POS-RESPONSE ID="PR_11131">
            <SHORT-NAME>PR_ReadDataByIdentifier</SHORT-NAME>
            <PARAMS>
                <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>PARAM_SID</SHORT-NAME>
                    <BYTE-POSITION>0</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <CODED-VALUE>98</CODED-VALUE>
                    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
                        <BIT-LENGTH>8</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
                </PARAM>
                <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>DID_SWVersionNumber</SHORT-NAME>
                    <BYTE-POSITION>1</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <CODED-VALUE>18193</CODED-VALUE>
                    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
                        <BIT-LENGTH>16</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
                </PARAM>
                <PARAM xsi:type="VALUE">
                    <SHORT-NAME>Data_SWVersionNumber</SHORT-NAME>
                    <BYTE-POSITION>3</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <DOP-SNREF SHORT-NAME="HexByteField"/>
                </PARAM>
            </PARAMS>
        </POS-RESPONSE>
    </POS-RESPONSES>
</FUNCTIONAL-GROUP>
</FUNCTIONAL-GROUPS>
<BASE-VARIANTS>
    <BASE-VARIANT ID="BV_11271">
        <SHORT-NAME>BV1</SHORT-NAME>
        <BASE-VARIANT-PATTERN>
            <MATCHING-BASE-VARIANT-PARAMETERS>
                <MATCHING-BASE-VARIANT-PARAMETER>
                    <EXPECTED-VALUE>AD47110815</EXPECTED-VALUE>
                    <USE-PHYSICAL-ADDRESSING>false</USE-PHYSICAL-ADDRESSING>
                    <DIAG-COMM-SNREF SHORT-NAME="DS_ReadSWVersionNumber"/>
                    <OUT-PARAM-IF-SNREF SHORT-NAME="Data_SWVersionNumber"/>
                </MATCHING-BASE-VARIANT-PARAMETER>
            </MATCHING-BASE-VARIANT-PARAMETERS>
        </BASE-VARIANT-PATTERN>
        <PARENT-REFS>
            <PARENT-REF xsi:type="FUNCTIONAL-GROUP-REF" ID-REF="FG_10424"/>
        </PARENT-REFS>
    </BASE-VARIANT>
    <BASE-VARIANT ID="BV_11331">
        <SHORT-NAME>BV2</SHORT-NAME>
        <BASE-VARIANT-PATTERN>
```

```
                    <MATCHING-BASE-VARIANT-PARAMETERS>
                        <MATCHING-BASE-VARIANT-PARAMETER>
                            <EXPECTED-VALUE>BV2_SW_V_NB</EXPECTED-VALUE>
                            <DIAG-COMM-SNREF SHORT-NAME="DS_ReadSWVersionNumber"/>
                            <OUT-PARAM-IF-SNREF SHORT-NAME="Data_SWVersionNumber"/>
                        </MATCHING-BASE-VARIANT-PARAMETER>
                    </MATCHING-BASE-VARIANT-PARAMETERS>
                </BASE-VARIANT-PATTERN>
                <PARENT-REFS>
                    <PARENT-REF xsi:type="FUNCTIONAL-GROUP-REF" ID-REF="FG_10424"/>
                </PARENT-REFS>
            </BASE-VARIANT>
        </BASE-VARIANTS>
    </DIAG-LAYER-CONTAINER>
</ODX>
```

7.4.4.4.2 The VEHICLE-INFORMATION shown below, now groups both BASE-VARIANTs into an ECU-GROUP. The corresponding GROUP-MEMBERs reference the BASE-VARIANTs and their resolution link. For base variant BV1 a functional and physical resolution link is given, while for BV2 only a physical resolution link exists.

NOTE Through mixing MATCHING-BASE-VARIANT-PARAMETERs with different values for the element USE-PHYSICAL-ADDRESSING in one BASE-VARIANT-PATTERN, both kinds of links may be needed to resolve the base variant.

```
<ODX xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="odx.xsd" MODEL-
VERSION="2.2.0">
    <VEHICLE-INFO-SPEC ID="VIS_10401">
        <SHORT-NAME>VIS</SHORT-NAME>
        <VEHICLE-INFORMATIONS>
            <VEHICLE-INFORMATION>
                <SHORT-NAME>VI1</SHORT-NAME>
                <LOGICAL-LINKS>
                    <LOGICAL-LINK ID="LL_11481" xsi:type="MEMBER-LOGICAL-LINK">
                        <SHORT-NAME>LL_FG</SHORT-NAME>
                        <PHYSICAL-VEHICLE-LINK-REF ID-REF="PVL_11511"/>
                        <FUNCTIONAL-GROUP-REF ID-REF="FG_10424" DOCREF="DLC_A" DOCTYPE="CONTAINER"/>
                    </LOGICAL-LINK>
                    <LOGICAL-LINK ID="LL_11521" xsi:type="MEMBER-LOGICAL-LINK">
                        <SHORT-NAME>LL_BV1</SHORT-NAME>
                        <PHYSICAL-VEHICLE-LINK-REF ID-REF="PVL_11511"/>
                        <BASE-VARIANT-REF ID-REF="BV_11271" DOCREF="DLC_A" DOCTYPE="CONTAINER"/>
                    </LOGICAL-LINK>
                    <LOGICAL-LINK ID="LL_145261" xsi:type="MEMBER-LOGICAL-LINK">
                        <SHORT-NAME>LL_BV2</SHORT-NAME>
                        <PHYSICAL-VEHICLE-LINK-REF ID-REF="PVL_11511"/>
                        <BASE-VARIANT-REF ID-REF="BV_11331" DOCREF="DLC_A" DOCTYPE="CONTAINER"/>
                    </LOGICAL-LINK>
                </LOGICAL-LINKS>
                <ECU-GROUPS>
                    <ECU-GROUP>
                        <SHORT-NAME>ECUG_SAMPLE</SHORT-NAME>
                        <GROUP-MEMBERS>
                            <GROUP-MEMBER>
                                <BASE-VARIANT-REF ID-REF="BV_11331" DOCREF="DLC_A" DOCTYPE="CONTAINER"/>
                                <PHYS-RESOLUTION-LINK-REF ID-REF="LL_145261"/>
                            </GROUP-MEMBER>
                            <GROUP-MEMBER>
```

```
                        <BASE-VARIANT-REF ID-REF="BV_11271" DOCREF="DLC_A" DOCTYPE="CONTAINER"/>
                        <FUNCT-RESOLUTION-LINK-REF ID-REF="LL_11481"/>
                        <PHYS-RESOLUTION-LINK-REF ID-REF="LL_11521"/>
                    </GROUP-MEMBER>
                </GROUP-MEMBERS>
            </ECU-GROUP>
        </ECU-GROUPS>
        <PHYSICAL-VEHICLE-LINKS>
            <PHYSICAL-VEHICLE-LINK ID="PVL_11511" TYPE="ISO_11898_2_DWCAN">
                <SHORT-NAME>PVL_A</SHORT-NAME>
                <VEHICLE-CONNECTOR-PIN-REFS>
                    <VEHICLE-CONNECTOR-PIN-REF ID-REF="VCPR_0815"/>
                </VEHICLE-CONNECTOR-PIN-REFS>
            </PHYSICAL-VEHICLE-LINK>
        </PHYSICAL-VEHICLE-LINKS>
    </VEHICLE-INFORMATION>
</VEHICLE-INFORMATIONS>
</VEHICLE-INFO-SPEC>
</ODX>
```

7.4.4.4.3 In the example given, resolution of base variant “BV1” would proceed as follows:

— open Logical Link on functional group level (LL_FG);

— execute DIAG-SERVICE DS_ReadSWVersionNumber functionally;

— compare the response parameter against the expected value.

7.4.4.4.4 Base variant “BV2” would be resolved as follows:

— open Logical Link on physical level (LL_BV2);

— execute DIAG-SERVICE DS_ReadSWVersionNumber physically;

— compare the response parameter against the expected value.

7.4.5 Diagnostic trouble code description

The service ReadDTCByStatus (ServiceId=0x18) is used in ISO 14230-3 to read diagnostic trouble codes (DTCs) from the ECU’s memory using a definite status. After that, the service ReadStatusOfDTC (ServiceId=0x17) can be used to read environment conditions for a single DTC. The complete implementation of both services can be found in Clause E.2.

At first, the service ReadDTCByStatus should be discussed.

Figure 142 — Objects used by the ReadDTCByStatus service of ISO 14230-3 gives an overview of the objects used by this service.

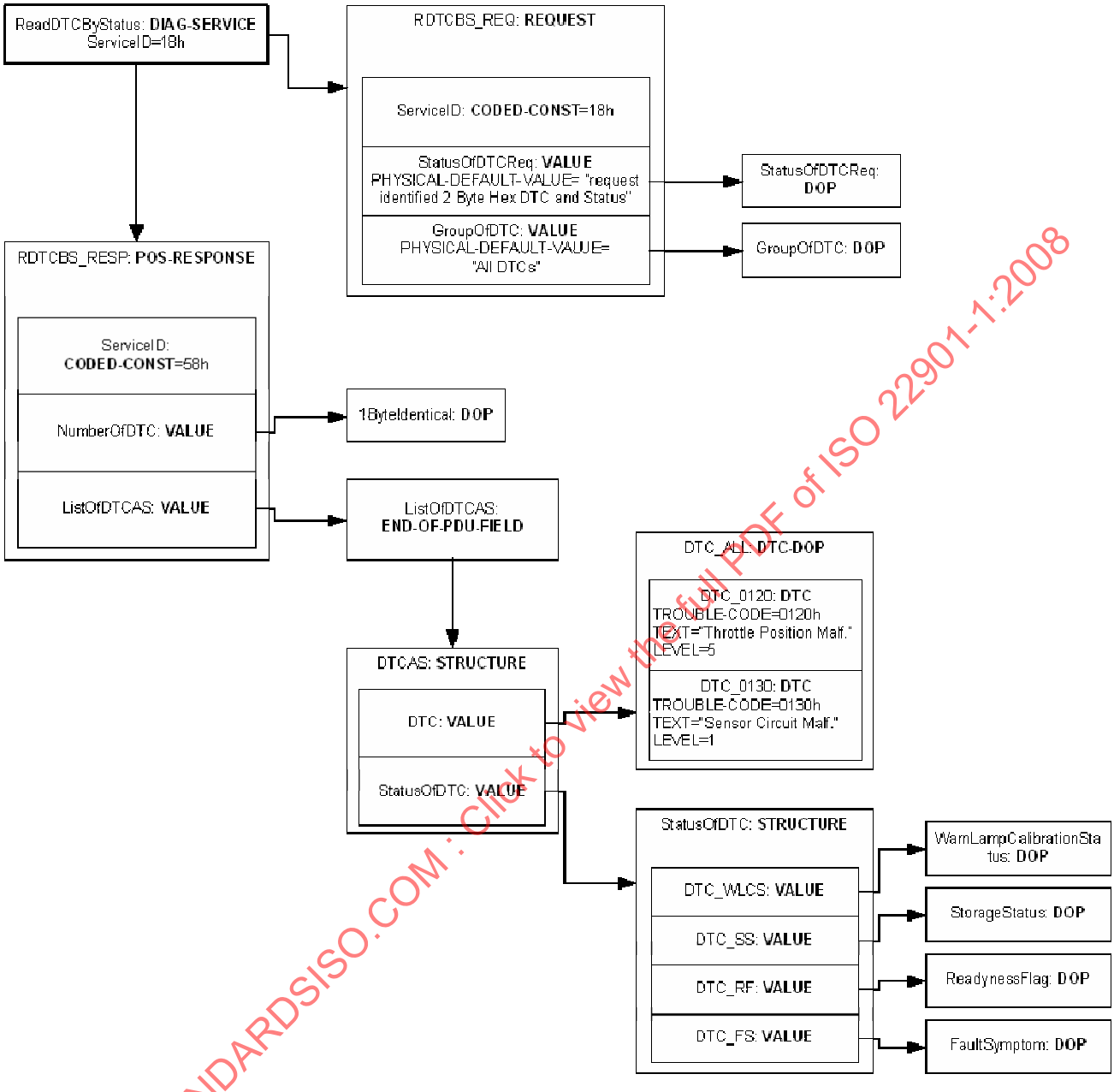


Figure 142 — Objects used by the ReadDTCByStatus service of ISO 14230-3

The service references a request (RDTCBS_REQ) and a positive response (RDTCBS_RESP). The request consists of three parameters: ServiceID, StatusOfDTCReq and GroupOfDTC. The first parameter is of type CODED-CONST and its value is set to 0x18. The next two parameters are of type VALUE with predefined default values. Each of them references one DOP (StatusOfDTCReq and GroupOfDTC), which are used for conversion of the values, given by the user, in to their coded representation.

The response also includes three parameters: ServiceID as a constant value and NumberOfDTC and ListOfDTC as values, which are to be interpreted. Number of DTCs is computed with a DOP that simply passes a one-byte value on. ListOfDTCAS references an END-OF-PDU-FIELD that interprets the received DTCs as a list of entries. An entry is a structure (DTCAS) consisting of two VALUE parameters: DTC and StatusOfDTC. The DTC parameter references the DTC-DOP with the SHORT-NAME “DTC_ALL” that is

responsible for extraction of the DTC. Finally, the structure StatusOfDTC extracts the status of DTC consisting of four values: warning lamp calibration status, storage status, readiness flag and fault symptom.

The names of the objects in the Figure 142 correspond to the SHORT-NAMEs of the objects in the XML instance. Here is an example, how the DIAG-SERVICE can be implemented in the XML:

```
<DIAG-SERVICE ID = "DS_ReadStatusOfDTC" SEMANTIC = "FAULTREAD" ADDRESSING = "PHYSICAL">
  <SHORT-NAME>DS_ReadStatusOfDTC</SHORT-NAME>
  <LONG-NAME>Read Status of DTC</LONG-NAME>
  <AUDIENCE IS-DEVELOPMENT = "true" IS-AFTERMARKET = "true" IS-MANUFACTURING = "true"/>
  <REQUEST-REF ID-REF = "REQ_RSODTC"/>
  <POS-RESPONSE-REFS>
    <POS-RESPONSE-REF ID-REF = "RESP_RSODTC"/>
  </POS-RESPONSE-REFS>
</DIAG-SERVICE>
```

After all DTCs are read from the ECU's memory, it is possible to query the environmental data, stored with the DTCs. The service ReadStatusOfDTC (ServiceId=0x17) is responsible for this task.

Figure 143 — Objects used by the ReadStatusOfDTC service of ISO 14230-3 shows all objects used by this service and relationships between them.

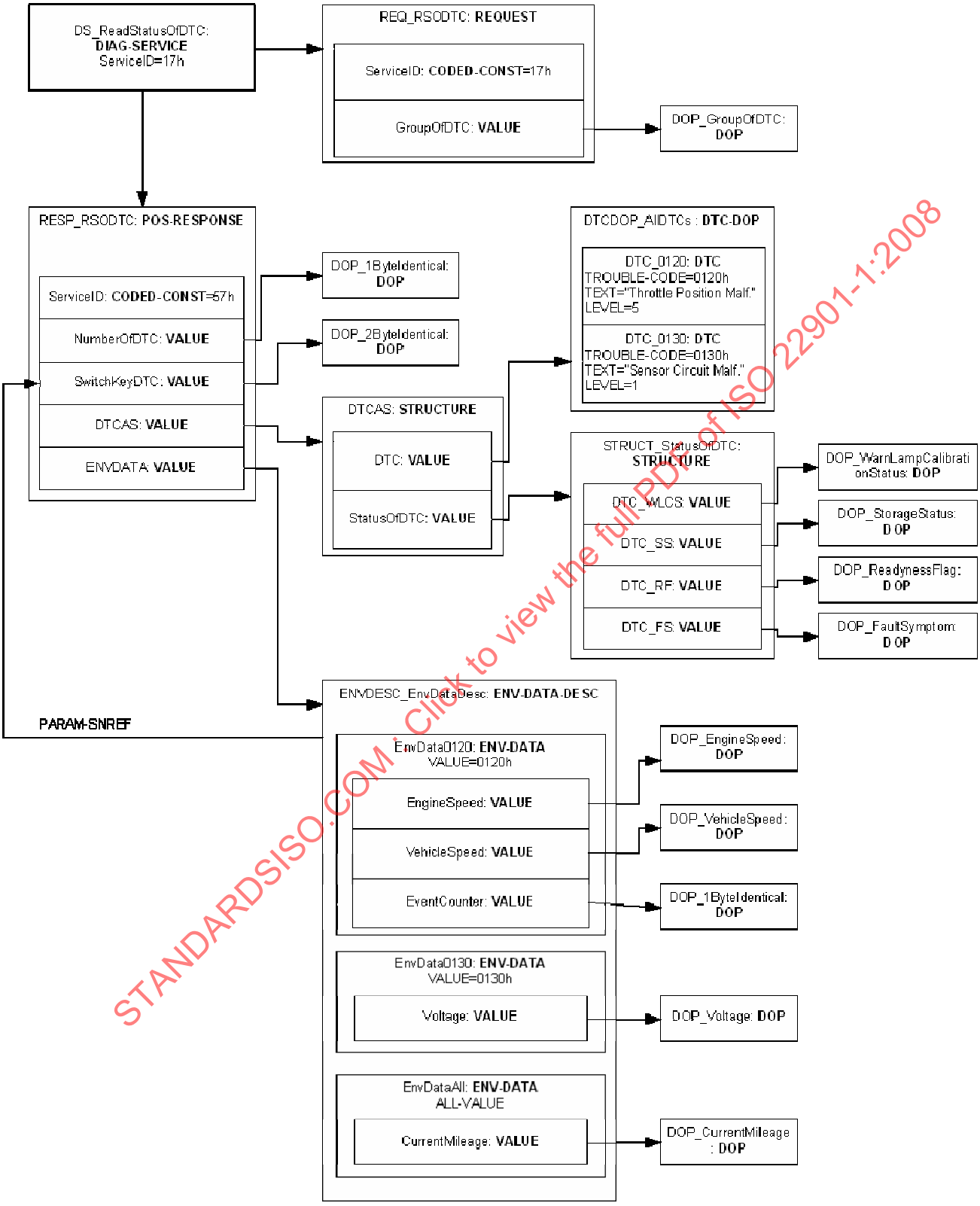


Figure 143 — Objects used by the ReadStatusOfDTC service of ISO 14230-3

The request (RSODTC_REQ), used by this service, contains two parameters: ServiceID and GroupOfDTC. The first one is set to a fixed value of 17h while the user shall specify the second one. The DOP GroupOfDTC converts the user input into the coded value that is put onto the PDU stream.

The response of the service consists of five parameters: ServiceID, NumberOfDTC, DTC, StatusOfDTC and ENVDATA. It is assumed that NumberOfDTC=1 and we only need to interpret a single DTC with its environmental data (this assumption is also recommended praxis in ISO 14230). The definition of the next two parameters (SwitchKeyDTC and DTCAS) occurs in a tricky way. They both start on the same byte position in the PDU. SwitchKeyDTC parameter is a part of the DTCAS parameter and addresses only to the DTC. In contrast to that, DTCAS references the structure DTCAS including the values DTC and StatusOfDTC. Here is an XML code implementing the response:

```
<POS-RESPONSE ID = "RESP_RSODTC">
  <SHORT-NAME>RESP_RSODTC</SHORT-NAME>
  <LONG-NAME>Read Status of DTC Response</LONG-NAME>
  <PARAMS>
    <PARAM xsi:type = "CODED-CONST" SEMANTIC = "SERVICE-ID">
     <SHORT-NAME>ServiceID</SHORT-NAME>
     <LONG-NAME>Read Status of DTC Response</LONG-NAME>
      <BYTE-POSITION>0</BYTE-POSITION>
      <CODED-VALUE>87</CODED-VALUE>
      <DIAG-CODED-TYPE BASE-DATA-TYPE = "A_UINT32" xsi:type = "STANDARD-LENGTH-TYPE">
        <BIT-LENGTH>8</BIT-LENGTH>
      </DIAG-CODED-TYPE>
    </PARAM>
    <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
     <SHORT-NAME>NumberOfDTC</SHORT-NAME>
     <LONG-NAME>number of DTC</LONG-NAME>
     <BYTE-POSITION>1</BYTE-POSITION>
     <DOP-REF ID-REF = "DOP_1ByteIdentical"/>
    </PARAM>
    <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
      <SHORT-NAME>SwitchKeyDTC</SHORT-NAME>
      <LONG-NAME>DTC SwitchKEY</LONG-NAME>
      <BYTE-POSITION>2</BYTE-POSITION>
      <DOP-REF ID-REF = "DOP_2ByteIdentical"/>
    </PARAM>
    <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
      <SHORT-NAME>DTCAS</SHORT-NAME>
      <LONG-NAME>DTC and Status</LONG-NAME>
      <BYTE-POSITION>2</BYTE-POSITION>
      <DOP-REF ID-REF = "STRUCT_DTCAS"/>
    </PARAM>
    <PARAM xsi:type = "VALUE" SEMANTIC = "DATA">
      <SHORT-NAME>ENVDATA</SHORT-NAME>
      <LONG-NAME>Environmental Data</LONG-NAME>
      <BYTE-POSITION>5</BYTE-POSITION>
      <DOP-REF ID-REF = "ENVDESC_EnvDataDesc"/>
    </PARAM>
  </PARAMS>
</POS-RESPONSE>
```

The last parameter of the response references the ENV-DATA-DESC object named “EnvDataDesc”. It references (via SHORT-NAME) the parameter SwitchKeyDTC, which is used as switch key to find the right ENV-DATA object within ENV-DATA-DESC. The latter defines three ENV-DATA objects: EnvData0120, EnvData0130 and EnvDataAll. Each of them represents a structure with the appropriate amount of

parameters. The last one (EnvDataAll) includes a value (CurrentMileage), which is common for all DTCs, i.e. this value is sent by the ECU as a reply to the request by each DTC (on the BYTE-POSITION=0).

The value DTC is extracted from the PDU twice. First time, it is used as a switch key. Another time, it is a part of the structure DTCAS. Within this structure, the parameter DTC is extracted as a complex value including trouble code, text and level. The second extraction can be omitted if the DTC should not be shown on display again (it was already shown after reading DTC by status with the service 18h). In this case the fourth parameter would reference the structure StatusOfDTC directly.

7.4.6 Protocol communication parameter

For the exchange of diagnostic data between clients and ECUs, it is necessary to assign communication parameters (e.g. timing, etc.) to a specific protocol layer. Only one COMPARAM-SPEC could be referenced from a PROTOCOL layer instance. The communication parameter values defined in the COMPARAM-SPEC could be assigned with new values in the PROTOCOL layer. This feature is necessary if the COMPARAM-SPEC is referenced by more than one PROTOCOL layer.

The following usage scenario will help to understand the handling of the communication-parameters (Figure 144 — Diagnostic protocol stack of an exemplary ECU). For that purpose it will be supposed that an exemplary ECU supports the KWP2000 protocol on CAN (ISO 14230-3 on ISO 15765-2) for enhanced diagnostic and also the legislated emission related diagnostics on CAN (ISO 15031-5 on ISO 15765-4). Figure 145 — Simplified Diagnostic layers structure for an exemplary ECU shows both diagnostic protocols with separate PROTOCOL layer instances and each protocol references the same COMPARAM-SPEC.

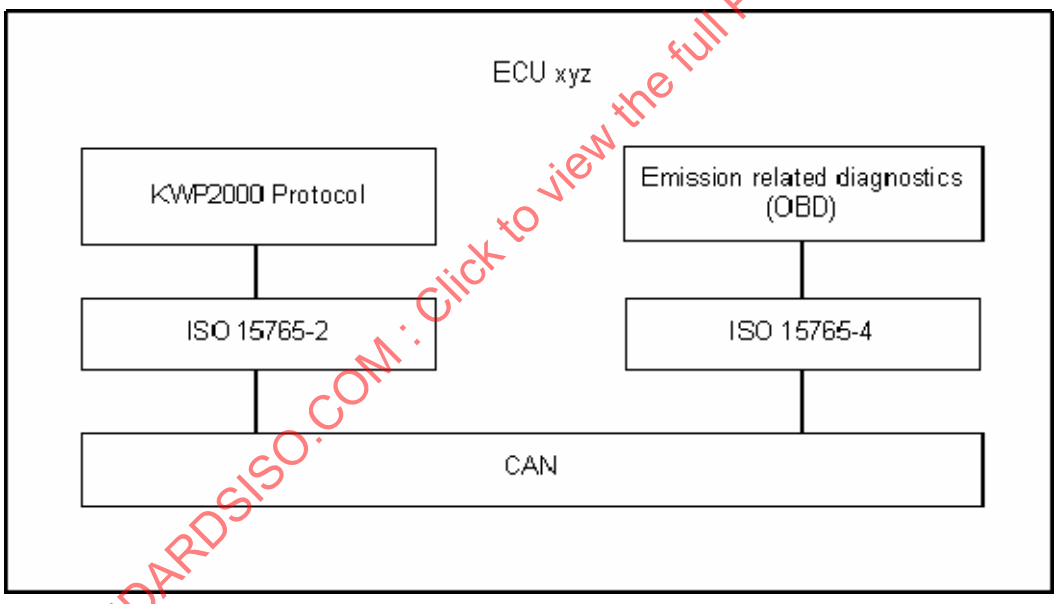


Figure 144 — Diagnostic protocol stack of an exemplary ECU

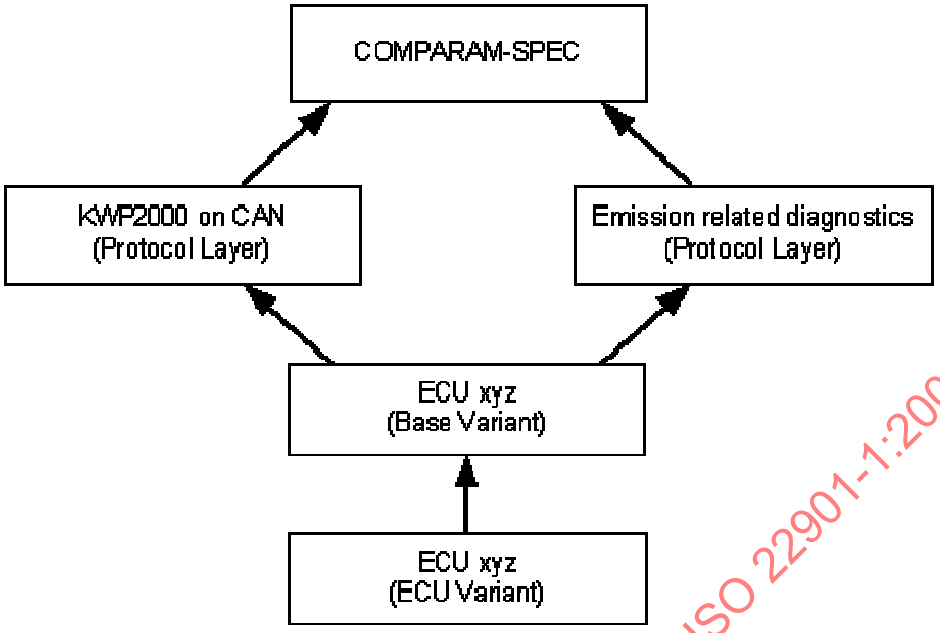


Figure 145 — Simplified Diagnostic layers structure for an exemplary ECU

The common COMPARAM-SPEC instance for both protocols could include the following three communication parameters:

— Address Schema: Select Normal or Extended Addressing;

— CANPaddingHandling: Fill unused bytes in every PDU with a special padding bytes;

— P2CANServerMax: Maximum Time between Request and Response on ECU side.

The COMPARAM-SPEC instance references the COMPARAM-SUBSET instances the parameters are defined in.

```
<COMPARAM-SPEC ID="compKWPonCAN_id123">
    <SHORT-NAME>compKWPonCAN</SHORT-NAME>
    <PROT-STACKS>
        <PROT-STACK ID="compKWPonCAN.PROT-STACK.id1">
            <SHORT-NAME>compKWPonCANprotStack</SHORT-NAME>
            <PDU-PROTOCOL-TYPE>ISO_15031_5_on_ISO_15765_4</PDU-PROTOCOL-TYPE>
            <PHYSICAL-LINK-TYPE>CAN</PHYSICAL-LINK-TYPE>
            <COMPARAM-SUBSET-REFS>
                <!-- APP -->
                <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="compKWPonCANSubset1" ID-
REF="compKWPonCAN.id1"/>
                <!-- TRANS -->
                <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="compKWPonCANSubset2" ID-
REF="compKWPonCAN.id2"/>
            </COMPARAM-SUBSET-REFS>
        </PROT-STACK>
    </PROT-STACKS>
</COMPARAM-SPEC>
```

The default values for these three communication parameters are initialised with valid values for the Emission related diagnostics. According to the ISO OSI layer model, they are defined in several COMPARAM-SUBSET structures.

The ISO 15765-4 Network layer requires the “Normal Addressing” as Address Schema.

```
<COMPARAM-SUBSET ID = "compKWPonCAN.id1" CATEGORY = "APP">
    <SHORT-NAME>compKWPonCAN</SHORT-NAME>
    <COMPARAMS>
        <COMPARAM
            CPTYPE = "STANDARD"
            DISPLAY-LEVEL = "1"
            ID = "compKWPonCAN.AddrSchema"
            CPUSAGE = "ECU-COMM"
            PARAM-CLASS = "COM">
            <SHORT-NAME>AddrSchema</SHORT-NAME>
            <PHYSICAL-DEFAULT-VALUE>normal</PHYSICAL-DEFAULT-VALUE>
            <DATA-OBJECT-PROP-REF ID-REF = "compKWPonCAN.DOP04"/>
        </COMPARAM>
    </COMPARAMS>
    <DATA-OBJECT-PROPS>...</DATA-OBJECT-PROPS>
</COMPARAM-SUBSET>
```

Each diagnostic CAN frame shall contain eight bytes of data. Therefore CANPaddingHandling shall be enabled. The P2CANServerMax timing for legislated diagnostics is 50 ms.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ODX MODEL-VERSION="2.2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="odx.xsd">
     <COMPARAM-SPEC ID="CPS_ISO_15765_3_on_ISO_15765_2">
          <SHORT-NAME>ISO_15765_3_on_ISO_15765_2</SHORT-NAME>
          <LONG-NAME>ISO_15765_3_on_ISO_15765_2</LONG-NAME>
          <DESC>
               <p>ISO UDS On Can</p>
          </DESC>
          <PROT-STACKS>
               <PROT-STACK ID="PS_ISO_15765_3_on_ISO_15765_2_on_ISO_11898_2_DWCAN">
                    <SHORT-NAME>PS_ISO_15765_3_on_ISO_15765_2_on_ISO_11898_2_DWCAN</SHORT-NAME>
                    <LONG-NAME>ISO_15765_3 on ISO_15765_2 on ISO_11898_2_DWCAN</LONG-NAME>
                    <DESC>
                         <p>protocol ISO_15765_3_on_ISO_15765_2, on physical layer ISO_11898_2_DWCAN</p>
                    </DESC>
                    <PDU-PROTOCOL-TYPE>ISO_15765_3_on_ISO_15765_2</PDU-PROTOCOL-TYPE>
                    <PHYSICAL-LINK-TYPE>ISO_11898_2_DWCAN</PHYSICAL-LINK-TYPE>
                    <COMPARAM-SUBSET-REFS>
                         <!-- APP -->
                         <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_15765_3" ID-REF="ISO_15765_3"/>
                         <!-- TRANS -->
                         <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_15765_2" ID-REF="ISO_15765_2"/>
                         <!-- PHYS -->
                         <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_11898_2_DWCAN"
                                   ID-REF="ISO_11898_2_DWCAN"/>
                    </COMPARAM-SUBSET-REFS>
               </PROT-STACK>
               <PROT-STACK ID="PS_ISO_15765_3_on_ISO_15765_2_on_ISO_11898_3_DWFTCAN">
                    <SHORT-NAME>PS_ISO_15765_3_on_ISO_15765_2_on_ISO_11898_3_DWFTCAN</SHORT-NAME>
                    <LONG-NAME>ISO_15765_3 on ISO_15765_2 on ISO_11898_3_DWFTCAN</LONG-NAME>
                    <DESC>
                         <p>protocol ISO_15765_3_on_ISO_15765_2, on physical layer ISO_11898_3_DWFTCAN</p>
                    </DESC>
                    <PDU-PROTOCOL-TYPE>ISO_15765_3_on_ISO_15765_2</PDU-PROTOCOL-TYPE>
```

```
                <PHYSICAL-LINK-TYPE>ISO_11898_3_DWFTCAN</PHYSICAL-LINK-TYPE>
                <COMPARAM-SUBSET-REFS>
                    <!-- APP -->
                    <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_15765_3" ID-REF="ISO_15765_3"/>
                    <!-- TRANS -->
                    <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_15765_2" ID-REF="ISO_15765_2"/>
                    <!-- PHYS -->
                    <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_11898_3_DWFTCAN"
                                         ID-REF="ISO_11898_3_DWFTCAN"/>
                </COMPARAM-SUBSET-REFS>
            </PROT-STACK>
            <PROT-STACK ID="PS_ISO_15765_3_on_ISO_15765_2_on_SAE_J2411_SWCAN">
                <SHORT-NAME>PS_ISO_15765_3_on_ISO_15765_2_on_SAE_J2411_SWCAN</SHORT-NAME>
                <LONG-NAME>PS_ISO_15765_3 on ISO_15765_2 on SAE_J2411_SWCAN</LONG-NAME>
                <DESC>
                    <p>protocol ISO_15765_3_on_ISO_15765_2, on physical layer SAE_J2411_SWCAN</p>
                </DESC>
                <PDU-PROTOCOL-TYPE>ISO_15765_3_on_ISO_15765_2</PDU-PROTOCOL-TYPE>
                <PHYSICAL-LINK-TYPE>SAE_J2411_SWCAN</PHYSICAL-LINK-TYPE>
                <COMPARAM-SUBSET-REFS>
                    <!-- APP -->
                    <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_15765_3" ID-REF="ISO_15765_3"/>
                    <!-- TRANS -->
                    <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="ISO_15765_2" ID-REF="ISO_15765_2"/>
                    <!-- PHYS -->
                    <COMPARAM-SUBSET-REF DOCTYPE="COMPARAM-SUBSET" DOCREF="SAE_J2411_SWCAN"
                                         ID-REF="SAE_J2411_SWCAN"/>
                </COMPARAM-SUBSET-REFS>
            </PROT-STACK>
        </PROT-STACKS>
    </COMPARAM-SPEC>
</ODX>
```

The protocol layer for emission related diagnostics (OBDonCAN) only needs to reference this COMPARAM-SPEC without any changes.

```
<PROTOCOL ID="protOBDonCAN">
  <SHORT-NAME>protOBDonCAN</SHORT-NAME>
  ...
  <COMPARAM-SPEC-REF ID-REF="compKWPonCAN.id123" DOCREF="compKWPonCAN"
   DOCTYPE="COMPARAM-SPEC"/>
  ...
</PROTOCOL>
```

For enhanced diagnostics (ISO 14230) a different configuration of the communication parameters is be required (extended addressing and no padding of unused bytes in CAN frames). This makes it necessary to override the different communication parameters in the PROTOCOL layer instance.

```
<PROTOCOL ID="protKWPonCAN">
    <SHORT-NAME>protKWPonCAN</SHORT-NAME>
    <COMPARAM-REFS>
        <COMPARAM-REF ID-REF="compKWPonCAN.AddrSchema" DOCREF="compKWPonCAN" DOCTYPE="COMPARAM-SPEC">
            <SIMPLE-VALUE>extended</SIMPLE-VALUE>
        </COMPARAM-REF>
        <COMPARAM-REF ID-REF="compKWPonCAN.CANPaddingHandling" DOCREF="compKWPonCAN" DOCTYPE="COMPARAM-SPEC">
            <SIMPLE-VALUE>disabled</SIMPLE-VALUE>
```

```
        </COMPARAM-REF>
    </COMPARAM-REFS>
    <COMPARAM-SPEC-REF ID-REF="compKWPonCAN.id123" DOCREF="compKWPonCAN" DOCTYPE="COMPARAM-SPEC"/>
    <PROT-STACK-SNREF SHORT-NAME="_1"/>
    <PARENT-REFS>
        <PARENT-REF ID-REF="_P1" xsi:type="ECU-SHARED-DATA-REF"/>
    </PARENT-REFS>
</PROTOCOL>
```

Further on an additional requirement could make it necessary to change the timing of a special DIAG-SERVICE in the ISO 14230 protocol layer.

EXAMPLE The execution of the service “ClearDiagnosticTroubleCodes” needs 100 ms instead of 50 ms defined in the referenced COMPARAM-SPEC because of the time for the EEPROM access. To solve this problem, the timing parameter P2CANServerMax is set to the new value of 100 ms.

The next example contains all changes for the specific ISO 14230 PROTOCOL layer instance.

```
<PROTOCOL ID="protKWPonCAN.id235">
    <SHORT-NAME>protKWPonCAN</SHORT-NAME>
    <DIAG-COMMS>
        <DIAG-SERVICE ID="ClearDiagTroubleCodes.id456">
            <SHORT-NAME>ClearDiagTroubleCodes</SHORT-NAME>
            <COMPARAM-REFS>
                <COMPARAM-REF ID-REF="compKWPonCAN.P2CANServerMax" DOCREF="compKWPonCAN" DOCTYPE="COMPARAM-SPEC">
                    <SIMPLE-VALUE>100</SIMPLE-VALUE>
                </COMPARAM-REF>
            </COMPARAM-REFS>
        </DIAG-SERVICE>
    </DIAG-COMMS>
    <COMPARAM-REFS>
        <COMPARAM-REF ID-REF="compKWPonCAN.AddrSchema" DOCREF="compKWPonCAN" DOCTYPE="COMPARAM-SPEC">
            <SIMPLE-VALUE>extended</SIMPLE-VALUE>
        </COMPARAM-REF>
        <COMPARAM-REF ID-REF="compKWPonCAN.CANPaddingHandling" DOCREF="compKWPonCAN" DOCTYPE="COMPARAM-SPEC">
            <SIMPLE-VALUE>disabled</SIMPLE-VALUE>
        </COMPARAM-REF>
    </COMPARAM-REFS>
    <COMPARAM-SPEC-REF ID-REF="compKWPonCAN.id123" DOCREF="compKWPonCAN" DOCTYPE="COMPARAM-SPEC"/>
</PROTOCOL>
```

7.4.7 Dynamic diagnostic response

7.4.7.1 Example of DYNAMIC-LENGTH-FIELD

Figure 146 — DYNAMIC-LENGTH-FIELD in readDTCByStatus response shows the overview of objects used by the response of the service readDTCByStatus (0x18) in ISO 14230.

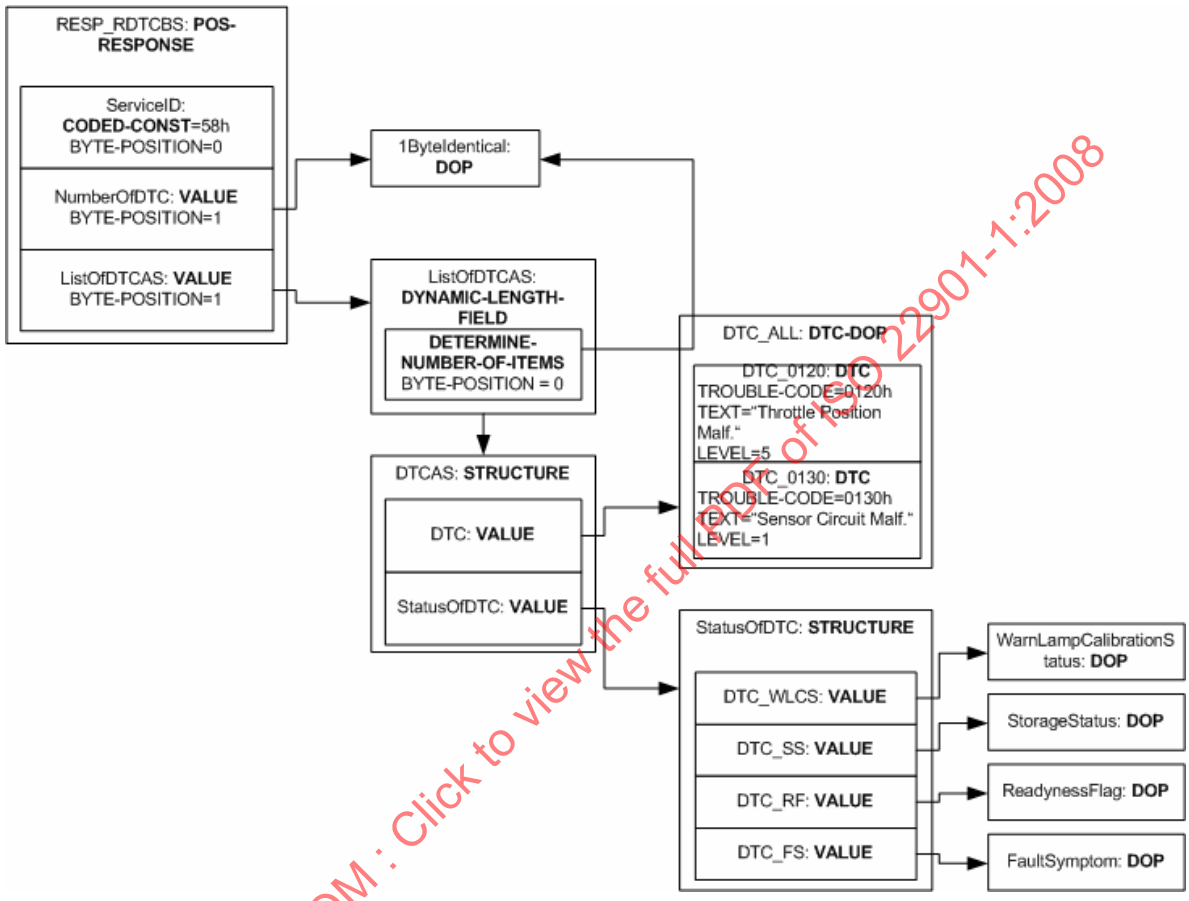


Figure 146 — DYNAMIC-LENGTH-FIELD in readDTCByStatus response

DYNAMIC-LENGTH-FIELD is used there to interpret the list of diagnostic trouble codes (DTCs) received from the ECU. It references the structure DTCAS, which is taken for interpretation of a single DTC with the associated status. The parameter NumberOfDTC specifies how often this structure is repeated. Therefore, the member DETERMINE-NUMBER-OF-ITEMS refers to this parameter by the BYTE-POSITION and to the DOP 1ByteIdentical used to extract the number of DTCs out of the PDU.

In XML language, the DYNAMIC-LENGTH-FIELD looks like this:

```
<DYNAMIC-LENGTH-FIELD ID = "ListOfDTCAS">
    <SHORT-NAME>ListOfDTCAS</SHORT-NAME>
    <BASIC-STRUCTURE-REF ID-REF = "STRUCT_DTCAS"/>
    <OFFSET>1</OFFSET>
    <DETERMINE-NUMBER-OF-ITEMS>
        <BYTE-POSITION>0</BYTE-POSITION>
        <DATA-OBJECT-PROP-REF ID-REF = "DOP_1ByteIdentical"/>
    </DETERMINE-NUMBER-OF-ITEMS>
</DYNAMIC-LENGTH-FIELD>
```

In this example, the list ListOfDTCAS reaches the end of the PDU, so we actually do not need the number of DTCs and the class END-OF-PDU-FIELD can be used instead of DYNAMIC-LENGTH-FIELD. This alternative implementation can be found in 7.4.5.

7.4.7.2 Example of DYNAMIC-ENDMARKER-FIELD

In the following example, a DYNAMIC-ENDMARKER-FIELD is defined, which contains a field of elements with the structure defined by the object with ID="STRUCT-3ByteWrapper". Before each iteration, the DOP with ID="DOP-2ByteIdentical" extracts a number from the PDU (a 2-byte value) at the current position which is compared with TERMINATION-VALUE. The iteration is stopped when the value 65535 (FFFFh) is reached.

```
<DYNAMIC-ENDMARKER-FIELD ID = "DEMFIELD_3ByteValuesUntilFFFF">
    <SHORT-NAME>3ByteValuesUntilFFFF</SHORT-NAME>
    <BASIC-STRUCTURE-REF ID-REF = "STRUCT_3ByteWrapper"/>
    <DATA-OBJECT-PROP-REF ID-REF = "DOP_2ByteIdentical">
        <TERMINATION-VALUE>65535</TERMINATION-VALUE>
    </DATA-OBJECT-PROP-REF>
</DYNAMIC-ENDMARKER-FIELD>
```

Assuming, the tester receives the following byte sequence:

0x75 0x45 0x32 0xFF 0xFF 0xEE 0x1A 0xFF 0xFF

Figure 147 — DYNAMIC-ENDMARKER-FIELD in a response shows the following object structure of the response.

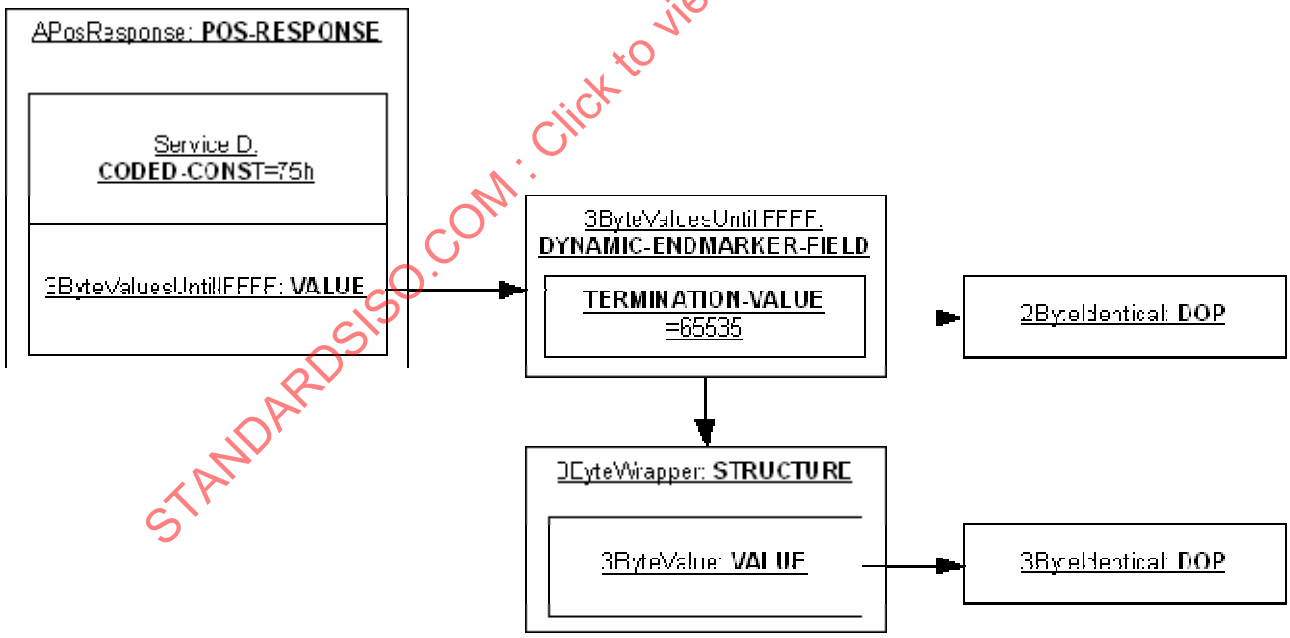


Figure 147 — DYNAMIC-ENDMARKER-FIELD in a response

We get these two 3-byte values: 0x4532FF and 0xFFEE1A. The first byte of the PDU is the service identifier (0x75). The value 0x4532 is then extracted from the PDU using the DOP 2ByteIdentical. Since the extracted value does not equal 65535 (0xFFFF), the structure 3ByteWrapper is applied. It extracts a 3-byte value (0x4532FF) using the DOP 3ByteIdentical. The DOP 2ByteIdentical is then applied again. 0xFFEE does not match the termination value, so the structure 3ByteWrapper extracts a 3-byte value (0xFFEE1A) for the

second time. Finally, the DOP 2ByteIdentical extracts 0xFFFF and the iteration stops because the extracted value matches the given termination value. 0x4532FF and 0xFFEE1A are then presented to the user.

The termination value 0xFFFF can be found twice in the PDU above. Only the second occurrence of this value is accepted as termination value, because the first occurrence can never be extracted by the DOP 2ByteIdentical due to the unfavourable byte arrangement.

7.4.7.3 Example of END-OF-PDU-FIELD

See implementation of the response RDTCBS_RESP in 7.4.5.

7.4.7.4 Example of MUX

In ISO 14230, the service ReadEcuIdentification (0x1A) is used to read data for ECU identification purposes. By means of this service it is possible for example to request the VIN (Vehicle Identification Number) or the programming date of the software. In the request the parameter identificationOption defines the kind of the requested information. Depending on it, different DOPs are used for the interpretation of the data received from the ECU. Figure 148 — MUX in ReadEcuIdentification response shows how this service can be implemented in ODX.

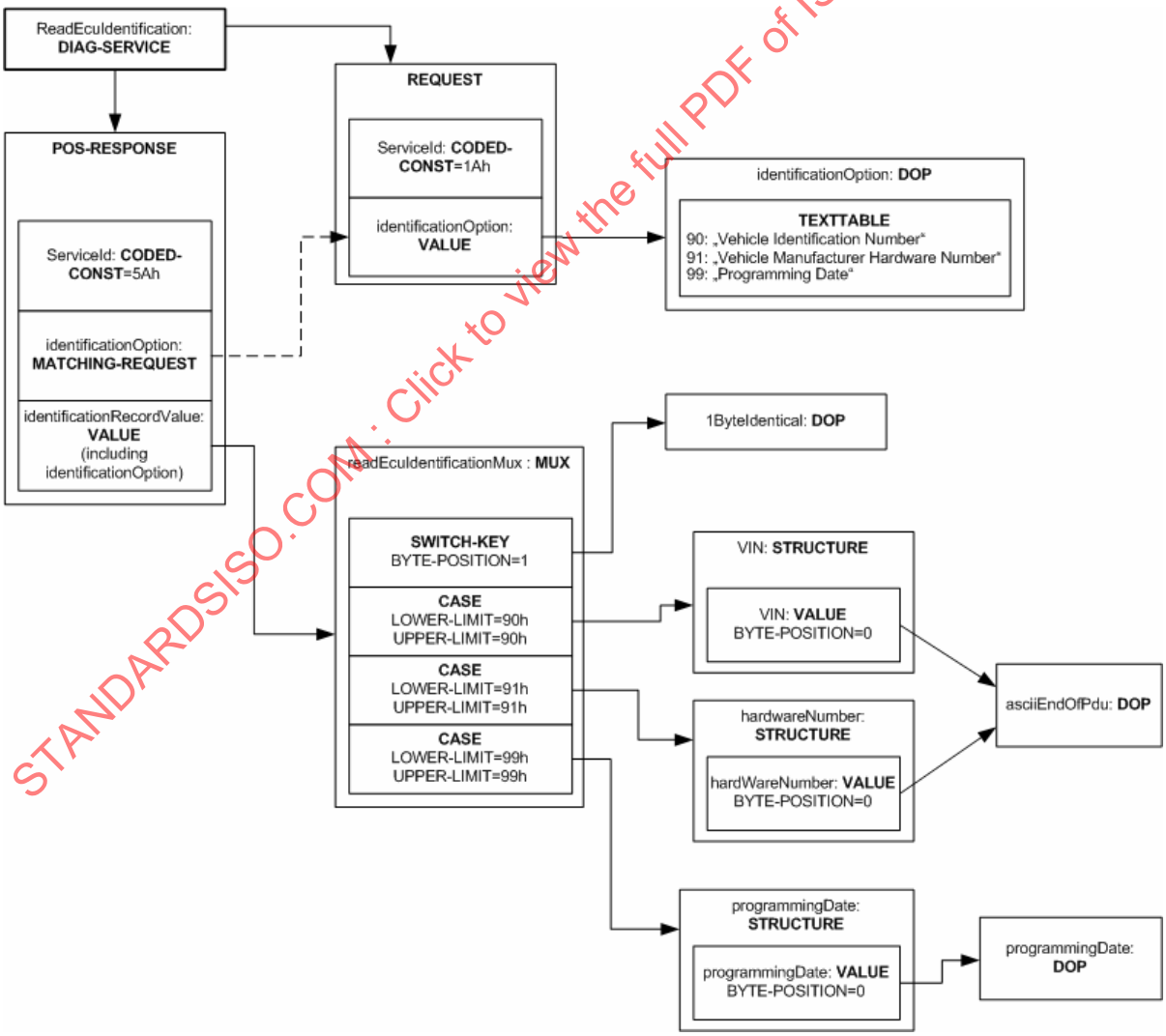


Figure 148 — MUX in ReadEcuIdentification response

The request of the service contains two parameters: ServiceId and identificationOption. In the response, the value of identificationOption is repeated after the parameter posRespServiceId (ServiceId + 0x40). After that, the requested data follow, which is interpreted by the MUX.

The following example code implements the MUX in XML:

```
<MUX ID = "MUX_ReadEcuIdentificationMux">
    <SHORT-NAME>MUX_ReadEcuIdentificationMux</SHORT-NAME>
    <BYTE-POSITION>1</BYTE-POSITION>
    <SWITCH-KEY>
        <BYTE-POSITION>0</BYTE-POSITION>
        <DATA-OBJECT-PROP-REF ID-REF = "DOP_1ByteIdentical"/>
    </SWITCH-KEY>
    <CASES>
        <CASE>
            <SHORT-NAME>from90to90hex</SHORT-NAME>
            <STRUCTURE-REF ID-REF = "STRUCT_VIN"/>
            <LOWER-LIMIT>144</LOWER-LIMIT>
            <UPPER-LIMIT>144</UPPER-LIMIT>
        </CASE>
        <CASE>
            <SHORT-NAME>from91to91hex</SHORT-NAME>
            <STRUCTURE-REF ID-REF = "STRUCT_hardwareNumber"/>
            <LOWER-LIMIT>145</LOWER-LIMIT>
            <UPPER-LIMIT>145</UPPER-LIMIT>
        </CASE>
        <CASE>
            <SHORT-NAME>from99to99hex</SHORT-NAME>
            <STRUCTURE-REF ID-REF = "STRUCT_programmingDate"/>
            <LOWER-LIMIT>153</LOWER-LIMIT>
            <UPPER-LIMIT>153</UPPER-LIMIT>
        </CASE>
    </CASES>
</MUX>
```

STRUCT_VIN is used for interpretation of the parameter, which references this MUX, if the byte before this parameter equals 0x90. STRUCT_hardwareNumber is used if this byte has a value of 0x91 and finally, programmingDate is relevant in the case of 0x99.

7.4.8 Variable length parameter

This usage scenario demonstrates the use of the LENGTH-KEY parameter. There are some cases, in which one parameter determines the length of the other one. E.g. the readMemoryByAddress service in ISO 14229-1 uses a parameter called “addressAndLengthFormatIdentifier” to define the length of the following two parameters called “memoryAddress” and “memorySize”.

Figure 149 — LENGTH-KEY parameter in request of readMemoryByAddress service shows how the request of this service could be implemented in ODX.

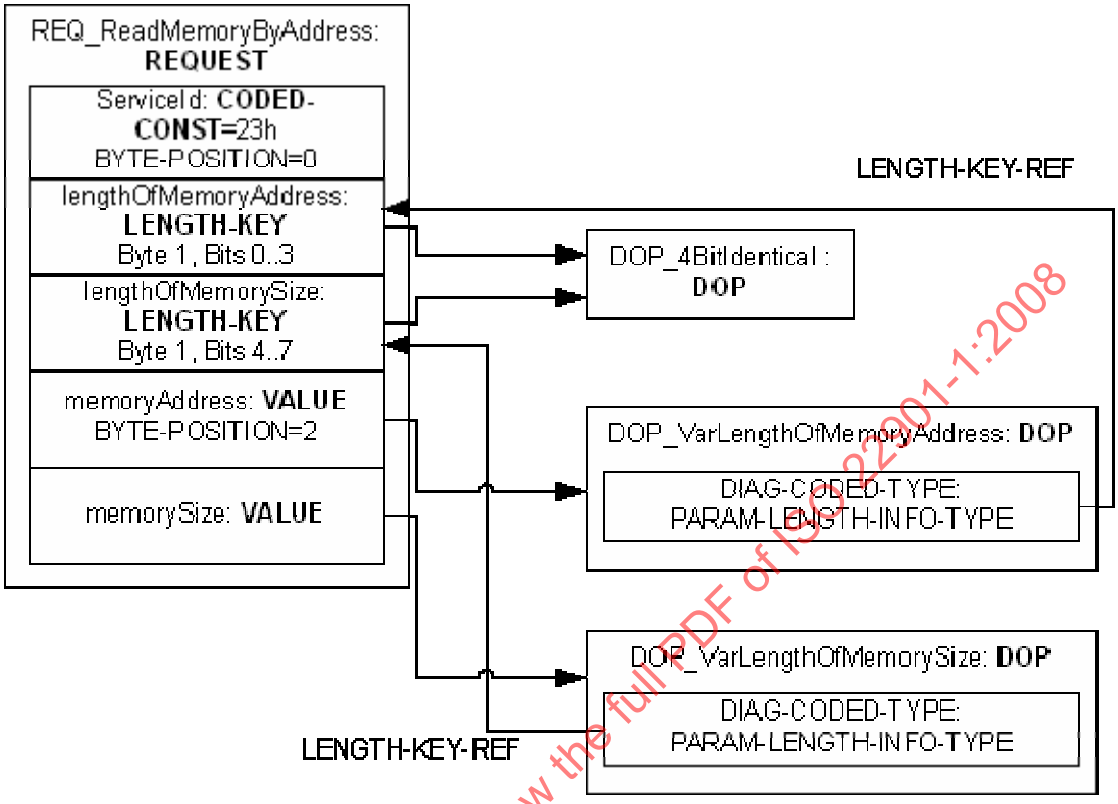


Figure 149 — LENGTH-KEY parameter in request of readMemoryByAddress service

The length of the parameter "memoryAddress" is given by the lower nibble of byte 1 (parameter "lengthOfMemoryAddress"). Therefore the DATA-OBJECT-PROP referenced by this parameter uses PARAM-LENGTH-INFO-TYPE as DIAG-CODED-TYPE. The latter refers to the parameter "lengthOfMemoryAddress" defining the length of the parameter "memoryAddress". The length of the parameter "memorySize" is defined in the similar way. While the position of the first three parameters can be declared in the ODX instance, the position of the last parameter is unknown until runtime. For that reason the BYTE-POSITION of the last parameter is not defined there.

7.4.9 Functional addressing

7.4.9.1 Overview

This subclause (7.4.9) describes how data for functional addressing communication scenarios can be defined using ODX in a mostly protocol neutral manner.

7.4.9.2 Communication on the vehicle bus

Functional addressing depicts a communication scenario, where a tester sends out a request on a dedicated functional address and multiple ECUs respond to such a request. It closely resembles broadcast communication in common network technology. In opposition to functional addressing, the term physical addressing is used, if the tester sends a request to an address uniquely identifying one ECU on the bus.

The communication scenario on the vehicle bus is shown in Figure 150 — Functional addressing communication on the vehicle bus. A tester sends a functional request on the bus to which any number of ECUs on the bus may be responding. In the scenario shown one ECU does not respond to the functional request.

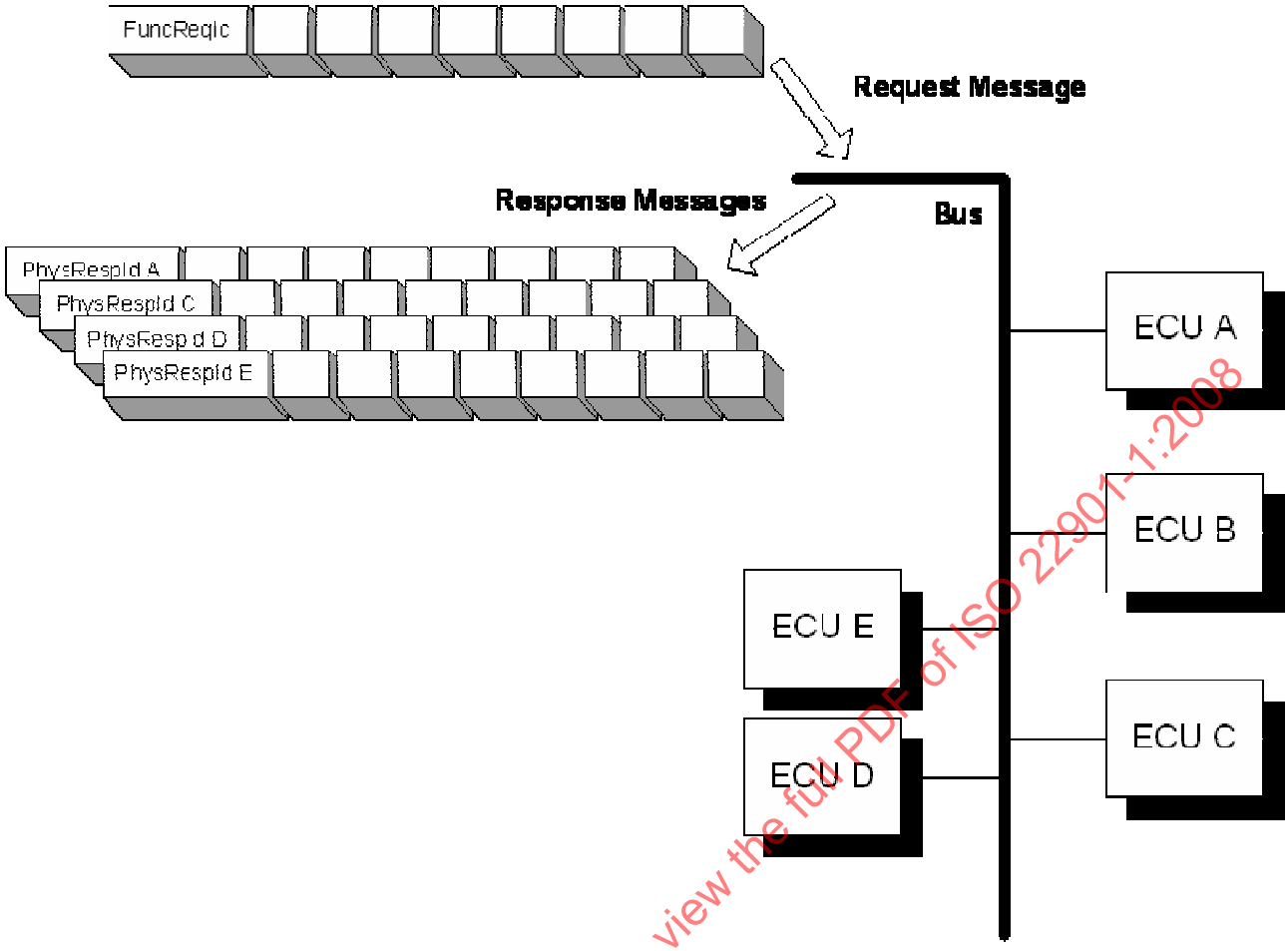


Figure 150 — Functional addressing communication on the vehicle bus

In general, at the time of sending out the request, the tester does not know how many responses it will receive. This may have multiple reasons:

— an ECU is broken and does not respond even though it should;

— an ECU listed in the vehicle information is not really built into the particular vehicle under test.

Consequently, the D-PDU API and its protocol drivers do not know when the communication primitive initiated through the request message terminates. It is assumed in the following that termination of a functionally addressed communication primitive is always detected on a time-out. In these cases, the time-out does not indicate an error or exceptional situation and shall not be handled as such by either the D-PDU API or the D-server.

Another important observation is that not all ECU's responses may have the exactly same structure. Even when by definition they should, incorrect implementations of the diagnostic protocol in ECUs shall also be considered.

Furthermore, communication scenarios exist, where the ECU sends a multi-part response, that is multiple individual response messages to one single request, where even the individual responses may not be structured identically. This makes for the most complicated functional addressing scenario, as the multiple responses of multiple ECUs shall be assigned to the right physical ECUs correctly.

It also needs to be considered that ECUs do not necessarily respond to the functional request using a physical response identifier. Communication scenarios exist, where the ECU will respond using a functional response identifier.

7.4.9.3 Data structures in ODX to define functional addressing

To define functional addressing communication in ODX, the following aspects need to be considered and will be explained in detail:

a) definition of the functional address for the request message;

b) definition of the list of response identifiers for all possibly responding ECUs;

c) definition of the functional DIAG-SERVICE(s) within a FUNCTIONAL-GROUP;

d) definition of possibly overridden RESPONSE messages for some of the responding ECUs.

The definition of data for functional addressing communication scenarios shall permit a D-server to open a logical link on a FUNCTIONAL-GROUP and communicate functionally without having to consider any BASE-VARIANT data. This is important for e.g. the OBD use case. However, if BASE-VARIANT data is available, functional communication should also be able to use this data.

The most important element for a functional addressing scenario is the FUNCTIONAL-GROUP. Only DIAG-SERVICES contained in a functional group and with addressing modes FUNCTIONAL or FUNCTIONAL-OR-PHYSICAL can be executed functionally within a D-server. For this, the application shall open a logical link on such a FUNCTIONAL-GROUP.

The functional request address needs to be supported by the protocol. That is, the FUNCTIONAL-GROUP shall inherit from a PROTOCOL layer and within this PROTOCOL layer's COMPARAM-SPEC one or multiple communication parameters for the functional request address need to be defined. The value of this or these communication parameter(s) can then be re-defined on the level of the FUNCTIONAL-GROUP.

A similar statement holds true for the list of response identifiers or response addresses for the possibly responding ECUs. Here, too, the COMPARAM-SPEC of the protocol layer needs to support a COMPLEX-COMPARAM of CPTYPE UNIQUE_ID to set up this address table. The content of the COMPLEX-COMPARAM can then again be overridden on the FUNCTIONAL-GROUP level.

The constraints listed below hold on the setup of this COMPLEX-COMPARAM.

— Every FUNCTIONAL-GROUP needs to override the value of the COMPARAM named "CP_UniqueRespIdTable".

— The ALLOW-MULTIPLE-VALUES attribute of the CP_UniqueRespIdTable shall be set to "true".

— The general content of this COMPLEX-COMPARAM shall be filled in a protocol-specific manner. The details for standardized protocols can be found in the D-PDU API specification. For non-standardized protocols this needs to be defined amongst the users of this protocol.

— This COMPLEX-COMPARAM always shall contain a COMPARAM with SHORT-NAME CP_ECULayerShortName. This communication parameter is needed by the D-server to set up the correct Access Keys for every received response. The CPUSAGE of this communication parameter shall be set to APPLICATION.

— The PARAM-CLASS of the COMPLEX-COMPARAM and all contained COMPARAMs shall be set to UNIQUE_ID.

To enable functional addressing in the D-PDU API, the defined communication parameter CP_AddressMode shall be set to functional addressing on the FUNCTIONAL-GROUP level and to physical addressing on any lower layer. See Figure 151 — Functional addressing setup of ODX data.

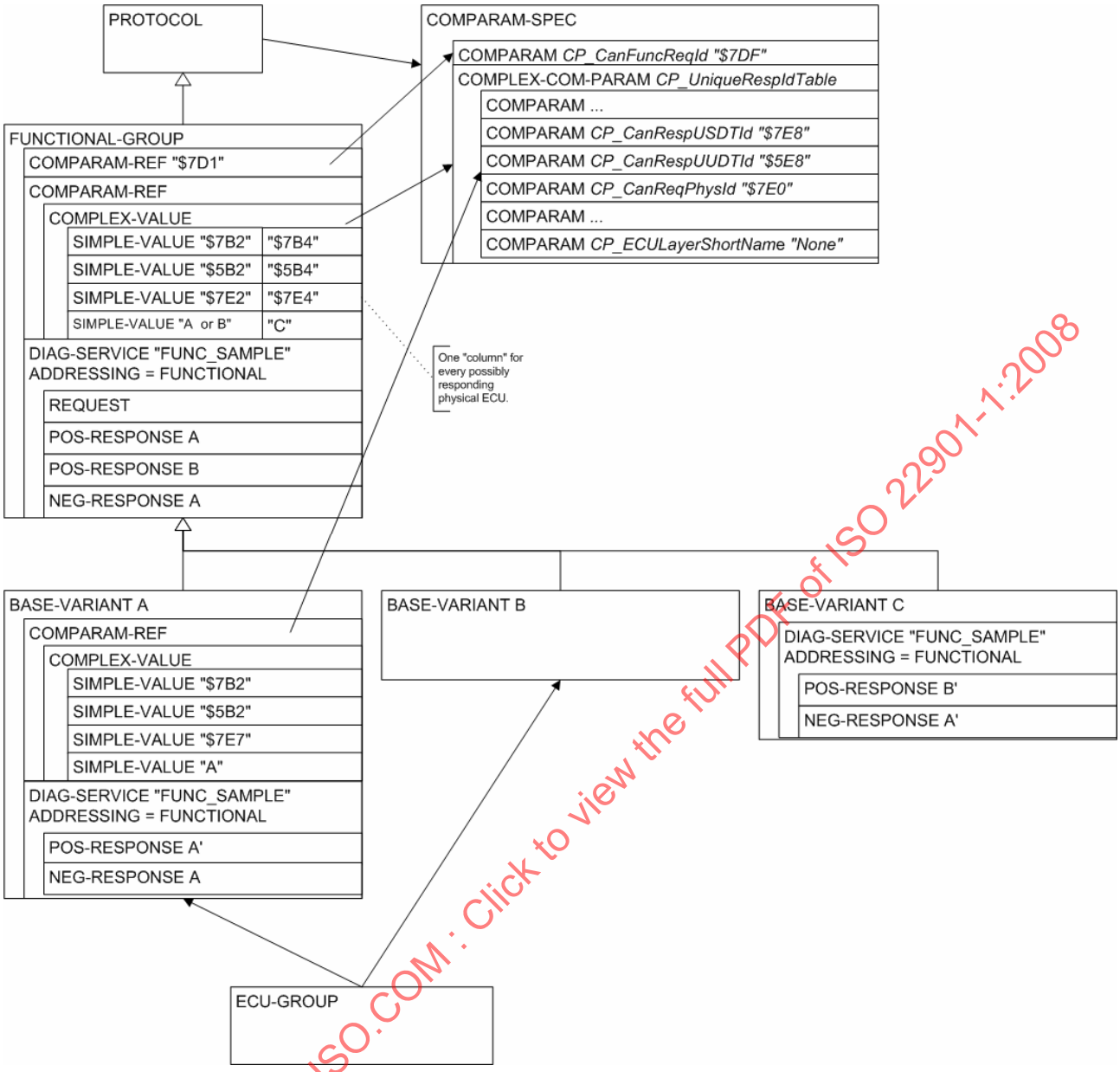


Figure 151 — Functional addressing setup of ODX data

Overridden RESPONSE messages are defined on the level of BASE-VARIANTs in the regular manner: A DIAG-SERVICE with the same SHORT-NAME as on the FUNCTIONAL-GROUP layer is introduced. A new RESPONSE is referenced by this overridden service. For overridden response messages the relevance of ECU-GROUPs needs to be analyzed. In case multiple BASE-VARIANTs belonging to the same ECU-GROUP are also part of the same FUNCTIONAL-GROUP, consideration of overridden RESPONSE messages of such BASE-VARIANTs requires a previous BASE-VARIANT identification step.

The overall composition of data structures to set up functional communication scenarios is depicted in Figure 151, which shows the most complex setup. Easier setups without involvement of ECU-GROUPs or even without involvement of BASE-VARIANTs also exist. The shown setup has a FUNCTIONAL-GROUP holding a *CP_UniqueRespIdTable* with entries for 2 possibly responding ECUs. Three BASE-VARIANTs inherit from the FUNCTIONAL-GROUP. BASE-VARIANT A overrides the complex communication parameter value for *CP_UniqueRespIdTable* and also overrides the functional service. BASE-VARIANT B neither overrides the communication parameter nor the functional service. BASE-VARIANT C overrides the service only. BASE-VARIANTs A and B reside in an ECU-GROUP. Consequently, only one of the two can actually be built into any given vehicle.

7.4.9.4 Sequence of events for functional addressing

7.4.9.4.1 A description of a sequence of events for interpreting the ODX data structures correctly in a functional addressing communication scenario is given below.

a) Application opens a logical link on FUNCTIONAL-GROUP level via D-server.

b) Application selects DIAG-SERVICE with ADDRESSING = FUNCTIONAL | FUNCTIONAL-OR-PHYSICAL and executes it.

c) D-server composes the functional request.

1) It sets up a runtime RespId Table, including all the necessary information to map responses to their physical source ECUs.

i) It takes the COMPLEX-COMPARAM *CP_UniqueRespIdTable* with the values as defined on the FUNCTIONAL-GROUP level.
IMPORTANT — The communication parameter *CP_ECULayerShortName* shall be used only within the D-server. It shall not be sent to the D-PDU API.

ii) For every BASE-VARIANT that is not part of an ECU-GROUP, add the overridden CP_UniqueRespIdTable values to the one created in step i) above. If entries of CP_ECULayerShortName are duplicated, the entry found on BASE-VARIANT level has higher priority than the one on FUNCTIONAL-GROUP level. The entries found on BASE-VARIANT level shall always be added before the ones on FUNCTIONAL-GROUP level. (With this setup, the D-PDU API can go through the table in the right order of priorities.)

iii) For every BASE-VARIANT that is part of an ECU-GROUP but has been identified through a base variant identification process, add the overridden CP_UniqueRespIdTable of the FUNCTIONAL-GROUP values to the one created in step ii) above. If entries are duplicated, the entry found on BASE-VARIANT level has higher priority than the one on FUNCTIONAL-GROUP level.

iv) For every BASE-VARIANT that is part of an ECU-GROUP and has not yet been identified through a base variant identification process, ignore any possibly overridden CP_UniqueRespIdTable values.

v) For every entry in the resulting runtime RespId Table, add a unique identifier. This identifier will be returned by the D-PDU API, if a response was mapped to this entry in the table. (This unambiguously identifies the source of the response message.)

2) It sets up the expected response table (table with acceptance masks for every possible received response message) information for every response that can possibly be received.

i) Acceptance mask information is set up per communication primitive, but for the complete functional group, i.e. POS- and NEG-RESPONSE data from FUNCTIONAL-GROUP level and BASE-VARIANT level are combined into one expected response table.

ii) In case a FUNCTIONAL-GROUP has no inheriting BASE-VARIANTs, the possible physical sources are determined through the CP_UniqueRespIdTable communication parameter value. The expected response table (the acceptance masks) are set up using all POS-RESPONSEs and NEG-RESPONSEs as listed for the DIAG-SERVICE on the FUNCTIONAL-GROUP level.

iii) In case a FUNCTIONAL-GROUP has inheriting BASE-VARIANTs, the possible physical sources are the union of entries in the CP_UniqueRespIdTable and the inheriting BASE-VARIANTs.

I) For every entry in the *CP_UniqueRespIdTable* that does not match an inherited BASE-VARIANT with respect to the value of *CP_UniqueRespIdTable*, add the POS-RESPONSE and NEG-RESPONSE acceptance mask data to the expected response table.

II) If the entry in the CP_UniqueRespIdTable does match with an inherited BASE-VARIANT and that BASE-VARIANT does not reside in an ECU-GROUP, add the BASE-VARIANT's overridden POS-RESPONSEs and NEG-RESPONSEs to the expected response table for the DIAG-SERVICE.

III) If the entry in the CP_UniqueRespIdTable does match with an inherited BASE-VARIANT and that BASE-VARIANT does reside in an ECU-GROUP but the BASE-VARIANT has been identified through a base variant identification process, follow the guidelines from the previous case, i.e. add the BASE-VARIANT's overridden POS-RESPONSEs and NEG-RESPONSEs to the expected response table for the DIAG-SERVICE.

IV) If the entry in the CP_UniqueRespIdTable does match with an inherited BASE-VARIANT and that BASE-VARIANT does reside in an ECU-GROUP but the BASE-VARIANT has not yet been identified through a base variant identification process, follow the guidelines from the first case above, i.e. add the FUNCTIONAL-GROUP's POS-RESPONSE and NEG-RESPONSE acceptance mask data to the expected response table.

V) [Remark: Through this process, multiple identical acceptance masks could be set up in the expected response table. The D-server shall ensure uniqueness. In case multiple equivalent acceptance masks would have to be added that really represent different messages (e.g. one on the FUNCTIONAL-GROUP level and one on the BASE-VARIANT level), the D-server needs to store internally, that a D-PDU API returned acceptance Id points to different response message templates dependent on the UNIQUE_ID that has previously been obtained – see below.]

3) It uses the functional request address communication parameter(s) to fill the header information.

d) D-server sends out the functional request.

e) D-PDU API collects all incoming responses until a timeout occurs. Every response is sent directly to the D-server. This is also true for multi-part responses (e.g. UUDT messages with ISO 15765-2 based protocols).

f) D-server matches the unique identifier returned with every response message to identify the physical source of the response message. It also takes the returned acceptance id to select the RESPONSE template to decode the response PDU into physical data.

g) D-server sets the Access Key information for every received response based on the UNIQUE_ID and acceptance id returned from the D-PDU API.

1) In case the response was matched against a response template as defined on the FUNCTIONAL-GROUP level [cases b) iii) 1) and b) iii) 4)], build the Access Key as follows: <PROTOCOL.SHORT-NAME>.<FUNCTIONAL-GROUP.SHORT-NAME>.<*CP_ECULayerShortName*.SIMPLE-VALUE>

2) In case the response was matched against the overridden response template as defined on a BASE-VARIANT level [cases b) iii) 2) and b) iii) 3)] and the CP_ECULayerShortName parameter value has been overridden on BASE-VARIANT level, build the Access Key as follows: <PROTOCOL.SHORT-NAME>.<FUNCTIONAL-GROUP.SHORT-NAME>.>.<CP_ECULayerShortName.SIMPLE-VALUE>

3) In case the response was matched against the overridden response template as defined on a BASE-VARIANT level [cases b) iii) 2) and b) iii) 3)] and the CP_ECULayerShortName parameter value has not been overridden on BASE-VARIANT level, build the Access Key as follows: <PROTOCOL.SHORT-NAME>.<FUNCTIONAL-GROUP.SHORT-NAME>.<BASE-VARIANT.SHORT-NAME>

h) D-server sends responses to application within one result as soon as the communication primitive was terminated by the D-PDU API, which requires occurrence of a timeout.

i) Application interprets result as necessary.

7.4.9.4.2 For the example structure given in Figure 151 , the sequence of events is as described below.

a) **Case 1**: No BASE-VARIANT identification performed.

In case no BASE-VARIANT identification has been performed yet, the sequence of events is as in case c)1)i), and the RespId Table setup is as in Table 18.

Table 18 — RespId Table setup for no BASE-VARIANT identification has been performed

Resp Id (built by the D-server at runtime)	COMPARAM 1	COMPARAM 2	COMPARAM 3	CP_ECULayerShortName
1	0x7B2	0x5B2	0x7E2	"A or B"
2	0x7B4	0x5B4	0x7E4	"C"

For step c)2), the acceptance masks are set up from the following responses:

— for physical source "A or B": POS-RESPONSES A&B, NEG-RESPONSE A;

— for physical source "C": POS-RESPONSE B, NEG-RESPONSE A.

For case f), the Access-Keys would be set-up as follows:

— for responses on RespID 1: use *CP_ECULayerShortName*;

— for responses on RespID 2: use BASE-VARIANT C SHORT-NAME.

b) **Case 2**: BASE-VARIANT identification performed, identified BASE-VARIANT "A".

In case BASE-VARIANT "A" has been identified the sequence of events is as in case c)1)i), and the RespId Table set up is as in Table 19.

Table 19 — RespId Table setup for BASE-VARIANT identification - BASE-VARIANT "A"

Resp Id (built by the D-server at runtime)	COMPARAM 1	COMPARAM 2	COMPARAM 3	CP_ECULayerShortName
1	0x7B2	0x5B2	0x7E7	"A"
2	0x7B4	0x5B4	0x7E4	"C"

For step c)2), the acceptance masks are set up from the following responses:

— for physical source "A": POS-RESPONSE A, NEG-RESPONSE A;

— for physical source "C": POS-RESPONSE B, NEG-RESPONSE A.

For case f), the Access-Keys would be set-up as follows:

— for responses on RespID 1: use *CP_ECULayerShortName*;

— for responses on RespID 2: use BASE-VARIANT C SHORT-NAME.

c) **Case 3:** BASE-VARIANT identification performed, identified BASE-VARIANT B.

In case BASE-VARIANT B has been identified the sequence of events is as in case c)1)i), and the RespId Table set up is as in Table 20.

Table 20 — RespId Table setup for BASE-VARIANT identification - BASE-VARIANT B

Resp Id (built by the D-server at runtime)	COMPARAM 1	COMPARAM 2	COMPARAM 3	CP_ECULayerShortName
1	0x7B2	0x5B2	0x7E2	"A or B"[a]
2	0x7B4	0x5B4	0x7E4	"C"

[a] "A or B" remains in the CP_ECULayerShortName COMPARAM.

For step c)2), the acceptance masks are set up from the following responses:

— for physical source "B": POS-RESPONSEs A&B, NEG-RESPONSE A;

— for physical source "C": POS-RESPONSE B, NEG-RESPONSE A.

For case f), the Access-Keys would be set-up as follows:

— for responses on RespID 1: use BASE-VARIANT B SHORT-NAME;

IMPORTANT — The "A or B" setting for CP_ECULayerShortName is ignored in this case.

— for responses on RespID 2: use BASE-VARIANT C SHORT-NAME.

7.4.9.5 ODX examples

In the following, an ODX example is given that reflects the structure in Figure 151.

A COMPARAM-SUBSET and corresponding COMPARAM-SPEC are defined as shown below.

NOTE This is a reduced view: these are not complete communication parameter specifications for the protocol given.

```
<ODX xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="odx.xsd" MODEL-
VERSION="2.2.0">
  <COMPARAM-SUBSET ID="CP_SUBS_11361" CATEGORY="TRANSPORT">
    <SHORT-NAME>CPS_UDS_on_CAN</SHORT-NAME>
    <COMPARAMS>
      <COMPARAM ID="CP_14291" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
        <SHORT-NAME>CP_CanFuncReqId</SHORT-NAME>
        <PHYSICAL-DEFAULT-VALUE>2015</PHYSICAL-DEFAULT-VALUE>
        <DATA-OBJECT-PROP-REF ID-REF="DOP_11482"/>
      </COMPARAM>
    </COMPARAMS>
    <COMPLEX-COMPARAMS>
      <COMPLEX-COMPARAM ID="CCP_11381" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1"
                        CPUSAGE="ECU-COMM" ALLOW-MULTIPLE-VALUES="true">
        <SHORT-NAME>CP_UniqueRespIdTable</SHORT-NAME>
        <COMPARAM ID="CP_11401" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
          <SHORT-NAME>CP_CanPhysReqFormat</SHORT-NAME>
          <PHYSICAL-DEFAULT-VALUE>4</PHYSICAL-DEFAULT-VALUE>
          <DATA-OBJECT-PROP-REF ID-REF="DOP_11441"/>
        </COMPARAM>
        <COMPARAM ID="CP_11461" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
          <SHORT-NAME>CP_CanPhysReqId</SHORT-NAME>
```

```
      <PHYSICAL-DEFAULT-VALUE>2016</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11482"/>
    </COMPARAM>
    <COMPARAM ID="CP_11531" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
      <SHORT-NAME>CP_CanPhysReqExtAddr</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>0</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11441"/>
    </COMPARAM>
    <COMPARAM ID="CP_11551" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
      <SHORT-NAME>CP_CanRespUSDTFormat</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>4</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11441"/>
    </COMPARAM>
    <COMPARAM ID="CP_11571" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
      <SHORT-NAME>CP_CanRespUSDTId</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>2024</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11482"/>
    </COMPARAM>
    <COMPARAM ID="CP_11581" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
      <SHORT-NAME>CP_CanRespUSDTExtAddr</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>0</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11441"/>
    </COMPARAM>
    <COMPARAM ID="CP_11591" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
      <SHORT-NAME>CP_CanRespUUDTFormat</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>0</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11441"/>
    </COMPARAM>
    <COMPARAM ID="CP_12001" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
      <SHORT-NAME>CP_CanRespUUDTExtAddr</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>0</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11441"/>
    </COMPARAM>
    <COMPARAM ID="CP_12021" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="ECU-COMM">
      <SHORT-NAME>CP_CanRespUUDTId</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>1912</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_11482"/>
    </COMPARAM>
    <COMPARAM ID="CP_14261" PARAM-CLASS="UNIQUE_ID" CPTYPE="STANDARD" DISPLAY-LEVEL="1" CPUSAGE="APPLICATION">
      <SHORT-NAME>CP_ECULayerShortName</SHORT-NAME>
      <PHYSICAL-DEFAULT-VALUE>"None"</PHYSICAL-DEFAULT-VALUE>
      <DATA-OBJECT-PROP-REF ID-REF="DOP_14272 "/>
    </COMPARAM>
  </COMPLEX-COMPARAM>
</COMPLEX-COMPARAMS>
<DATA-OBJECT-PROPS>
  <DATA-OBJECT-PROP ID="DOP_11441">
    <SHORT-NAME>One_byte_identical</SHORT-NAME>
    <COMPU-METHOD>
      <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
      <BIT-LENGTH>8</BIT-LENGTH>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
  </DATA-OBJECT-PROP>
  <DATA-OBJECT-PROP ID="DOP_11482">
    <SHORT-NAME>Four_byte_identical</SHORT-NAME>
```

```
        <COMPU-METHOD>
          <CATEGORY>IDENTICAL</CATEGORY>
        </COMPU-METHOD>
        <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
          <BIT-LENGTH>32</BIT-LENGTH>
        </DIAG-CODED-TYPE>
        <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
      </DATA-OBJECT-PROP>
      <DATA-OBJECT-PROP ID="DOP_14272">
        <SHORT-NAME>String</SHORT-NAME>
        <COMPU-METHOD>
          <CATEGORY>IDENTICAL</CATEGORY>
        </COMPU-METHOD>
        <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UNICODE2STRING" xsi:type="MIN-MAX-LENGTH-TYPE" TERMINATION="ZERO">
          <MAX-LENGTH>256</MAX-LENGTH>
          <MIN-LENGTH>1</MIN-LENGTH>
        </DIAG-CODED-TYPE>
        <PHYSICAL-TYPE BASE-DATA-TYPE="A_UNICODE2STRING"/>
      </DATA-OBJECT-PROP>
    </DATA-OBJECT-PROPS>
  </COMPARAM-SUBSET>
</ODX>
```

The CP_UniqueRespIdTable complex communication parameter is defined as specified in the D-PDU API specification, with the additional CP_ECULayerShortName communication parameter added at its end.

```
<ODX xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="odx.xsd" MODEL-
VERSION="2.2.0">
    <COMPARAM-SPEC ID="CP_13391">
        <SHORT-NAME>CP_UDS_on_CAN</SHORT-NAME>
        <PROT-STACKS>
            <PROT-STACK ID="PS_13392">
                <SHORT-NAME>UDS_on_CAN</SHORT-NAME>
                <PDU-PROTOCOL-TYPE>ISO_15765_3_on_ISO_15765_2</PDU-PROTOCOL-TYPE>
                <PHYSICAL-LINK-TYPE>CAN</PHYSICAL-LINK-TYPE>
                <COMPARAM-SUBSET-REFS>
                    <COMPARAM-SUBSET-REF ID-REF="CP_SUBS_11361" DOCTYPE="COMPARAM-SUBSET" DOCREF="CPS_UDS_on_CAN"/>
                </COMPARAM-SUBSET-REFS>
            </PROT-STACK>
        </PROT-STACKS>
    </COMPARAM-SPEC>
</ODX>
```

Next, the functional group and the BASE-VARIANTs are defined (in this case within one DIAG-LAYER-CONTAINER).

```
<ODX xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="odx.xsd" MODEL-
VERSION="2.2.0">
    <DIAG-LAYER-CONTAINER ID="DLC_14121">
        <SHORT-NAME>DL_Container</SHORT-NAME>
        <FUNCTIONAL-GROUPS>
            <FUNCTIONAL-GROUP ID="FG_14122">
                <SHORT-NAME>FG</SHORT-NAME>
                <DIAG-COMMS>
                    <DIAG-SERVICE ID="DS_14401" ADDRESSING="FUNCTIONAL">
                        <SHORT-NAME>FUNC_SAMPLE</SHORT-NAME>
```

```
            <REQUEST-REF ID-REF="REQ_14412"/>
            <POS-RESPONSE-REFS>
                <POS-RESPONSE-REF ID-REF="PR_14581"/>
                <POS-RESPONSE-REF ID-REF="PR_15001"/>
            </POS-RESPONSE-REFS>
            <NEG-RESPONSE-REFS>
                <NEG-RESPONSE-REF ID-REF="NR_15031"/>
            </NEG-RESPONSE-REFS>
        </DIAG-SERVICE>
    </DIAG-COMMS>
    <REQUESTS>
        <REQUEST ID="REQ_14412">
            <SHORT-NAME>FUNC_REQ</SHORT-NAME>
            <PARAMS>
                <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>SID</SHORT-NAME>
                    <CODED-VALUE>34</CODED-VALUE>
                    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
                        <BIT-LENGTH>8</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
                </PARAM>
            </PARAMS>
        </REQUEST>
    </REQUESTS>
    <POS-RESPONSES>
        <POS-RESPONSE ID="PR_14581">
            <SHORT-NAME>PR_A</SHORT-NAME>
            <PARAMS>
                <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>SID</SHORT-NAME>
                    <BYTE-POSITION>0</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <CODED-VALUE>98</CODED-VALUE>
                    <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                        <BIT-LENGTH>8</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
                </PARAM>
            </PARAMS>
        </POS-RESPONSE>
        <POS-RESPONSE ID="PR_15001">
            <SHORT-NAME>PR_B</SHORT-NAME>
            <PARAMS>
                <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>SID</SHORT-NAME>
                    <BYTE-POSITION>0</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <CODED-VALUE>98</CODED-VALUE>
                    <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                        <BIT-LENGTH>8</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
                </PARAM>
                <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>Value</SHORT-NAME>
                    <BYTE-POSITION>1</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <CODED-VALUE>312</CODED-VALUE>
                    <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                        <BIT-LENGTH>16</BIT-LENGTH>
```

```
                    </DIAG-CODED-TYPE>
                </PARAM>
            </PARAMS>
        </POS-RESPONSE>
    </POS-RESPONSES>
    <NEG-RESPONSES>
        <NEG-RESPONSE ID="NR_15031">
            <SHORT-NAME>NR_A</SHORT-NAME>
            <PARAMS>
                <PARAM xsi:type="CODED-CONST">
                    <SHORT-NAME>SID</SHORT-NAME>
                    <BYTE-POSITION>0</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <CODED-VALUE>127</CODED-VALUE>
                    <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                        <BIT-LENGTH>8</BIT-LENGTH>
                    </DIAG-CODED-TYPE>
                </PARAM>
                <PARAM xsi:type="VALUE">
                    <SHORT-NAME>NRC</SHORT-NAME>
                    <BYTE-POSITION>1</BYTE-POSITION>
                    <BIT-POSITION>0</BIT-POSITION>
                    <DOP-SNREF SHORT-NAME="One_byte_identical"/>
                </PARAM>
            </PARAMS>
        </NEG-RESPONSE>
    </NEG-RESPONSES>
    <COMPARAM-REFS>
        <COMPARAM-REF ID-REF="CP_14291">
            <SIMPLE-VALUE>2001</SIMPLE-VALUE>
        </COMPARAM-REF>
        <COMPARAM-REF ID-REF="CCP_11381">
            <COMPLEX-VALUE>
                <SIMPLE-VALUE>4</SIMPLE-VALUE>
                <SIMPLE-VALUE>2018</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>4</SIMPLE-VALUE>
                <SIMPLE-VALUE>1970</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>1458</SIMPLE-VALUE>
                <SIMPLE-VALUE>A or B</SIMPLE-VALUE>
            </COMPLEX-VALUE>
            <PROT-STACK-SNREF SHORT-NAME="UDS_on_CAN"/>
        </COMPARAM-REF>
        <COMPARAM-REF ID-REF="CCP_11381">
            <COMPLEX-VALUE>
                <SIMPLE-VALUE>4</SIMPLE-VALUE>
                <SIMPLE-VALUE>2020</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>4</SIMPLE-VALUE>
                <SIMPLE-VALUE>1972</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>0</SIMPLE-VALUE>
                <SIMPLE-VALUE>1460</SIMPLE-VALUE>
                <SIMPLE-VALUE>C</SIMPLE-VALUE>
```

```
                </COMPLEX-VALUE>
                <PROT-STACK-SNREF SHORT-NAME="UDS_on_CAN"/>
            </COMPARAM-REF>
        </COMPARAM-REFS>
    </FUNCTIONAL-GROUP>
</FUNCTIONAL-GROUPS>
<BASE-VARIANTS>
    <BASE-VARIANT ID="BV_15301">
        <SHORT-NAME>A</SHORT-NAME>
        <DIAG-COMMS>
            <DIAG-SERVICE ID="DS_16011" ADDRESSING="FUNCTIONAL">
                <SHORT-NAME>FUNC_SAMPLE</SHORT-NAME>
                <POS-RESPONSE-REFS>
                    <POS-RESPONSE-REF ID-REF="PS_15361"/>
                </POS-RESPONSE-REFS>
                <NEG-RESPONSE-REFS>
                    <NEG-RESPONSE-REF ID-REF="NR_15381"/>
                </NEG-RESPONSE-REFS>
            </DIAG-SERVICE>
        </DIAG-COMMS>
        <POS-RESPONSES>
            <POS-RESPONSE ID="PS_15361">
                <SHORT-NAME>PR_A</SHORT-NAME>
                <PARAMS>
                    <PARAM xsi:type="CODED-CONST">
                        <SHORT-NAME>SID</SHORT-NAME>
                        <BYTE-POSITION>0</BYTE-POSITION>
                        <BIT-POSITION>0</BIT-POSITION>
                        <CODED-VALUE>99</CODED-VALUE>
                        <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                            <BIT-LENGTH>8</BIT-LENGTH>
                        </DIAG-CODED-TYPE>
                    </PARAM>
                </PARAMS>
            </POS-RESPONSE>
        </POS-RESPONSES>
        <NEG-RESPONSES>
            <NEG-RESPONSE ID="NR_15381">
                <SHORT-NAME>NR_A</SHORT-NAME>
                <PARAMS>
                    <PARAM xsi:type="CODED-CONST">
                        <SHORT-NAME>SID</SHORT-NAME>
                        <BYTE-POSITION>0</BYTE-POSITION>
                        <BIT-POSITION>0</BIT-POSITION>
                        <CODED-VALUE>127</CODED-VALUE>
                        <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                            <BIT-LENGTH>8</BIT-LENGTH>
                        </DIAG-CODED-TYPE>
                    </PARAM>
                </PARAMS>
            </NEG-RESPONSE>
        </NEG-RESPONSES>
        <COMPARAM-REFS>
            <COMPARAM-REF ID-REF="CCP_11381">
                <COMPLEX-VALUE>
                    <SIMPLE-VALUE>4</SIMPLE-VALUE>
                    <SIMPLE-VALUE>2018</SIMPLE-VALUE>
                    <SIMPLE-VALUE>0</SIMPLE-VALUE>
```

```
                    <SIMPLE-VALUE>4</SIMPLE-VALUE>
                    <SIMPLE-VALUE>1970</SIMPLE-VALUE>
                    <SIMPLE-VALUE>0</SIMPLE-VALUE>
                    <SIMPLE-VALUE>0</SIMPLE-VALUE>
                    <SIMPLE-VALUE>0</SIMPLE-VALUE>
                    <SIMPLE-VALUE>1463</SIMPLE-VALUE>
                    <SIMPLE-VALUE>A</SIMPLE-VALUE>
                </COMPLEX-VALUE>
                <PROT-STACK-SNREF SHORT-NAME="UDS_on_CAN"/>
            </COMPARAM-REF>
        </COMPARAM-REFS>
    </BASE-VARIANT>
    <BASE-VARIANT ID="BV_15481">
        <SHORT-NAME>B</SHORT-NAME>
    </BASE-VARIANT>
    <BASE-VARIANT ID="BV_16041">
        <SHORT-NAME>C</SHORT-NAME>
        <DIAG-COMMS>
            <DIAG-SERVICE ID="DS_16061" ADDRESSING="FUNCTIONAL">
                <SHORT-NAME>FUNC_SAMPLE</SHORT-NAME>
                <POS-RESPONSE-REFS>
                    <POS-RESPONSE-REF ID-REF="PS_16051"/>
                </POS-RESPONSE-REFS>
                <NEG-RESPONSE-REFS>
                    <NEG-RESPONSE-REF ID-REF="NR_16052"/>
                </NEG-RESPONSE-REFS>
            </DIAG-SERVICE>
        </DIAG-COMMS>
        <POS-RESPONSES>
            <POS-RESPONSE ID="PS_16051">
                <SHORT-NAME>PR_B</SHORT-NAME>
                <PARAMS>
                    <PARAM xsi:type="CODED-CONST">
                        <SHORT-NAME>SID</SHORT-NAME>
                        <BYTE-POSITION>0</BYTE-POSITION>
                        <BIT-POSITION>0</BIT-POSITION>
                        <CODED-VALUE>98</CODED-VALUE>
                        <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                            <BIT-LENGTH>8</BIT-LENGTH>
                        </DIAG-CODED-TYPE>
                    </PARAM>
                </PARAMS>
            </POS-RESPONSE>
        </POS-RESPONSES>
        <NEG-RESPONSES>
            <NEG-RESPONSE ID="NR_16052">
                <SHORT-NAME>NR_A</SHORT-NAME>
                <PARAMS>
                    <PARAM xsi:type="CODED-CONST">
                        <SHORT-NAME>SID</SHORT-NAME>
                        <BYTE-POSITION>0</BYTE-POSITION>
                        <BIT-POSITION>0</BIT-POSITION>
                        <CODED-VALUE>127</CODED-VALUE>
                        <DIAG-CODED-TYPE xsi:type="STANDARD-LENGTH-TYPE" BASE-DATA-TYPE="A_UINT32">
                            <BIT-LENGTH>8</BIT-LENGTH>
                        </DIAG-CODED-TYPE>
                    </PARAM>
                </PARAMS>
```

```
                </NEG-RESPONSE>
            </NEG-RESPONSES>
            <COMPARAM-REFS>
                <COMPARAM-REF ID-REF="CCP_11381">
                    <COMPLEX-VALUE>
                        <SIMPLE-VALUE>4</SIMPLE-VALUE>
                        <SIMPLE-VALUE>2018</SIMPLE-VALUE>
                        <SIMPLE-VALUE>0</SIMPLE-VALUE>
                        <SIMPLE-VALUE>4</SIMPLE-VALUE>
                        <SIMPLE-VALUE>1970</SIMPLE-VALUE>
                        <SIMPLE-VALUE>0</SIMPLE-VALUE>
                        <SIMPLE-VALUE>0</SIMPLE-VALUE>
                        <SIMPLE-VALUE>0</SIMPLE-VALUE>
                        <SIMPLE-VALUE>1463</SIMPLE-VALUE>
                        <SIMPLE-VALUE>C</SIMPLE-VALUE>
                    </COMPLEX-VALUE>
                    <PROT-STACK-SNREF SHORT-NAME="UDS_on_CAN"/>
                </COMPARAM-REF>
            </COMPARAM-REFS>
        </BASE-VARIANT>
    </BASE-VARIANTS>
</DIAG-LAYER-CONTAINER>
</ODX>
```

7.5 ODX data model for ECU memory programming

7.5.1 Overview

One use case of data transfer between any ECU and any D-server is the programming/(partial) reprogramming of ECUs, called “ECU memory programming”. ODX offers a possibility to describe the data needed for an upload or a download procedure. In the following the notation “flash” is also used instead of download. Inside this ODX specification, download and upload are defined in the same way as in ISO 14229-1 and ISO 14230, i.e. download in ODX case means data stream from D-server into ECU, upload means data stream from ECU to D-server. A diagnostic application can use the information in this subclause (7.5) to initiate such a flash process. For this purpose it offers the user a list of all available sessions. After the user has taken his choice, the selected session can be flashed. All needed information, i.e. flash job, compatibility of data with ECU or files containing the actual flash data can be defined within an ODX instance. Even the flash data itself can be defined there. See ISO 22900-3 specification for details of flash jobs and their handling. This subclause specifies the object classes used for programming/reprogramming of ECUs.

NOTE The usage scenario in 7.4.8 Variable length parameter is an example for the upload of data.

7.5.2 ECU-MEM data model description

7.5.2.1 Structure of ECU-MEM

Figure 152 — UML representation and structure of ECU-MEM illustrates the modelling.

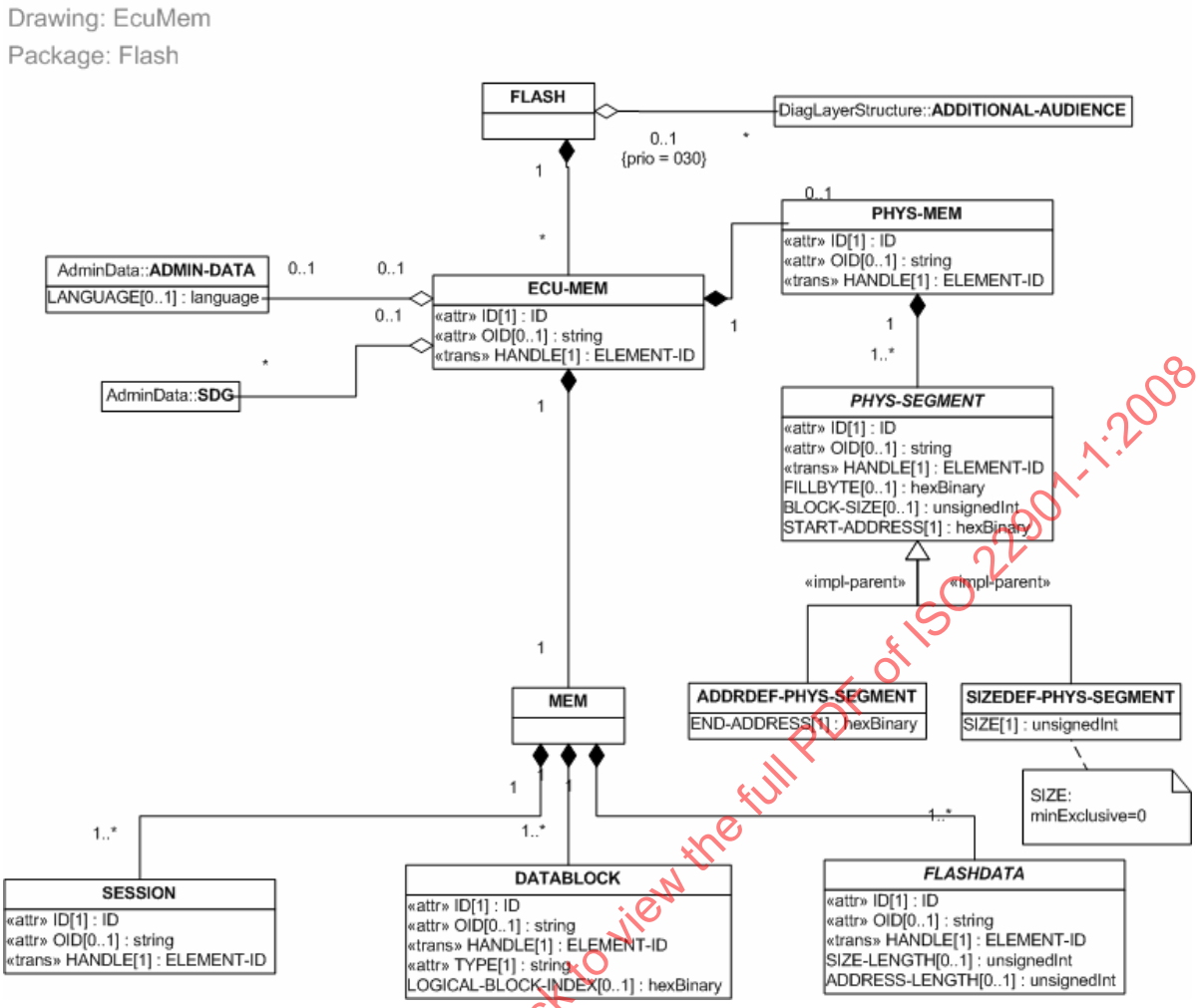


Figure 152 — UML representation and structure of ECU-MEM

A single ECU-MEM object is a container for flash data transfer between development, factory and customer. It includes the description of memory of one or more ECUs, given by SESSIONs, DATABLOCKs, FLASHDATAs and PHYS-MEMs.

Typically, an ECU-MEM instance can only be used in conjunction with the appropriate single ECU job (the flash job). For example, it is possible to define manufacturer-specific information like checksum algorithm or encryption method. For this reason, an ECU-MEM instance and the flash job that understand the data in this instance should be exchanged together.

7.5.2.2 Sessions

SESSIONs are the only logical units that can be chosen for programming. The choice of a session occurs by SESSION-DESC (see 7.5.3), which references the appropriate SESSION object and the flash job responsible for programming. Figure 153 — UML representation and structure of SESSION illustrates the modelling.

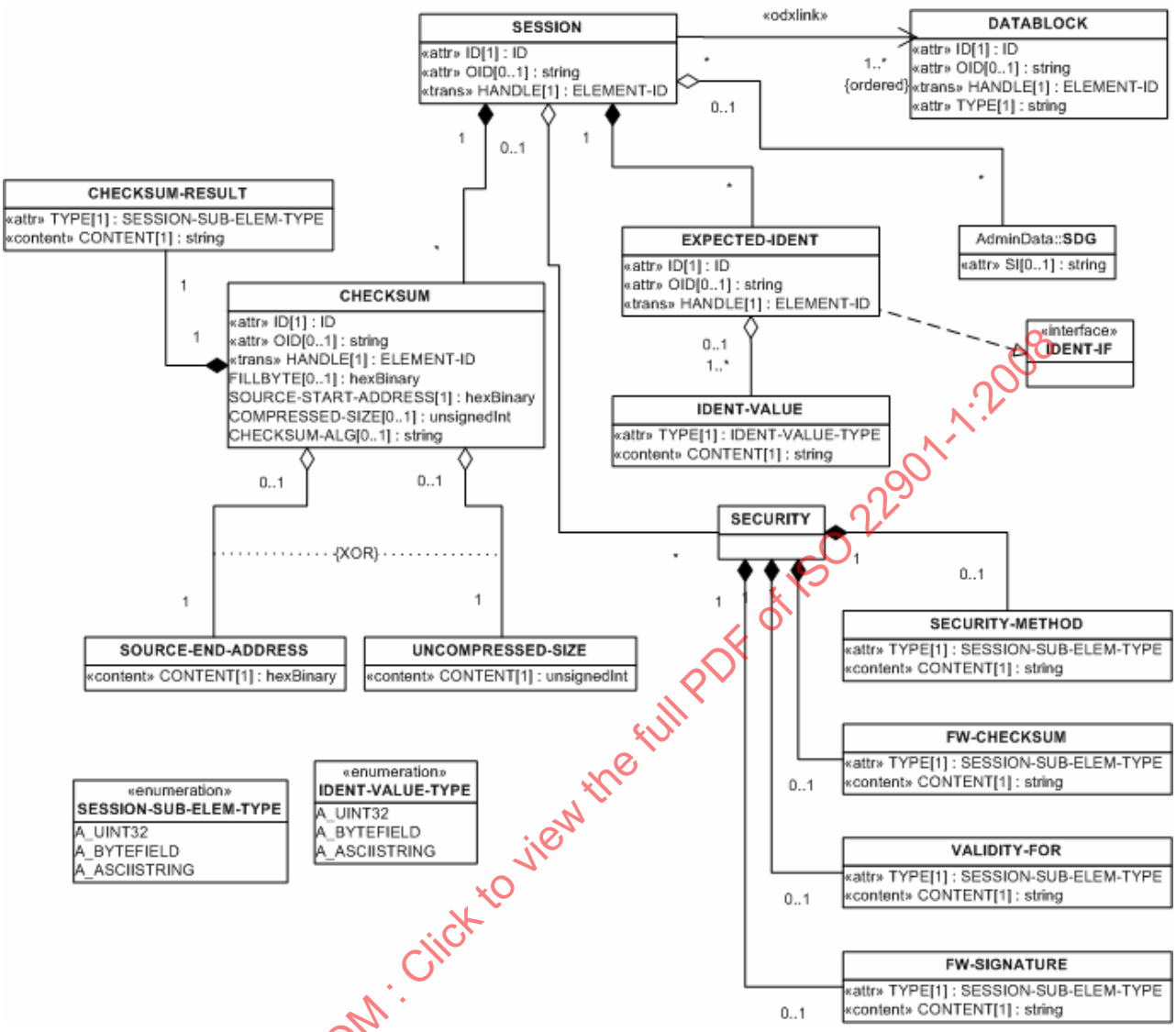


Figure 153 — UML representation and structure of SESSION

A SESSION object involves the standard members SHORT-NAME, LONG-NAME, and DESC. It describes which DATABLOCKs are programmed within this session. The DATABLOCKs are referenced by <<odxlink>> and are programmed according to the order of the references.

Figure 154 — Example: ECU-MEM with three sessions shows an example of an ECU-MEM object. There are three sessions in this ECU-MEM, e.g. Session1 consists of three datablocks: FlashDriver, Code, and DataA. Depending on the selected session, different datablocks will be programmed.

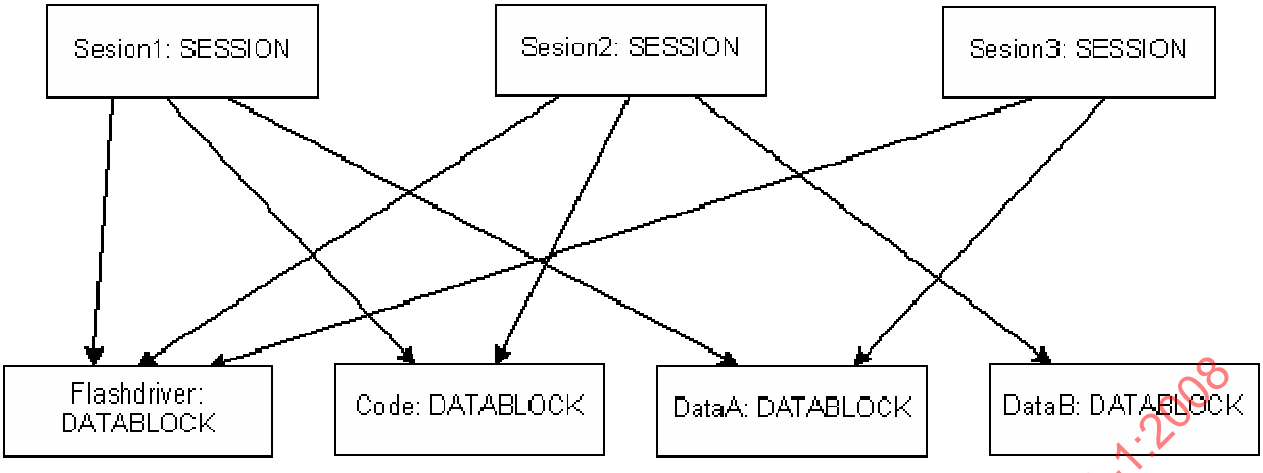


Figure 154 — Example: ECU-MEM with three sessions

EXPECTED-IDENTs hold information regarding the target device. This information can be used to verify if the SESSION is compatible to the target device. This means that the current SESSION can only be programmed into an ECU if all EXPECTED-IDENTs match with the respective idents read from the ECU. The service and the response parameter used to read the identification of the ECU is specified by that object IDENT-DESC of the referencing ECU-MEM-CONNECTOR which IDENT-IF-SNREF specifies the same SHORT-NAME as the EXPECTED-IDENT. One EXPECTED-IDENT matches if at least one of its IDENT-VALUEs is equal to the result of the response parameter.

```
<EXPECTED-IDENTS>
    <EXPECTED-IDENT ID = "F.ECUMEM.MEM.SESSION.EXPIDENT01.ID">
        <SHORT-NAME>HARDWARE</SHORT-NAME>
        <LONG-NAME>Hardware</LONG-NAME>
        <DESC>
            <p>Required hardware version for this calibration data version</p>
        </DESC>
        <IDENT-VALUES>
            <IDENT-VALUE TYPE = "A_ASCIISTRING">hw-value 1</IDENT-VALUE>
            <IDENT-VALUE TYPE = "A_ASCIISTRING">hw-value 2</IDENT-VALUE>
        </IDENT-VALUES>
    </EXPECTED-IDENT>
    <EXPECTED-IDENT ID = "F.ECUMEM.MEM.SESSION.EXPIDENT02.ID">
        <SHORT-NAME>SOFTWARE2</SHORT-NAME>
        <LONG-NAME>Software 2</LONG-NAME>
        <DESC>
            <p>Required software version 2 for this calibration version</p>
        </DESC>
        <IDENT-VALUES>
            <IDENT-VALUE TYPE = "A_ASCIISTRING">3.1</IDENT-VALUE>
            <IDENT-VALUE TYPE = "A_ASCIISTRING">3.2</IDENT-VALUE>
        </IDENT-VALUES>
    </EXPECTED-IDENT>
</EXPECTED-IDENTS>
```

NOTE No logical connection exists between the IDENT-VALUEs across the EXPECTED-IDENTs. Hence, not only the two combinations (HARDWARE = “hw-value 1”, SOFTWARE = “3.1”) and (HARDWARE = “hw-value 2”, SOFTWARE = “3.2”) fulfil this EXPECTED-IDENTS description, but the combinations (HARDWARE = “hw-value 2”, SOFTWARE = “3.1”) and (HARDWARE = “hw-value 1”, SOFTWARE = “3.2”), too.

Within one session one or more CHECKSUMs may be declared to allow defining of various sets of checksum-result, address range, and algorithm. A CHECKSUM is used to perform checks within the flash process. This information can be used by the flash job to calculate the checksum of the flashed data. If checksum monitoring is performed, the respective algorithm is identified by the name in CHECKSUM-ALG, which may be needed by a job to calculate the CHECKSUM-RESULT. The address range considered for the checksum calculation, is specified by SOURCE-START-ADDR and SOURCE-END-ADDR (or UNCOMPRESSED-SIZE). SOURCE-START-ADDR points to the first byte to be included in the checksum calculation. SOURCE-END-ADDR denotes the last byte to be included in the checksum calculation. Alternatively, UNCOMPRESSED-SIZE can be used to calculate the end address of checksum calculation. COMPRESSED-SIZE holds the size of the compressed SEGMENT/CHECKSUM. The expected result can be found in CHECKSUM-RESULT. A FILLBYTE is used to fill gaps among the logical segments (SEGMENT) in the flash data for checksum calculation of the given address range. Because the checksum calculation is implemented inside the flash job instead of the D-server, the user is free to adapt the algorithm to his own requirements. The SOURCE-START-ADDR and SOURCE-END-ADDR shall be specified as unsigned integer values in the hexadecimal format (Schema datatype hexBinary), while UNCOMPRESSED-SIZE and COMPRESSED-SIZE shall be specified as unsigned integer values in the decimal format (Schema datatype unsignedInt).

With SECURITY it is possible to define security specific information (e.g.: checksum, signature) of the whole session. This information can be used by the flash job to check the integrity and the authenticity of the flash data. The member SECURITY-METHOD denotes the chosen security concept. This might be either single information about the used method or a reference to an external access table. FW-SIGNATURE holds the signature, which shall be sent to the target device (ECU) for verification of authenticity. FW-CHECKSUM holds the checksum (e.g. a CRC32), which shall be sent to the target device for verification of integrity. VALIDITY-FOR describes, which ECU belongs to the given signature. Together with SECURITY-METHOD it is used to influence the behaviour of the flash job. Because the security method is implemented inside the flash job instead of the D-server, the user is free to adapt the algorithm to his own requirements.

```
<SECURITYS>
    <SECURITY>
        <SECURITY-METHOD TYPE = "A_ASCIISTRING">method1</SECURITY-METHOD>
        <FW-SIGNATURE TYPE = "A_ASCIISTRING">00112233</FW-SIGNATURE>
        <FW-CHECKSUM TYPE = "A_ASCIISTRING">44556677</FW-CHECKSUM>
        <VALIDITY-FOR TYPE = "A_ASCIISTRING">hw1</VALIDITY-FOR>
    </SECURITY>
</SECURITYS>
```

It is recommended to use SECURITY to check the integrity and authenticity of the source data and CHECKSUM to calculate the checksum of the data flashed to (or stored in) the ECU. It is also reasonable to use SECURITY directly in DATABLOCKs, because they define the actual flash data.

7.5.2.3 Datablocks

7.5.2.3.1 General

A SESSION references one or more DATABLOCKs, which describe the logical structure of the referenced FLASHDATA, e.g. a DATABLOCK can describe a whole code section, a single element like one calibration value; or a flash driver, which shall be loaded into the RAM before the reprogramming routine, can be executed. See Figure 155 — UML representation and structure of datablock for detailed modelling information.

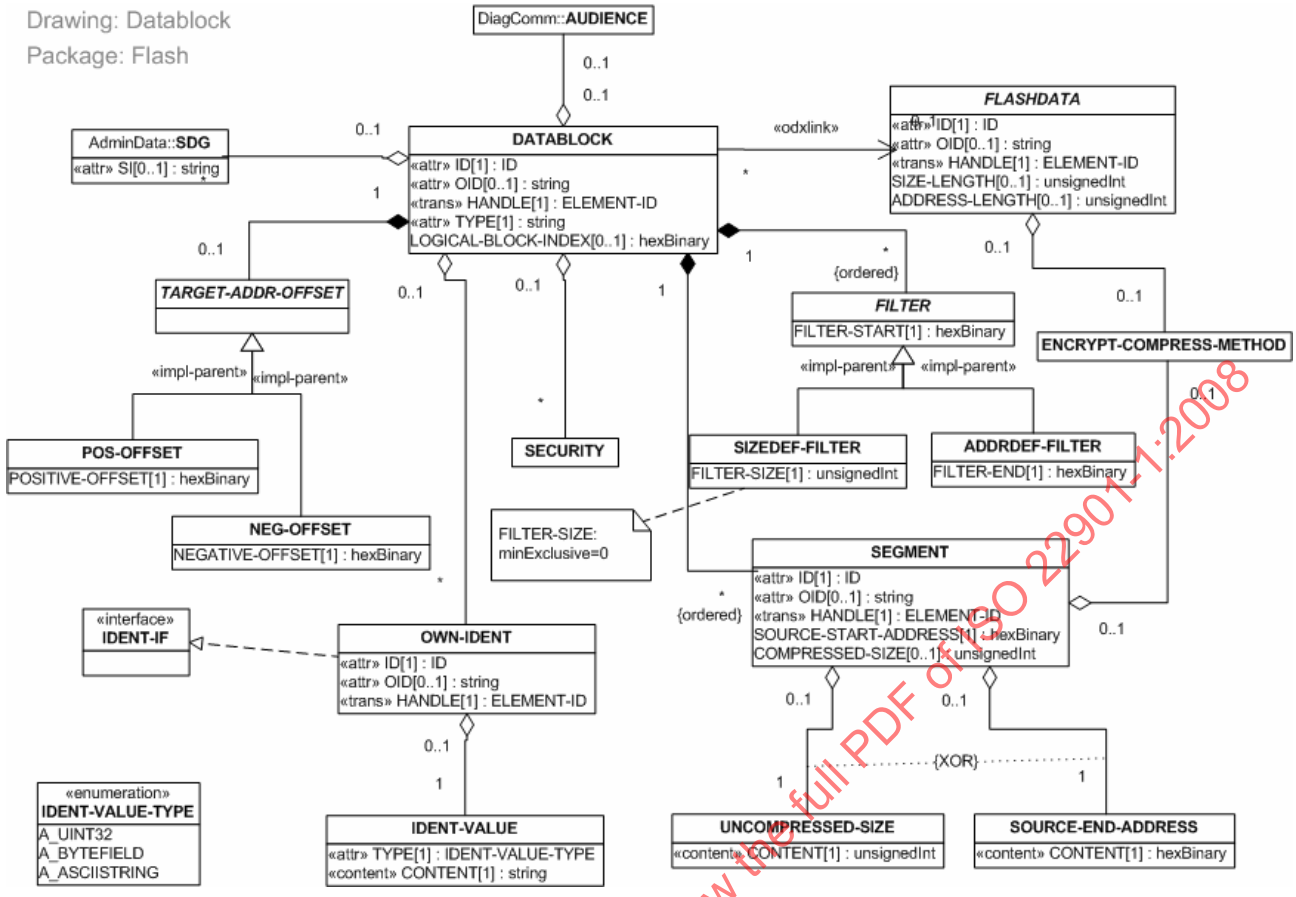


Figure 155 — UML representation and structure of datablock

The member TYPE describes the kind of datablock (e.g. “BOOT”, “CODE”, “DATA”, or other values, this TYPE is just a classification and any other proprietary TYPE could be used). This value will be used by a generic programming job to select the method used to unlock or to program the flashdata in this datablock.

The member LOGICAL-BLOCK-INDEX can be used as parameter for the service “eraseMemory” as described in the following examples. In this case the physical address range of the logical data block is known by the bootloader of the ECU.

Each DATABLOCK references to the corresponding FLASHDATA, that contains the actual data described in 0.

FILTERs can be used to reduce the flash data from the source. The data inside the source which are not covered by one of the filters will be ignored. If no FILTER is defined, the whole file or the content of DATA will be processed further. Otherwise, the data inside the source is reduced to that part of the data, which fall into one FILTER range (positive filter, pass filter). The D-server will not present an error if no data are available which pass a FILTER. Address ranges of two different FILTERs shall not overlap. The FILTER size shall always be greater than zero. The first of the following filters defines an address range going from 0x0000 to 0x1FFF, the second an address range from 0x4000 to 0x5FFF.

The lengths of both intervals are equal 0x2000:

```
<FILTERS>
    <FILTER xsi:type = "ADDRDEF-FILTER">
        <FILTER-START>0000</FILTER-START>
        <FILTER-END>1FFF</FILTER-END>
    </FILTER>
    <FILTER xsi:type = "SIZEDEF-FILTER">
        <FILTER-START>4000</FILTER-START>
```

```
        <FILTER-SIZE>8192</FILTER-SIZE>
    </FILTER>
</FILTERS>
```

Depending on the type of data, a FILTER describes:

— The address range in case of hex formatted data (INTEL-HEX, MOTOROLA-S). Applying of all FILTERs leads to a sub-set of data with the address information taken over from the source. If the sub-set does not cover a continuous range of addresses the D-server shall order the ranges according to increasing addresses, concatenate all adjoining ranges, and report an error if a range overlaps another one.

EXAMPLE 1 Assume that the hex formated file contains three lines in the following order:

(a) address = 0x0104, size = 16, data = 0x04, 0x05, 0x06, 0x07 … 0x10, 0x11, 0x12, 0x13

(b) address = 0x0114, size = 8, data = 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B

(c) address = 0x00FC, size = 8, data = 0xFC, 0xFD, 0xFE, 0xFF, 0x00, 0x01, 0x02, 0x03

Assume that three FILTERs are specified:

(A) filter start = 0x0120, filter end = 0xFFFF

(B) filter start = 0x0100, filter size = 8

(C) filter start = 0x0110, filter end = 0x0117

In the first step all pieces of data are determined which pass one of the FILTERs. These are four pieces in this order:

(1) address = 0x0104, size = 4, data = 0x04, 0x05, 0x06, 0x07 (line a passing filter B)

(2) address = 0x0110, size = 4, data = 0x10, 0x11, 0x12, 0x13 (line a passing filter C)

(3) address = 0x0114, size = 4, data = 0x14, 0x15, 0x16, 0x17 (line b passing filter C)

(4) address = 0x0100, size = 4, data = 0x00, 0x01, 0x02, 0x03 (line c passing filter B)

Ordering according to increases addresses results in:

(4) address = 0x0100, size = 4, data = 0x00, 0x01, 0x02, 0x03

(1) address = 0x0104, size = 4, data = 0x04, 0x05, 0x06, 0x07

(2) address = 0x0110, size = 4, data = 0x10, 0x11, 0x12, 0x13

(3) address = 0x0114, size = 4, data = 0x14, 0x15, 0x16, 0x17

Concatenation of adjoining pieces results in the following two address ranges:

address = 0x0100, size = 8, data = 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 (pieces 4 + 1)

address = 0x0110, size = 8, data = 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 (pieces 2 + 3)

— The source offset and the length in case of unformatted data (BINARY), where the source can be an external file or inline code defined by DATA in FLASHDATA (see 7.5.2.4). FILTERs shall be applied in the sequence they are defined in ODX. After applying of all FILTERs in this case, we get a single byte stream without any address information.

EXAMPLE 2 Assume that the binary formatted data contains 256 bytes with the values 0x00 … 0xFF in this order. Assume that three FILTERs are specified in this order:

(A) filter start = 0x100, filter end = 0xFFFF

(B) filter start = 0x020, filter end = 0x27

(C) filter start = 0x010, filter size = 8

Appling the three FILTERs in the order of specification results in only two pieces, because no data passing filter A:

size = 8, data = 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27 (passing filter B)

size = 8, data = 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 (passing filter C)

To get the result, all pieces are concatenated to a single byte stream of 16 bytes length:

0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17

Each DATABLOCK contains at least one programmable segment, which is given explicitly (by SEGMENTs) or is computed (by any flash tool e.g.) from the address information in the FLASHDATA source. A segment represents a continuous data stream (without gaps) with a start and an end address. A SEGMENT object defines therefore a SOURCE-START-ADDR and either SOURCE-END-ADDR or UNCOMPRESSED-SIZE.

The alternative value can easily be calculated by the equations:

```
UNCOMPRESSED-SIZE = SOURCE-END-ADDR - SOURCE-START-ADDR + 1
SOURCE-END-ADDR = SOURCE-START-ADDR + UNCOMPRESSED-SIZE - 1
```

Address ranges of two different SEGMENTs shall not overlap. The following is an example for an address and a size defined SEGMENTs, each of size 3889 bytes:

```
<SEGMENTS>
    <SEGMENT ID = "F.ECUMEM.MEM.SESSION.DATABLOCK.SEGMENT01.ID">
        <SHORT-NAME>SEGMENT1</SHORT-NAME>
        <LONG-NAME>Segment 1</LONG-NAME>
        <SOURCE-START-ADDRESS>0000</SOURCE-START-ADDRESS>
        <SOURCE-END-ADDRESS>0F31</SOURCE-END-ADDRESS>
    </SEGMENT>
    <SEGMENT ID = "F.ECUMEM.MEM.SESSION.DATABLOCK.SEGMENT02.ID">
        <SHORT-NAME>SEGMENT2</SHORT-NAME>
        <LONG-NAME>Segment 2</LONG-NAME>
        <SOURCE-START-ADDRESS>1000</SOURCE-START-ADDRESS>
        <UNCOMPRESSED-SIZE>3889</UNCOMPRESSED-SIZE>
    </SEGMENT>
</SEGMENTS>
```

In general, the start address inside the ECU, which is passed in the service “requestDownload” in ISO 14229-1 and ISO 14230 (service 0x34), is equal to the SOURCE-START-ADDR. Within the object TARGET-ADDR-

OFFSET a difference between source address and target address (ECU address) can be specified. This can be either a POS-OFFSET, if the target address is higher than the source address, or a NEG-OFFSET, if the target address is lower than the source address. In both cases a positive value shall be entered. After all segments are determined, TARGET-ADDR-OFFSET is added to (or subtract from) all start and end addresses of the segments. For all segments inside one DATABLOCK the same address offset is applied.

The UNCOMPRESSED-SIZE can be used e.g. for the calculation of the memory consumption when it is flashed into the ECU. In general, it is passed to the ECU in the service “requestDownload”.

An encryption or a compression usually affect the number of flash data to be transferred to the ECU. In this case, the value of COMPRESSED-SIZE contains the actual number of flash data to transfer, while UNCOMPRESSED-SIZE describes the number of bytes occupying the memory inside the ECU after decrypting and/or uncompressing of the flash data. If COMPRESSED-SIZE is not specified it is assumed to be equal to UNCOMPRESSED-SIZE. The flash job shall transfer this number of bytes in one or multiple calls of the transfer service, e.g. “transferData” in ISO 14229-1and ISO 14230 (service 0x36). It is strongly recommended to use unformatted data (BINARY) for compressed and/or encrypted flash data, because usually the encryption method and sometimes even the compression method increases the number of bytes. In such cases, it is impossible to use hex formatted data.

The optional element ENCRYPT-COMPRESS-METHOD specifies the encryption and compression algorithm used for the SEGMENT. In contrast to ENCRYPT-COMPRESS-METHOD inside FLASH-DATA, the element in SEGMENT may be specified individually for each SEGMENT and overrides those in FLASHDATA, if present. This way, it is possible to combine e.g. uncompressed and compressed data in a single DATABLOCK. The flash job can use the value of ENCRYPT-COMPRESS-METHOD to set the parameter “dataFormatIdentifier” of the service “requestDownload”. If defined, the value at the SEGMENT shall be used, otherwise the value defined at the FLASHDATA.

Again, depending on data type, the interpretation of SEGMENTs differ, as described below.

a) In the case of hex formatted data (INTEL-HEX, MOTOROLA-S), it can be used as a filter in a second step. Each address range passing the FILTERs is interpreted as one segment. The address of the range is interpreted as the start address of the segment and its size as the uncompressed size. The order of segments is determined by their increasing start addresses, however, there exist three use cases to specify SEGMENTs:

   1) not all address ranges passing the FILTERs shall actually be programmed;

   2) the flash data passing the FILTERs are compressed (only possible if the compressed data are actually not longer than the uncompressed data);

   3) the order of programming the segments should explicitly specified.

b) In all cases, the processing in the D-server is the same. For each SEGMENT, the procedure below is followed:

   1) First, it applies the address information of the SEGMENT (SOURCE-START-ADDR, SOURCE-END-ADDR or UNCOMPRESSED-SIZE) to all address ranges passing the FILTERs similar to FILTER. Exactly one address range shall be passed. In addition, it shall contain a byte at the SOURCE-START-ADDR.

   2) Secondly, all passing bytes of this address range are the flash data of this segment and the number is the compressed size. If COMPRESSED-SIZE is explicitly specified, its content shall be equal to this compressed size. Because UNCOMPRESSED-SIZE is applied in the filtering step, the compressed size cannot be greater than the uncompressed size.

   3) Third, this data is interpreted as one segment with start address given by SOURCE-START-ADDR, the uncompressed size given by UNCOMPRESSED-SIZE, and the flash data those number of bytes are the compressed size.

If any of the conditions mentioned above is not fulfilled, the D-server shall report an error and immediately breaks the processing.

EXAMPLE 3 Resume the example for FILTERs above. Two address ranges passing the FILTERs:

— address = 0x0100, size = 8, data = 0x00, 0x01, 0x02, … 0x07

— address = 0x0110, size = 8, data = 0x10, 0x11, 0x12, … 0x17

Table 21 — Example for SEGMENT processing of hex formatted data gives some corresponding examples.

Table 21 — Example for SEGMENT processing of hex formatted data

SEGMENT specification	Result
SOURCE-START-ADDR = 0x0102 UNCOMPRESSED-SIZE = 4 optionally: COMPRESSED-SIZE = 4	bytes passing filter: addresses 0x0102 … 0x0105 start address = 0x0102, compressed size = 4, uncompressed size = 4, flash data = 0x02, 0x03, 0x04, 0x05
SOURCE-START-ADDR = 0x0102 UNCOMPRESSED-SIZE = 14 optionally: COMPRESSED-SIZE = 6	bytes passing filter: addresses 0x0102 … 0x0107 start address = 0x0102, compressed size = 6, uncompressed size = 14, flash data = 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
SOURCE-START-ADDR = 0x0102 UNCOMPRESSED-SIZE = 15	bytes passing filter: addresses 0x0102 … 0x0107 and 0x0110 error, because not exactly one address range are passing the filter 0x0102 … 0x0110
SOURCE-START-ADDR = 0x00FF UNCOMPRESSED-SIZE = 8	bytes passing filter: addresses 0x0100 … 0x0106 error, because no byte available at source start address 0x0FF
SOURCE-START-ADDR = 0x0102 UNCOMPRESSED-SIZE = 14 COMPRESSED-SIZE = 4	bytes passing filter: addresses 0x0102 … 0x0107 error, because address range contains 6 bytes, but exactly 4 bytes are expected

c) Before a segment can be flashed, the TARGET-ADDR-OFFSET defined in DATABLOCK shall be applied to all address values.

d) In the case of unformatted data (BINARY), the flash data or in the case of available FILTERS the filtered data (see above) form a byte stream of certain length without any address information.

1) It is strongly recommended to explicitly specify SEGMENTs in the case of binary data. If no SEGMENTS are specified, the reprogramming device shall use the following address information:

i) if one or more FILTERs are defined, assume the corresponding number of segments with FILTER-START as SOURCE-START-ADDR and FILTER-END as SOURCE-END-ADDR or FILTER-SIZE as UNCOMPRESSED-SIZE: the order of FILTERs determines the order of segments;

ii) if there is no FILTER in the DATABLOCK, assume a single segment with 00h as start address and the byte stream length as UNCOMPRESSED-SIZE.

2) Each segment, either defined by explicit SEGMENT elements or via the above interpretation, specifies the address and the number of bytes of the segment. The address is given by SOURCE-START-ADDR which could be modified by TARGET-ADDR-OFFSET. The length is given by COMPRESSED-SIZE. If this optional element is not specified, it is either specified by UNCOMPRESSED-SIZE or the difference between SOURCE-END-ADDR and SOURCE-START-ADDR (incremented by 1).

3) All segments shall be processed in the order of specification. They take the number of bytes (determined by the way described above) out of the byte stream continuously. The D-server shall report an error, if more bytes are requested than available in the byte stream. It is allowed that bytes remain in the byte stream.

EXAMPLE 4 Resume the example for FILTERs above. This byte stream of 16 bytes passed the FILTERs:

0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17

Table 22 — Example for SEGMENT processing of unformatted data gives some corresponding examples.

Table 22 — Example for SEGMENT processing of unformatted data

SEGMENTS specification	Result
First SEGMENT: SOURCE-START-ADDR = 0x0100 UNCOMPRESSED-SIZE = 5	extract 5 bytes from byte stream start address = 0x0100, compressed size = 5, uncompressed size = 5, flash data = 0x20, 0x21, 0x22, 0x23, 0x24
Second SEGMENT: SOURCE-START-ADDR = 0x0200 UNCOMPRESSED-SIZE = 5 COMPRESSED-SIZE = 5	extract 5 bytes from byte stream start address = 0x0200, compressed size = 5, uncompressed size = 5, flash data = 0x25, 0x26, 0x27, 0x10, 0x11
Third SEGMENT: SOURCE-START-ADDR = 0x0300 UNCOMPRESSED-SIZE = 16 COMPRESSED-SIZE = 5	extract 5 bytes from byte stream start address = 0x0300, compressed size = 5, uncompressed size = 16, flash data = 0x12, 0x13, 0x14, 0x15, 0x16
–	one byte with value 0x17 is left in the byte stream, which will be ignored

IMPORTANT — In this example, an explicit specification of SEGMENT is necessary. Otherwise, the FILTERs would be applied, but all FILTERs require 0xFF10 bytes, but only 0x10 bytes are available.

A DATABLOCK may contain a list of OWN-IDENTs, which are used for check purposes. For example, a test device needs the possibility to compare the current software version within the target device (ECU) with the new software version before reprogramming. The information for this comparison is described within the object OWN-IDENT, e.g. a particular OWN-IDENT could be a part number, a software version, or a supplier.

```
<OWN-IDENTS>
    <OWN-IDENT ID = "F.ECUMEM.MEM.SESSION.DATABLOCK.OWNIDENT01.ID">
        <SHORT-NAME>SUPPLIER</SHORT-NAME>
        <LONG-NAME>Supplier</LONG-NAME>
```

```
        <IDENT-VALUE TYPE = "A_ASCIISTRING">John Doe</IDENT-VALUE>
    </OWN-IDENT>
    <OWN-IDENT ID = "F.ECUMEM.MEM.SESSION.DATABLOCK.OWNIDENT02.ID">
        <SHORT-NAME>PARTNUMBER</SHORT-NAME>
        <LONG-NAME>Part Number</LONG-NAME>
        <IDENT-VALUE TYPE = "A_BYTEFIELD">1A</IDENT-VALUE>
    </OWN-IDENT>
</OWN-IDENTS>
```

As described above, SECURITY can be used to define security specific information (e.g. checksum, signature). In this case it is applied to a certain DATABLOCK instead of the whole SESSION.

The service and the response parameter used to read the identification of the ECU are specified by object IDENT-DESC of the referencing ECU-MEM-CONNECTOR. The SHORT-NAME of the OWN-IDENT specified by a SESSION-DESC is used to find the corresponding IDENT-DESC element (via the SHORT-NAME of its IDENT-IF sub-element) to be used for retrieving the parameter value for determining the correct SESSION-DESC.

Special data groups (SDGS) are used to store OEM-specific data not covered by the ODX data model in a structured form. They are detailed described in 7.1.2.1.

A DATABLOCK may have AUDIENCE-restrictions since not all kind of datablocks of an ECU may be reprogrammed in production of aftersales, e.g. datablocks that contain data for certain ECU-modules that are only reprogrammed in development.

All address values (FILTER-START, FILTER-END, SOURCE-START-ADDR, SOURCE-END-ADDR, POS-OFFSET, and NEG-OFFSET) shall be specified as unsigned integer values in the hexadecimal format (Schema datatype hexBinary). The D-server restricts the size of these unsigned integer values to 32 bit. All size values (FILTER-SIZE, UNCOMPRESSED-SIZE, and COMPRESSED-SIZE) shall be specified as 32 bit unsigned integer values in the decimal format (Schema datatype unsignedInt).

7.5.2.3.2 Example

The following extract of two ODX instances of the categories FLASH and DIAG-LAYER-CONTAINER demonstrates, how the request message of the service RequestDownload (0x34) of the protocol UDS can be specified:

```
<ECU-MEM ID="EM.EM1">
    <SHORT-NAME>EM_1</SHORT-NAME>
    <MEM>
        <SESSIONS>
            <SESSION ID="SESSA">
                <SHORT-NAME>A</SHORT-NAME>
                <DATABLOCK-REFS>
                    <DATABLOCK-REF ID-REF="EM.EM1.DB.Example"/>
                </DATABLOCK-REFS>
            </SESSION>
        </SESSIONS>
        <DATABLOCKS>
            <DATABLOCK ID="EM.EM1.DB.Example" TYPE="DATA">
                <SHORT-NAME>Example</SHORT-NAME>
                <LONG-NAME>DataBlock Example</LONG-NAME>
                <FLASHDATA-REF ID-REF="EM.EM1.FD.Example"/>
                <SEGMENTS>
                    <SEGMENT ID="EM.EM1.DB.Example.SG.Segm1">
                        <SHORT-NAME>Segm1</SHORT-NAME>
                        <SOURCE-START-ADDRESS>3080</SOURCE-START-ADDRESS>
                        <COMPRESSED-SIZE>1024</COMPRESSED-SIZE>
                        <UNCOMPRESSED-SIZE>2048</UNCOMPRESSED-SIZE>
```

```
                    </SEGMENT>
                </SEGMENTS>
            </DATABLOCK>
        </DATABLOCKS>
        <FLASHDATAS>
            <FLASHDATA ID="EM.EM1.FD.Example" xsi:type="EXTERN-FLASHDATA">
                <SHORT-NAME>Example</SHORT-NAME>
                <DATAFORMAT SELECTION="BINARY"/>
                <ENCRYPT-COMPRESS-METHOD TYPE="A_UINT32">33</ENCRYPT-COMPRESS-METHOD>
                <DATAFILE LATEBOUND-DATAFILE="true">*.BIN</DATAFILE>
            </FLASHDATA>
        </FLASHDATAS>
    </MEM>
</ECU-MEM>

<DATA-OBJECT-PROPS>
  <DATA-OBJECT-PROP ID="DOP_4BitLinearFactor8">
    <SHORT-NAME>DOP_4BitLinearFactor8</SHORT-NAME>
    <LONG-NAME>4 bit linear factor 8</LONG-NAME>
    <COMPU-METHOD>
      <CATEGORY>LINEAR</CATEGORY>
      <COMPU-INTERNAL-TO-PHYS>
        <COMPU-SCALES>
          <COMPU-SCALE>
            <COMPU-RATIONAL-COEFFS>
              <COMPU-NUMERATOR>
                <V>0</V>
                <V>8</V>
              </COMPU-NUMERATOR>
              <COMPU-DENOMINATOR>
                <V>1</V>
              </COMPU-DENOMINATOR>
            </COMPU-RATIONAL-COEFFS>
          </COMPU-SCALE>
        </COMPU-SCALES>
      </COMPU-INTERNAL-TO-PHYS>
    </COMPU-METHOD>
    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
      <BIT-LENGTH>4</BIT-LENGTH>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
  </DATA-OBJECT-PROP>
  <DATA-OBJECT-PROP ID="DOP_8BitIdentical">
    <SHORT-NAME>DOP_8BitIdentical</SHORT-NAME>
    <LONG-NAME>8 bit identical</LONG-NAME>
    <COMPU-METHOD>
      <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
      <BIT-LENGTH>8</BIT-LENGTH>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
  </DATA-OBJECT-PROP>
  <DATA-OBJECT-PROP ID="DOP_VarLengthOfMemoryAddress">
    <SHORT-NAME>DOP_VarLengthOfMemoryAddress</SHORT-NAME>
    <COMPU-METHOD>
      <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
```

```
    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="PARAM-LENGTH-INFO-TYPE">
      <LENGTH-KEY-REF ID-REF="REQ_RequestDownload.LengthOfMemoryAddress"/>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
  </DATA-OBJECT-PROP>
  <DATA-OBJECT-PROP ID="DOP_VarLengthOfMemorySize">
    <SHORT-NAME>DOP_VarLengthOfMemorySize</SHORT-NAME>
    <COMPU-METHOD>
      <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="PARAM-LENGTH-INFO-TYPE">
      <LENGTH-KEY-REF ID-REF="REQ_RequestDownload.LengthOfMemorySize"/>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
  </DATA-OBJECT-PROP>
</DATA-OBJECT-PROPS>

<REQUEST ID="REQ_RequestDownload">
  <SHORT-NAME>REQ_RequestDownload</SHORT-NAME>
  <LONG-NAME>RequestDownload</LONG-NAME>
  <PARAMS>
    <PARAM xsi:type="CODED-CONST" SEMANTIC="SERVICE-ID">
      <SHORT-NAME>RequestDownloadRequestServiceId</SHORT-NAME>
      <LONG-NAME>RequestDownload Request Service Id</LONG-NAME>
      <BYTE-POSITION>0</BYTE-POSITION>
      <CODED-VALUE>52</CODED-VALUE>
      <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
        <BIT-LENGTH>8</BIT-LENGTH>
      </DIAG-CODED-TYPE>
    </PARAM>
    <PARAM xsi:type="VALUE">
      <SHORT-NAME>dataFormatIdentifier</SHORT-NAME>
      <LONG-NAME>dataFormatIdentifier</LONG-NAME>
      <BYTE-POSITION>1</BYTE-POSITION>
      <DOP-REF ID-REF="DOP_8BitIdentical"/>
    </PARAM>
    <PARAM xsi:type="LENGTH-KEY" ID="REQ_RequestDownload.LengthOfMemoryAddress">
      <SHORT-NAME>LengthOfMemoryAddress</SHORT-NAME>
      <DESC><p>Multiply with 8, because bytes are given, but bits required</p></DESC>
      <BYTE-POSITION>2</BYTE-POSITION>
      <BIT-POSITION>0</BIT-POSITION>
      <DOP-REF ID-REF="DOP_4BitLinearFactor8"/>
    </PARAM>
    <PARAM xsi:type="LENGTH-KEY" ID="REQ_RequestDownload.LengthOfMemorySize">
      <SHORT-NAME>LengthOfMemorySize</SHORT-NAME>
      <DESC><p>Multiply with 8, because bytes are given, but bits required</p></DESC>
      <BYTE-POSITION>2</BYTE-POSITION>
      <BIT-POSITION>4</BIT-POSITION>
      <DOP-REF ID-REF="DOP_4BitLinearFactor8"/>
    </PARAM>
    <PARAM xsi:type="VALUE" SEMANTIC="DATA">
      <SHORT-NAME>MemoryAddress</SHORT-NAME>
      <LONG-NAME>MemoryAddress</LONG-NAME>
      <DOP-REF ID-REF="DOP_VarLengthOfMemoryAddress"/>
    </PARAM>
    <PARAM xsi:type="VALUE" SEMANTIC="DATA">
      <SHORT-NAME>MemorySize</SHORT-NAME>
      <LONG-NAME>UnCompressedMemorySize</LONG-NAME>
```

```
      <DOP-REF ID-REF="DOP_VarLengthOfMemorySize"/>
    </PARAM>
  </PARAMS>
</REQUEST>
```

The flash job shall read the actual values from the structures DATABLOCK and FLASHDATA and shall assign them to the corresponding request parameters.

Table 23 — Flash job request parameters lists the parameters of the flash job and gives example values to each of them. In the following, the encoding of these parameters in a PDU is shown.

Table 23 — Flash job request parameters

Request parameter	Element to read	Content in example
dataFormatIdentifier	ENCRYPT-COMPRESS-METHOD	0x21
LengthOfMemorySize	SIZE-LENGTH	16
LengthOfMemoryAddress	ADDRESS-LENGTH	24
MemoryAddress	SOURCE-START-ADDRESS	0x3080
MemorySize	UNCOMPRESSED-SIZE	2048 dec (0x0800)

Any parameter of type LENGTH-KEY always expects the number of bits as the physical value. Therefore, the number of bits is specified in ADDRESS-LENGTH and SIZE-LENGTH too. It would be an error to use an IDENTICAL compumethod for both LENGTH-KEY parameters to immediately pass the byte length value, because the bit length value is expected.

Because the byte position of the parameter MemorySize depends on the actual length of the parameter MemoryAddress, it is not possible to specify BYTE-POSITION of MemorySize inside the ODX document. After assignment the values to both parameters LengthOfMemorySize and LengthOfMemoryAddress, the D-server will automatically insert the values of the parameters MemoryAddress and MemorySize with the correct length and at the correct byte position into the message. The flash job has only to pass the values as an A_UINT32 and has not to bother about the actual length or position in the message.

The request message in Table 24 — Calculated request message example shall be calculated for this example.

Table 24 — Calculated request message example

[0]	[1]	[2]		[3]	[4]	[5]	[6]	[7]
0x34	0x21	0x23		0x00	0x30	0x80	0x08	0x00
RequestDownload RequestServiceId	dataFormatIdentifier	LengthOfMemorySize	LengthOfMemoryAddress	MemoryAddress			MemorySize	

7.5.2.4 Flashdata

FLASHDATA contains the flashdata as inline code (usage of element INTERN-FLASHDATA) or references a file containing it (usage of element EXTERN-FLASHDATA). In the first case, an object DATA, and in the second case, an object DATAFILE shall be defined within FLASHDATA. The data defined inline shall be represented in ASCII characters. See Figure 156 — UML representation and structure of flashdata for detailed modelling information.

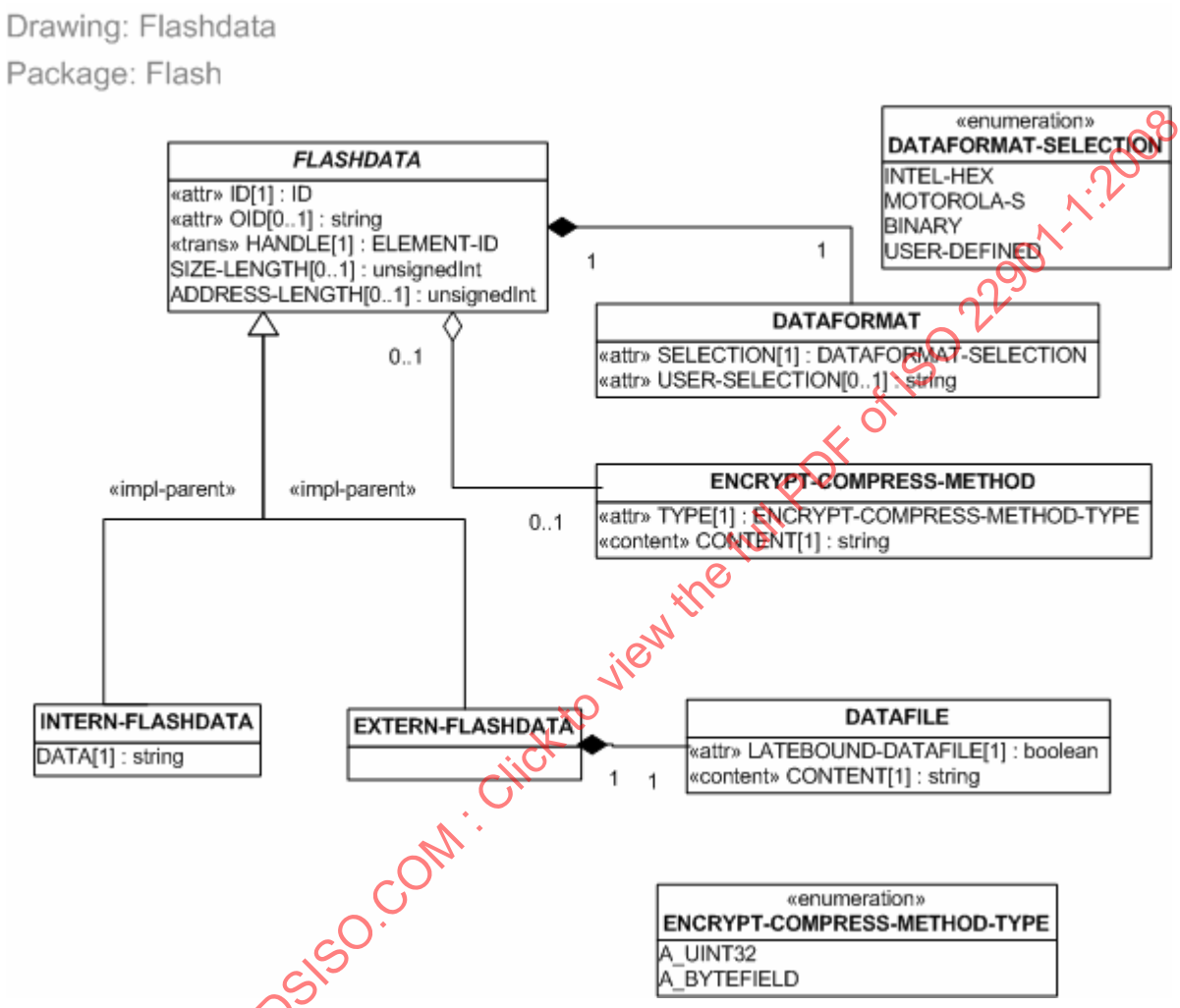


Figure 156 — UML representation and structure of flashdata

The member DATAFORMAT specifies the format in which data is represented in the current DATA or the referenced file. The possible values are MOTOROLA-S, INTEL-HEX, BINARY, or USER-DEFINED. USER-DEFINED is used for non-normative extensions. If and only if USER-DEFINED is specified, the attribute USER-SELECTION shall specify the actual format as a string value.

EXAMPLE 1 DATA in INTEL-HEX format:

```
<FLASHDATA ID="F.ECUMEM.MEM.SESSION.FLASHDATA01.ID" xsi:type="INTERN-FLASHDATA">
    <SHORT-NAME>FLASHDATA01</SHORT-NAME>
    <DATAFORMAT SELECTION="INTEL-HEX"/>
    <DATA>
        :020000020000FC
        :100000003821F64FB80000003800BF193800000052
```

```
        :100010003821F64FB80000003821F64FB800000034
        :100020003821F64FB80000003880BC81380000004D
        :100030003821F64FB80000003821F64FB800000014
        :100040003821F64FB8000000770060988000000006B
        :100050003821F64FB80000003821F64FB8000000F4
        ...
    </DATA>
</FLASHDATA>
```

EXAMPLE 2 DATA in MOTOROLA-S format:

```
<FLASHDATA ID = "F.ECUMEM.MEM.SESSION.FLASHDATA02.ID" xsi:type = "INTERN-FLASHDATA">
    <SHORT-NAME>FLASHDATA02</SHORT-NAME>
    <DATAFORMAT SELECTION = "MOTOROLA-S"/>
    <DATA>
        S0030000FC
        S113010404050607080 90A0B0C0D0E0F101112132F
        S10B01141415161718191A1B23
        S10B00FCFCFDFEFF00010203FC
        S9030000FC
    </DATA>
</FLASHDATA>
```

If BINARY is used as a type for the file format, the format of the referenced file shall be binary. Each byte of the file can be sent to the ECU without any transformation.

EXAMPLE 3 DATA in BINARY format:

```
<FLASHDATA ID = "F.ECUMEM.MEM.SESSION.FLASHDATA03.ID" xsi:type = "INTERN-FLASHDATA">
    <SHORT-NAME>FLASHDATA03</SHORT-NAME>
    <DATAFORMAT SELECTION = "BINARY"/>
   <DATA>
        000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
        202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
        404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F
        606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F
        808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F
        A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF
        C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF
        E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
    </DATA>
</FLASHDATA>
```

Each byte is encoded as 2 hexadecimal digits. There is no restriction on the maximum length. But to avoid huge lines, line breaks are allowed. Each line shall contain an even number of digits and shall not contain white spaces between the hexadecimal digits. The D-server shall convert all pairs of hexadecimal digits to a byte stream at first. This byte stream is processed identical to the content of an external binary file specified by DATAFILE, e.g. the optional FILTER is applied to this byte stream instead of the content of DATA.

Regardless of the DATAFORMAT, the content of DATA may be normalized. As the examples above show, each line can be indented. The D-server ignores any leading or trailing spaces and tabs as well as empty lines.

ENCRYPT-COMPRESS-METHOD may describe the parameter “dataFormatIdentifier” of the service “requestDownload” in ISO 14230 and ISO 14229-1(service 0x34). To describe how to interpret the values the data type is given by the member TYPE. A detailed description of this data types is given in 7.3.6.2.

Some examples are given below:

```
<ENCRYPT-COMPRESS-METHOD TYPE="A_UINT32">12</ENCRYPT-COMPRESS-METHOD>
<ENCRYPT-COMPRESS-METHOD TYPE="A_BYTEFIELD">1234</ENCRYPT-COMPRESS-METHOD>
```

In the case (TYPE="A_BYTEFIELD"), the length of the parameter results from the length of the specified value.

The optional elements SIZE-LENGTH and ADDRESS-LENGTH describe the parameters LengthOfMemorySize and LengthOfMemoryAddress, respectively, of the service "requestDownload" in ISO 14229-1(service 0x34). For details see 7.5.2.3.2.

The number of flash data often extends the size necessary to describe the meta data inside the ODX instance by some magnitudes. The XML format is not intended to hold giant data. Therefore, it is possible to describe the flashdata alternatively to DATA outside the ODX-F document in an external file which is referenced by DATAFILE. Such a flash data file should be statically (early) bound, i.e. LATEBOUND-DATAFILE is set "false". A statically bound flash data file shall be available as soon as the reference is created and should not be changed.

Other use cases require that the flash data can be changed or created until the flash job is started. Such an external flash data file shall be dynamically (late) bound, i.e. LATEBOUND-DATAFILE is set "true". In this and only this case, wildcard characters "*" and "?" can be contained inside the filename specified in DATAFILE. If exactly one file matches at runtime, it is selected automatically. If the D-server found more than one matching file, it shall ask the diagnostic application for selecting one of these files. If not any file matches, the D-server breaks the processing with an error message. After the filename is automatically or manually selected, the processing is equal to static bound flash data files.

7.5.2.5 Physical memory

The class PHYS-MEM is used to describe the physical memory layout of an ECU. This memory description may be required by the flash job to check whether logical SEGMENTs do exactly fit to the physical segments defined by PHYS-SEGMENT. For each PHYS-SEGMENT the programming job needs the members START-ADDRESS, END-ADDRESS or SIZE, BLOCKSIZE and FILLBYTE described below. A PHYS-MEM object involves the standard members SHORT-NAME, LONG-NAME, and DESC. DATABLOCKs (described above) are independent from PHYS-MEM.

See Figure 157 — UML representation of PHYS-MEM for detailed modelling information.

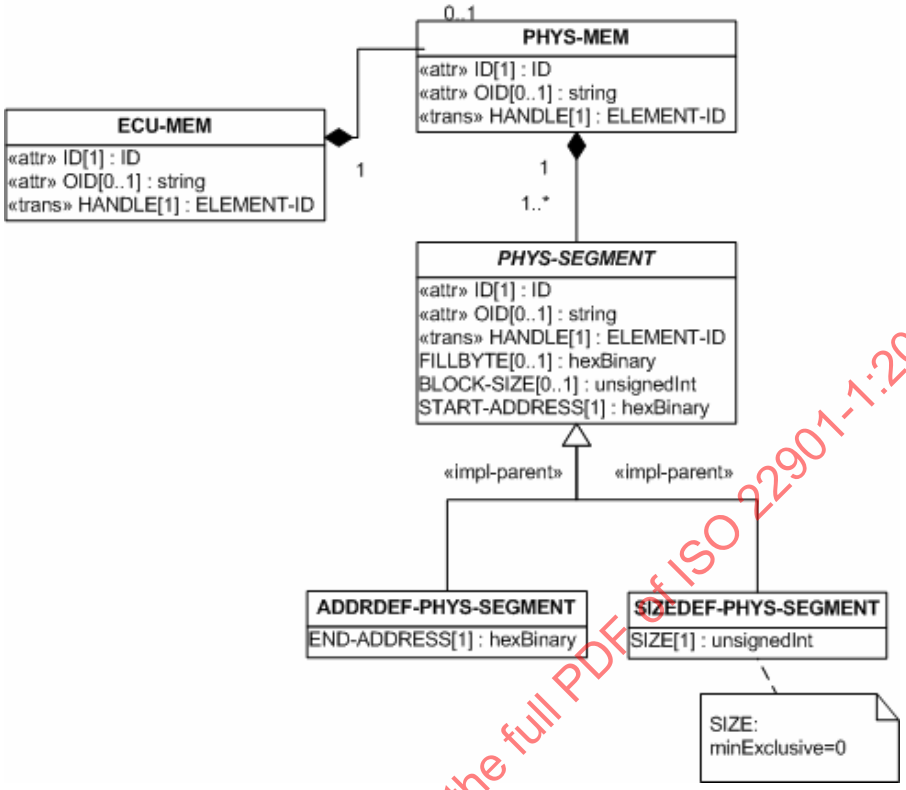


Figure 157 — UML representation of PHYS-MEM

FILLBYTE describes the byte for filling empty areas to complete the physical segment, i.e. it fills gaps between logical segments in the flash data, if necessary.

BLOCK-SIZE can be used by the flash job to enable parallel programming of memory sub-units to increase the performance of the flash process.

START-ADDRESS describes the first valid address of segment. END-ADDRESS defines the last valid address that belongs to the current segment. SIZE can be used as alternative to END-ADDRESS. It defines the size of the segment in bytes.

EXAMPLE Physical memory:

```
<PHYS-MEM ID = "F.ECUMEM.MEM.PHYSMEM01.ID">
    <SHORT-NAME>PHYSMEM01</SHORT-NAME>
    <LONG-NAME>Physical Memory Map of EXU xyz</LONG-NAME>
    <PHYS-SEGMENTS>
        <PHYS-SEGMENT ID = "F.ECUMEM.MEM.PHYSMEM01.PHYSSEG01.ID" xsi:type = "ADDRDEF-PHYS-SEGMENT">
            <SHORT-NAME>PHYSSEG01</SHORT-NAME>
            <LONG-NAME>Physical Memory Segment 01</LONG-NAME>
            <FILLBYTE>FF</FILLBYTE>
            <BLOCK-SIZE>512</BLOCK-SIZE>
            <START-ADDRESS>0000</START-ADDRESS>
            <END-ADDRESS>FFFF</END-ADDRESS>
        </PHYS-SEGMENT>
    </PHYS-SEGMENTS>
</PHYS-MEM>
```

7.5.3 ECU-MEM-CONNECTOR data model description

ECU-MEM-CONNECTOR is used to connect the ECU-MEM to the DIAG-LAYER. This part of ODX also links the ECU-MEM-objects with DIAG-COMMS e.g. used to read idents, download or upload the data. It allows an ECU to be flashed using ODX.

Figure 158 — UML representation and structure of ECU-MEM-CONNECTOR shows the structure of the ECU-MEM-CONNECTOR:

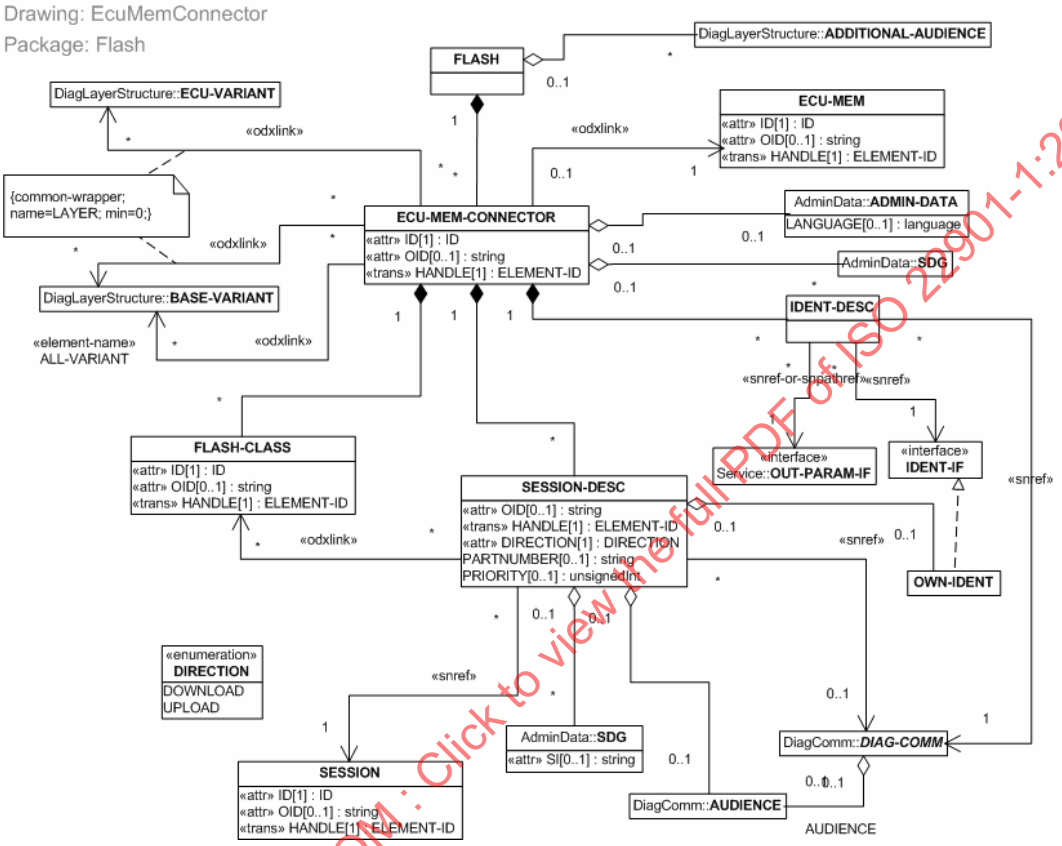


Figure 158 — UML representation and structure of ECU-MEM-CONNECTOR

At first, an ECU-MEM-CONNECTOR object involves the standard members which are described in 7.1.2. FLASH-CLASSes can be used to group sessions according to certain criteria. A FLASH-CLASS is defined by SHORT-NAME, LONG-NAME and DESC. In order to reference a class it also have the ID attribute. A link from a SESSION-DESC to a FLASH-CLASS defines the membership of the SESSION-DESC in the referenced class.

SESSION-DESCs are used to store information about sessions and jobs used for download or upload. A particular SESSION-DESC involves the following members: DIRECTION specifies whether it is a description of a download or an upload session. In accordance with that it shall be set to “DOWNLAOD” or to “UPLOAD”. SHORT-NAME, LONG-NAME, DESC and PARTNUMBER describe the SESSION-DESC object. SHORT-NAME or PARTNUMBER can alternatively be used by tools to allow the selection of concrete sessions for flashing. SHORT-NAMEs of SESSION-DESCs in all ECU-MEM-CONNECTORs shall be unique in the scope of each supported logical link as given by LAYER-REFs and ALL-VARIANTS-REFs. Alternatively or additional to PARTNUMBER the element OWN-IDENT can be used. The member PRIORITY defines the priority of the session. If more than one session is selected for programming, sessions with higher priority will be processed first. The value of PRIORITY may take values greater or equal to 0 (non-negative integers). The highest priority is indicated by the value of 0, the default value is 100.

Each SESSION-DESC refers to a SESSION from the ECU-MEM structure via SHORT-NAME. In addition, a DIAG-COMM object (flash job) in the referenced diagnostic layer is referenced by SESSION-DESC. This DIAG-COMM is responsible for download or upload of the flash data. The link DIAG-COMM-SNREF at SESSION-DESC is optional because not all applications have to use the referenced flash job. The flash logic can also be part of the application itself. Special data groups (SDGS) are used to store OEM-specific data not covered by the ODX data model in a structured form. They are described in 7.1.2.1. SESSION-DESCs can be AUDIENCE-restricted, since they might not be released for aftersales or production. Besides this restriction also the AUDIENCE restrictions at the referenced DATABLOCKs of a SESSION are evaluated i.e. some DATABLOCKs will not be programmed within a SESSION if their AUDIENCE does not match the current settings.

IDENT-DESCs include information about the read service (DIAG-COMM-SNREF) and response parameter (OUT-PARAM-IF) used to read the ECU identification. This service is executed to test the matching of the OWN-IDENTs and the EXPECTED-IDENTs inside the SESSION-DESC and the referenced ECU-MEM. The starting point for resolving the short-name-path is in case of a DIAG-SERVICE the PARAMS element within the RESPONSE of the referenced DIAG-SERVICE. In case of a SINGLE-ECU-JOB it is the OUTPUT-PARAMS element within the SINGLE-ECU-JOB.

EXPECTED-IDENTs are used to guarantee the compatibility of the ECU and the data to be flashed. A flashjob should reject the flashing if not all EXPECTED-IDENTs match. OWN-IDENTs can be used by the flashjob to detect whether it is actually necessary to flash a SESSION (inside SESSION-DESC) or DATABLOCK (inside DATABLOCK) or if the flash data are already inside the ECU. A flashjob could ignore the flashing if all OWN-IDENTs match.

Each time an EXPECTED-IDENT or an OWN-IDENT is checked, the corresponding IDENT-DESC is determined by the short name reference of IDENT-IF-SNREF to this ident. Because EXPECTED-IDENT and OWN-IDENT usually test the same identification parameters inside the ECU, the same IDENT-DESC can be used for both idents.

ECU-MEM-CONNECTOR references an ECU-MEM object and one or more BASE-VARIANT or ECU-VARIANT objects. The contained SESSION-DESCs are valid for all the ECU-VARIANTs and BASE-VARIANTs referenced via LAYER-REF and all the ECU-VARIANTs that inherit from the BASE-VARIANT referenced via ALL-VARIANT-REF excluding the BASE-VARIANT itself.

NOTE A LAYER-REF means only the DIAG-LAYER referenced, so that a reference to a BASE-VARIANT will make the ECU-MEM applicable on that BASE-VARIANT but not the ECU-VARIANTs derived from that BASE-VARIANT. To add the ECU-VARIANTs also use an ALL-VARIANTS-REF to the BASE-VARIANT as well or reference them explicitly via additional LAYER-REFs.

SESSION-DESCs and IDENT-DESCs defined in an ECU-MEM-CONNECTOR may only refer to the SESSIONs and idents of the ECU-MEM referenced by this ECU-MEM-CONNECTOR. The diagnostic layer is referenced via odxlink. It contains services or jobs needed for the entire flash process.

EXAMPLE For ECU-MEM-CONNECTOR:

```
<ECU-MEM-CONNECTOR ID = "F.ECUMEMCON01.ID">
    <SHORT-NAME>Connector1</SHORT-NAME>
    <LONG-NAME>Connector 1</LONG-NAME>
    <FLASH-CLASSS>
        <FLASH-CLASS ID = "CLASS-1">
            <SHORT-NAME>CLASS_1</SHORT-NAME>
        </FLASH-CLASS>
    </FLASH-CLASSS>
    <SESSION-DESCS>
        <SESSION-DESC DIRECTION = "DOWNLOAD">
            <SHORT-NAME>SESSION_1</SHORT-NAME>
            <PARTNUMBER>1</PARTNUMBER>
            <PRIORITY>1</PRIORITY>
            <SESSION-SNREF SHORT-NAME = "ECU_MEM_SESSION_1"/>
            <DIAG-COMM-SNREF SHORT-NAME = "FlashService"/>
```

```
            <FLASH-CLASS-REFS>
                <FLASH-CLASS-REF ID-REF = "CLASS-1"/>
            </FLASH-CLASS-REFS>
            <AUDIENCE
              IS-AFTERMARKET = "false"/>
        </SESSION-DESC>
    </SESSION-DESCS>
    <IDENT-DESCS>
        <IDENT-DESC>
            <DIAG-COMM-SNREF SHORT-NAME = "ReadIdent"/>
            <IDENT-IF-SNREF SHORT-NAME = "Ident1"/>
            <OUT-PARAM-IF-SNREF SHORT-NAME = "identValue"/>
        </IDENT-DESC>
    </IDENT-DESCS>
    <ECU-MEM-REF ID-REF = "F.ECUMEM01.ID" DOCTYPE = "FLASH"/>
    <LAYER-REFS>
        <LAYER-REF DOCTYPE = "LAYER" DOCREF = "ECUVARIANT01" ID-REF = "ECUVARIANT01.ID"/>
    </LAYER-REFS>
</ECU-MEM-CONNECTOR>
```

7.5.4 The programming process as a whole

The whole flash process can be divided into the following steps:

a) select a SESSION-DESC in ECU-MEM-CONNECTOR via SHORT-NAME (or part number);

b) read session data from ECU-MEM (it is referenced via SHORT-NAME from SESSION-DESC);

c) check whether flash-ware can be flashed (yes, if all expected idents are present); the data to flash shall be compatible with the already existing data or code in the ECU;

d) load dynamically (late) bound flash data files, if necessary;

e) load the flash job responsible for flashing in the DIAG-LAYER instance (it is referenced via SHORT-NAME from SESSION-DESC);

f) flash every data block, which belongs to the selected session, according to the defined FILTERs and SEGMENTs.

In step a), a SESSION-DESC is selected. It is an entry point into the flash process and is identified by its SHORT-NAME. In addition, it may have a PARTNUMBER, which shall be unique among all SESSION-DESCs in the ODX database. SHORT-NAME and PARTNUMBER can be alternatively used to select a SESSION-DESC, which describes the SESSION to be flashed. Only SESSION-DESC visible in the current Logical Link shall be selected. In the second step, the appropriate SESSION, referenced via SHORT-NAME from the selected SESSION-DESC, is read inside the ECU-MEM which is referred to by ECU-MEM-REF of the nesting ECU-MEM-CONNECTOR.

Step c) then occurs as follows:

— for all EXPECTED-IDENTs of the selected SESSION, determine the corresponding IDENT-DESC inside the ECU-MEM-CONNECTOR (see example in Figure 159) to get the DIAG-COMM to be executed in the current DIAG-LAYER;

— read the value using the service (or job) and the parameter referenced from IDENT-DESC: the referenced DIAG-COMM is used for reading and the parameter holds the value to be matched; if at least one IDENT-VALUE defined by the EXPECTED-IDENT object (logical OR) matches the value returned by the DIAG-COMM, the check of this EXPECTED-IDENT is successful;

— if all EXPECTED-IDENTs checks (logical AND) were successful, the flash data can be flashed.

If the check of EXPECTED-IDENTs was successful, the external flash data files shall be checked. If the filename of a dynamical bound flash data files contains wildcard characters and more than one filename matches this pattern, the D-server asks the diagnostic application for selecting one of these matching files. If the external flash data file cannot be loaded, the D-server breaks the executing of the flash process.

If the external flash data file was successfully loaded, the external specified flash job referenced from the SESSION-DESC is loaded in the next step. This job shall be visible in the current Logical Link, i.e. it shall be defined there or inherited from the upper layer.

Finally, each DATABLOCK of the SESSION is flashed in the order of its occurrence in the step f). This involves the operations described below.

— Check whether flash data already exists in the ECU. Therefore, the presence of all OWN-IDENTs is checked in the same way as the presence of EXPECTED-IDENTs was checked in step 3). The presence of all OWN-IDENTs means that the identical data is already exists in the ECU. In this case, the diagnostic system should ask whether it should overwrite the data.

— Apply FILTERs to FLASHDATA according to the description in 7.5.2.3. If no FILTER is defined in the DATABLOCK, the whole INTERN-FLASHDATA or EXTERN-FLASHDATA is used as input for the next operation.

— Apply SEGMENTs to the output of the previous operation according to the description in 7.5.2.3. It is important, that one segment represents a coherent data area, because only those can be flashed to the ECU.

— Unlock the ECU to flash all segments of the DATABLOCK.

— Write segments to the ECU in the order they are defined in the DATABLOCK or in the order of their start addresses if no SEGMENTs are defined (segments have been calculated). This order is independent of the order of flash data in any kind of flashware container. Therefore the flash job referenced from the SESSION-DESC is used.

As an example, Figure 159 — ECU-MEM-CONNECTOR as mediator between DIAG-LAYER and ECU-MEM shows an overview of ODX objects used for flashing. The SESSION-DESC SessDesc1 is selected there, which references the SESSION Sess1 in the ECU-MEM instance. DATABLOCKs referenced by Sess1 should be flashed. In this example there is only one DATABLOCK (Block1), which is flashed if all EXPECTED-IDENTs defined in the session can be found in the ECU. To check this, the IDENT-DESC objects are sought in the ECU-MEM-CONNECTOR, contains the selected SESSION-DESC SessDesc1. IdentDesc1 references for example the EXPECTED-IDENT Ident1 of this session. It also references the service CheckIdent1Service that is used for reading of a value. The read value is then compared with the IDENT-VALUEs defined in Ident1. In this example there are two values: 17 and 18. If one of them matches the read value, the check of Ident1 terminates with success. After that, Ident2 is checked in the same manner. Finally, if both checks were successful, all segments of the DATABLOCK Block1 are calculated. Since there are no SEGMENTs defined in this DATABLOCK, the whole DATA content of Data1 is used as one segment and can be flashed all at once.

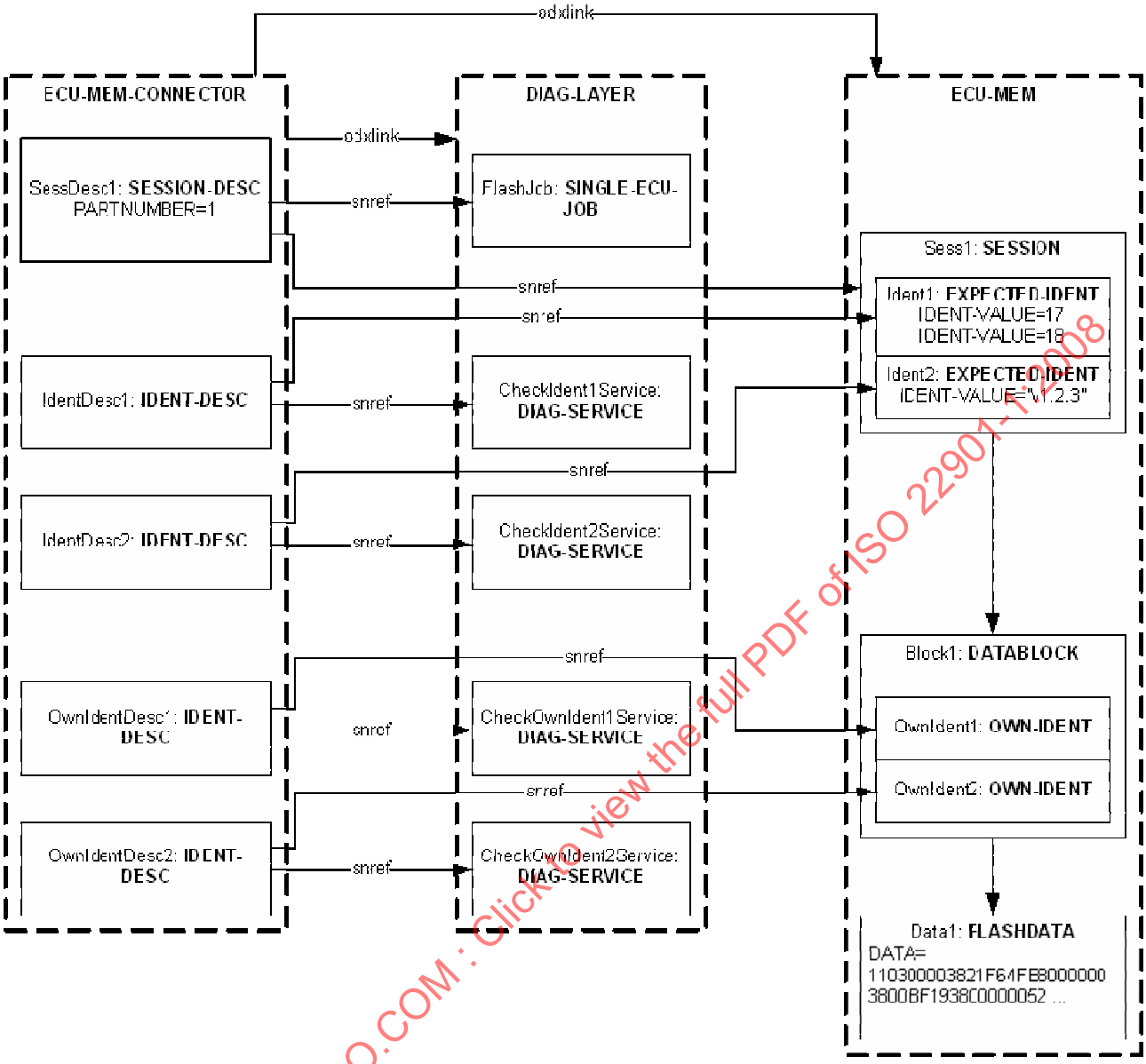


Figure 159 — ECU-MEM-CONNECTOR as mediator between DIAG-LAYER and ECU-MEM

7.5.5 The upload process as a whole

In contrast to the flash process, during the upload process, data are read from the ECU and then are written into a file.

The following restrictions are required when an upload is to be processed:

— no DATABLOCK referenced from the selected SESSION may define any FILTER objects or TARGET-ADDR-OFFSET;

— every DATABLOCK of the session shall reference a separate file (FLASHDATA with specified DATAFILE) which shall be dynamically (late) bound, i.e. LATEBOUND-DATAFILE is set “true”; before uploading starts, the file name shall be determined;

— at least one SEGMENT object shall be specified in each DATABLOCK referenced by the selected SESSION;

— the definition of idents (OWN- or EXPECTED-IDENT) is ignored by upload.

The upload process shall be performed in 4 steps:

a) select a SESSION-DESC in ECU-MEM-CONNECTOR via short-name (or part number);

b) read information about the appropriate SESSION object from ECU-MEM (it is referenced via SHORT-NAME from SESSION-DESC);

c) load the job responsible for uploading in the DIAG-LAYER instance (it is referenced via SHORT-NAME from SESSION-DESC);

d) read each DATABLOCK, which belongs to the selected session using addresses specified by the SEGMENTs, and store it in the file, specified by this DATABLOCK.

In case the data should be stored in a hex format (INTEL-HEX, MOTOROLA-S), the address information is stored within the destination file. This information complies with the address information given by SEGMENT objects of each DATABLOCK.

In the case of BINARY, no address information is stored within the destination file. Thus, the received data is stored in the file in the order of DATABLOCKs and SEGMENTs in the ODX instance.

7.6 ECU programming usage scenarios (flash)

7.6.1 ECU model description

To understand the handling of the ECU-MEM easier this subclause (7.6) explains the use of the important elements by an exemplary ECU.

This ECU model shall dispose of one microcontroller with four separate memories. The first memory area is reserved for the flash driver (8kByte boot code). The second memory area contains the program code and has a size of 120kBytes. The third and the fourth memory area are reserved for data (16kByte DataA and 16kByte DataB). Altogether there are four physically independent memory segments which could be programmed. In this example Figure 160 — Hardware and software of an exemplary ECU shows two software versions (V01 and V02) for each memory segment and two different versions of ECU hardware.

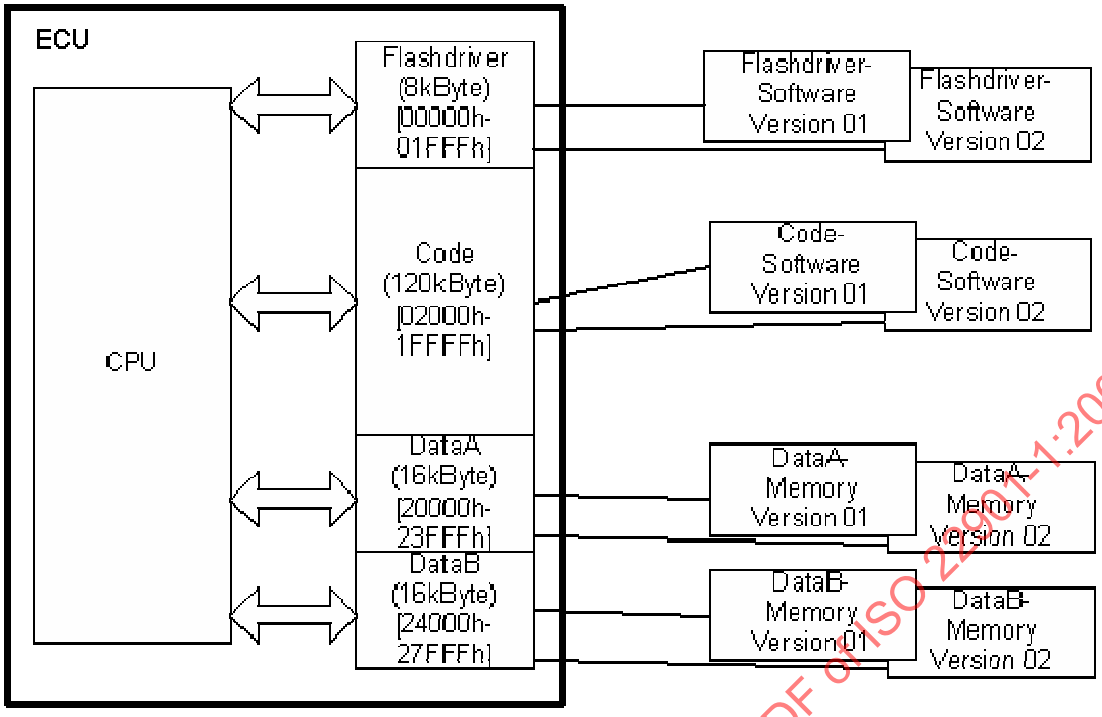


Figure 160 — Hardware and software of an exemplary ECU

Because different software versions may not be compatible to any hardware and also the content of the data memory depends on the software version of the code, all valid combinations shall be arranged in different sessions. The differentiation, which SESSION could be used with the respective ECU hardware or software, is done by the EXPECTED-IDENT elements.

Table 25 — Example for valid software combinations shows the dependencies between the ECU hardware and the possible software versions.

Table 25 — Example for valid software combinations

Combination	Hardware	Flashdriver	Code	DataA	DataB
1	Version 1	Version 1	Version 1	Version 1	Version 1
2	Version 1	Version 1	Version 1	Version 1	Version 2
3	Version 2	Version 2	Version 2	Version 1	Version 2
4	Version 2	Version 2	Version 2	Version 2	Version 2

The four possible combinations of hard- and software versions could be mapped to different flash SESSIONS. The elements EXPECTED-IDENT in each SESSION enables the tester to compare the identification values read form the ECU with the expected values of the available SESSIONs. Only the SESSION with matching identification parameters (valid flash sessions for the current ECU) shall be possible to be programmed by the flash tool.

7.6.2 ECU-MEM flash instance description

The complete file can be found in Annex F.

7.6.3 Flash session #1

The following ECU-MEM instance describes the FLASH instance for an ECU with the properties mentioned above. The first session is valid for ECUs with hardware version “V01”. All DATABLOCKs of these SESSIONs can be programmed if the ECU hardware is matching.

```
<SESSION ID="F.ECUMEM.MEM.SESSION01.ID">
<SHORT-NAME>FLASH_ECU_HW1_COMPLETE</SHORT-NAME>
<LONG-NAME>Flash ECU with hardware version 'V01' complete</LONG-NAME>
<EXPECTED-IDENTS>
    <EXPECTED-IDENT ID="F.ECUMEM.MEM.SESSION.EXPIDENT01.ID">
        <SHORT-NAME>HARDWARE_VERSION</SHORT-NAME>
        <IDENT-VALUES>
            <IDENT-VALUE TYPE="A_BYTEFIELD">¥01</IDENT-VALUE>
        </IDENT-VALUES>
    </EXPECTED-IDENT>
</EXPECTED-IDENTS>
```

Inside the SECURITYS branch a SIGNATURE for the current SESSION is defined. With the SECURITY-METHOD and the SIGNATURE the program tool or the ECU can be enabled to check the consistency and the authenticity. One possibility is to transmit the signature to the ECU after programming the memory during execution of the flash job. The ECU compares the signature with value calculated internally (e.g. by public key and flash-functions).

```
<SECURITYS>
    <SECURITY>
        <SECURITY-METHOD TYPE="A_ASCIISTRING">SECMETH_SIG01</SECURITY-METHOD>
        <FW-SIGNATURE TYPE="A_BYTEFIELD">0A5F37</FW-SIGNATURE>
    </SECURITY>
</SECURITYS>
```

The DATABLOCK-REFS contain the references to the DATABLOCKs which belong to this session. In this session four DATABLOCKs are referenced for programming all four memory segments of the ECU.

```
<DATABLOCK-REFS>
    <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK01.ID" DOCTYPE="FLASH"/>
    <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK02.ID" DOCTYPE="FLASH"/>
    <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK03.ID" DOCTYPE="FLASH"/>
    <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK04.ID" DOCTYPE="FLASH"/>
</DATABLOCK-REFS>
</SESSION>
```

7.6.4 Flash session #2

The next SESSION is only for updating the memory area reserved for DataB “V02”. Because the new data version is running on hardware “V01” only if Code and DataA is “V01” as well, three EXPECTED-IDENT elements will be needed. A flash tool shall process this SESSION only if all identifications read from the ECU do match with the values in the EXPECTED-IDENTs.

```
<SESSION ID="F.ECUMEM.MEM.SESSION02.ID">
    <SHORT-NAME>UPDATE_DATAB</SHORT-NAME>
    <LONG-NAME>Update DataB segment to version 'V02'</LONG-NAME>
    <EXPECTED-IDENTS>
        <EXPECTED-IDENT ID="F.ECUMEM.MEM.SESSION.EXPIDENT02.ID">
            <SHORT-NAME>HARDWARE_VERSION</SHORT-NAME>
            <IDENT-VALUES>
```

```
                    <IDENT-VALUE TYPE="A_BYTEFIELD">01</IDENT-VALUE>
                </IDENT-VALUES>
            </EXPECTED-IDENT>
            <EXPECTED-IDENT ID="F.ECUMEM.MEM.ESSION.EXPIDENT03.ID">
                <SHORT-NAME>CODE_VERSION</SHORT-NAME>
                <IDENT-VALUES>
                    <IDENT-VALUE TYPE="A_BYTEFIELD">01</IDENT-VALUE>
                </IDENT-VALUES>
            </EXPECTED-IDENT>
            <EXPECTED-IDENT ID="F.ECUMEM.MEM.SESSION.EXPIDENT04.ID">
                <SHORT-NAME>DATAA_VERSION</SHORT-NAME>
                <IDENT-VALUES>
                    <IDENT-VALUE TYPE="A_BYTEFIELD">01</IDENT-VALUE>
                </IDENT-VALUES>
            </EXPECTED-IDENT>
        </EXPECTED-IDENTS>
```

The data memory is not secured by a signature. Only a checksum is defined to make sure the integrity of the programmed data.

```
        <CHECKSUMS>
            <CHECKSUM ID="F.ECUMEM..MEM.SESSION.CHECKSUM01.ID">
                <SHORT-NAME>CHECKSUM_DATAB_V02</SHORT-NAME>
                <FILLBYTE>FF</FILLBYTE>
                <SOURCE-START-ADDRESS>024000</SOURCE-START-ADDRESS>
                <CHECKSUM-ALG>CHECKALGO_01</CHECKSUM-ALG>
                <SOURCE-END-ADDRESS>027FFF</SOURCE-END-ADDRESS>
                <CHECKSUM-RESULT TYPE="A_BYTEFIELD">55B0</CHECKSUM-RESULT>
            </CHECKSUM>
        </CHECKSUMS>
```

Inside DATABLOCK-REFS of this SESSION the DATABLOCK for DataB memory is referenced.

```
        <DATABLOCK-REFS>
            <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK09.ID" DOCTYPE="FLASH"/>
        </DATABLOCK-REFS>
    </SESSION>
```

7.6.5 Flash session #3

The third SESSION allows an update of the flash driver and the code memory to software version “V02”. This update is independent from the hardware version or other software components of the ECU. A flash tool might program this SESSION without compare any identifications read from the ECU.

```
    <SESSION ID="F.ECUMEM.MEM.SESSION03.ID">
        <SHORT-NAME>UPDATE_BOOT_AND_CODE_HW1</SHORT-NAME>
        <LONG-NAME>Update BOOT and CODE segments on hardware version 'V01' to software versions 'V02'</LONG-
NAME>
```

Each of the flashdriver and code memory segment need also a SECURITY element including the SIGNATURE. In this example the SIGNATURE is attached to the DATABLOCK for the code memory.

```
        <DATABLOCK-REFS>
            <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK05.ID" DOCTYPE="FLASH"/>
```

```
            <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK06.ID" DOCTYPE="FLASH"/>
        </DATABLOCK-REFS>
    </SESSION>
```

7.6.6 Flash session #4

The fourth session updates both data memory areas of an ECU with hardware version “V02” and Code version “V02”. The elements in this session are similar to the second session.

```
    <SESSION ID="F.ECUMEM.MEM.SESSION04.ID">
        <SHORT-NAME>UPDATE_DATA_HW2_COMPLETE</SHORT-NAME>
        <LONG-NAME>Update DataA and DataB segments on hardware version 'V02'  to data versions 'V02'
        </LONG-NAME>
        <EXPECTED-IDENTS>
            <EXPECTED-IDENT ID="F.ECUMEM.MEM.SESSION.EXPIDENT05.ID">
                <SHORT-NAME>HARDWARE_VERSION</SHORT-NAME>
                <IDENT-VALUES>
                    <IDENT-VALUE TYPE="A_BYTEFIELD">02</IDENT-VALUE>
                </IDENT-VALUES>
            </EXPECTED-IDENT>
            <EXPECTED-IDENT ID="F.ECUMEM.MEM.SESSION.EXPIDENT06.ID">
                <SHORT-NAME>CODE_VERSION</SHORT-NAME>
                <IDENT-VALUES>
                    <IDENT-VALUE TYPE="A_BYTEFIELD">02</IDENT-VALUE>
                </IDENT-VALUES>
            </EXPECTED-IDENT>
        </EXPECTED-IDENTS>
        <CHECKSUMS>
            <CHECKSUM ID="F.ECUMEM.MEM.SESSION.CHECKSUM02.ID">
                <SHORT-NAME>CHECKSUM_DATAA_V02</SHORT-NAME>
                <FILLBYTE>FF</FILLBYTE>
                <SOURCE-START-ADDRESS>020000</SOURCE-START-ADDRESS>
                <CHECKSUM-ALG>CHECKALGO_01</CHECKSUM-ALG>
                <SOURCE-END-ADDRESS>023FFF</SOURCE-END-ADDRESS>
                <CHECKSUM-RESULT TYPE="A_BYTEFIELD">56A9</CHECKSUM-RESULT>
            </CHECKSUM>
            <CHECKSUM ID="F.ECUMEM.MEM.SESSION.CHECKSUM03.ID">
                <SHORT-NAME>CHECKSUM_DATAB_V02</SHORT-NAME>
                <FILLBYTE>FF</FILLBYTE>
                <SOURCE-START-ADDRESS>024000</SOURCE-START-ADDRESS>
                <CHECKSUM-ALG>CHECKALGO_01</CHECKSUM-ALG>
                <SOURCE-END-ADDRESS>027FFF</SOURCE-END-ADDRESS>
                <CHECKSUM-RESULT TYPE="A_BYTEFIELD">55B0</CHECKSUM-RESULT>
            </CHECKSUM>
        </CHECKSUMS>
        <DATABLOCK-REFS>
            <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK07.ID" DOCTYPE="FLASH"/>
            <DATABLOCK-REF ID-REF="F.ECUMEM.MEM.DATABLOCK08.ID" DOCTYPE="FLASH"/>
        </DATABLOCK-REFS>
    </SESSION>
</SESSIONS>
```

7.6.7 Finalization

After the SESSIONS all referenced DATABLOCKs need to be defined inside the DATABLOCKS branch. The first DATABLOCK references the code for the flash driver version “V01”.

```
<DATABLOCKS>
    <DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK01.ID" TYPE="CODE">
        <SHORT-NAME>BOOT_V01</SHORT-NAME>
        <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA01.ID" DOCTYPE="FLASH"/>
        <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
                  IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
 </DATABLOCK>
```

The second DATABLOCK references the software for the code segment version “V01”.

```
    <DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK02.ID" TYPE="CODE">
        <SHORT-NAME>CODE_V01</SHORT-NAME>
        <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA02.ID" DOCTYPE="FLASH"/>
        <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
                  IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
    </DATABLOCK>
```

The third and the fourth DATABLOCK reference the memory for DataA and DataB segments. Because the data-source is in binary format, the element SEGMENT shall be used to inform about the target address. The element TARGET-ADDR-OFFSET is optional and represents the start address offset for programming this block on the target. BINARY code does not contain any other address information in contrast to INTEL-HEX or MOTOROLA-S format.

```
    <DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK03.ID" TYPE="DATA">
        <SHORT-NAME>DATAA_V01</SHORT-NAME>
        <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA03.ID" DOCTYPE="FLASH"/>
        <SEGMENTS>
            <SEGMENT ID="F.ECUMEM.MEM.DATABLOCK.SEGMENT01.ID">
                <SHORT-NAME>DATAA_SEG_V01</SHORT-NAME>
                <SOURCE-START-ADDRESS>00</SOURCE-START-ADDRESS>
                <UNCOMPRESSED-SIZE>8192</UNCOMPRESSED-SIZE>
            </SEGMENT>
        </SEGMENTS>
        <TARGET-ADDR-OFFSET xsi:type="POS-OFFSET">
            <POSITIVE-OFFSET>020000</POSITIVE-OFFSET>
        </TARGET-ADDR-OFFSET>
        <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
                  IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
    </DATABLOCK>
    <DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK04.ID" TYPE="DATA">
        <SHORT-NAME>DATAB_V01</SHORT-NAME>
        <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA04.ID" DOCTYPE="FLASH"/>
        <SEGMENTS>
            <SEGMENT ID="F.ECUMEM.MEM.DATABLOCK.SEGMENT02.ID">
                <SHORT-NAME>DATAB_SEG_V01</SHORT-NAME>
                <SOURCE-START-ADDRESS>00</SOURCE-START-ADDRESS>
                <UNCOMPRESSED-SIZE>8192</UNCOMPRESSED-SIZE>
            </SEGMENT>
        </SEGMENTS>
        <TARGET-ADDR-OFFSET xsi:type="POS-OFFSET">
            <POSITIVE-OFFSET>024000</POSITIVE-OFFSET>
        </TARGET-ADDR-OFFSET>
```

```
    <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
              IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
</DATABLOCK>
```

DATABLOCK five has the same structure as DATABLOCK one and references the flash driver code version “V02”.

```
<DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK05.ID" TYPE="CODE">
    <SHORT-NAME>BOOT_V02</SHORT-NAME>
    <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA05.ID" DOCTYPE="FLASH"/>
</DATABLOCK>
```

The sixth DATABLOCK reference the code memory version “V02”. Because this DATABLOCK is only used in a SESSION for updating the memory it is not necessary to replace the complete memory segments. With the filters only the different memory areas will masked from the source and transferred to the ECU. In this DATABLOCK only the memory beginning form address 0x2000 up to 0x17FFF need to be changed. An additional security element is assigned to the DATABLOCK, which includes the SIGNATURE for the part of code.

```
<DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK06.ID" TYPE="CODE">
    <SHORT-NAME>CODE_V02</SHORT-NAME>
    <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA06.ID" DOCTYPE="FLASH"/>
    <FILTERS>
        <FILTER xsi:type="ADDRDEF-FILTER">
            <FILTER-START>2000</FILTER-START>
            <FILTER-END>017FFF</FILTER-END>
        </FILTER>
    </FILTERS>
    <SECURITYS>
        <SECURITY>
            <SECURITY-METHOD TYPE="A_ASCIISTRING">SECMETH_SIG01</SECURITY-METHOD>
            <FW-SIGNATURE TYPE="A_BYTEFIELD">1F62BA</FW-SIGNATURE>
        </SECURITY>
    </SECURITYS>
    <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
              IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
</DATABLOCK>
```

The next two DATABLOCKs have the same structure like the other DATABLOCKs for DataA and DataB defined before. The difference between these DATABLOCKs is the new version (“V02” instead “V01”).

```
<DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK07.ID" TYPE="DATA">
    <SHORT-NAME>DATAA_V02</SHORT-NAME>
    <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA07.ID" DOCTYPE="FLASH"/>
    <SEGMENTS>
        <SEGMENT ID="F.ECUMEM.MEM.DATABLOCK.SEGMENT03.ID">
            <SHORT-NAME>DATAA_SEG_V02</SHORT-NAME>
            <SOURCE-START-ADDRESS>00</SOURCE-START-ADDRESS>
            <UNCOMPRESSED-SIZE>8192</UNCOMPRESSED-SIZE>
        </SEGMENT>
    </SEGMENTS>
    <TARGET-ADDR-OFFSET xsi:type="POS-OFFSET">
        <POSITIVE-OFFSET>020000</POSITIVE-OFFSET>
    </TARGET-ADDR-OFFSET>
    <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
              IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
```

```
</DATABLOCK>
<DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK08.ID" TYPE="DATA">
    <SHORT-NAME>DATAB_V02</SHORT-NAME>
    <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA08.ID" DOCTYPE="FLASH"/>
    <SEGMENTS>
        <SEGMENT ID="F.ECUMEM.MEM.DATABLOCK.SEGMENT04.ID">
            <SHORT-NAME>DATAB_SEG_V02</SHORT-NAME>
            <SOURCE-START-ADDRESS>00</SOURCE-START-ADDRESS>
            <UNCOMPRESSED-SIZE>8192</UNCOMPRESSED-SIZE>
        </SEGMENT>
    </SEGMENTS>
    <TARGET-ADDR-OFFSET xsi:type="POS-OFFSET">
        <POSITIVE-OFFSET>024000</POSITIVE-OFFSET>
    </TARGET-ADDR-OFFSET>
    <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
              IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
</DATABLOCK>
```

The last DATABLOCK is used by the second SESSION for updating DataB defined above. Because this SESSION is also an update, it is not be necessary to exchange the contents of the whole memory segment. The element <SOURCE-START-ADDRESS> is used as pointer in the data source and represents the beginning of the data, which need to be changed. The element <POSITIVE-OFFSET> is used as pointer in the target device memory.

```
<DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK07.ID" TYPE="DATA">
    <SHORT-NAME>DATAA_V02</SHORT-NAME>
    <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA07.ID" DOCTYPE="FLASH"/>
    <SEGMENTS>
        <SEGMENT ID="F.ECUMEM.MEM.DATABLOCK.SEGMENT03.ID">
            <SHORT-NAME>DATAA_SEG_V02</SHORT-NAME>
            <SOURCE-START-ADDRESS>00</SOURCE-START-ADDRESS>
            <UNCOMPRESSED-SIZE>8192</UNCOMPRESSED-SIZE>
        </SEGMENT>
    </SEGMENTS>
    <TARGET-ADDR-OFFSET xsi:type="POS-OFFSET">
        <POSITIVE-OFFSET>020000</POSITIVE-OFFSET>
    </TARGET-ADDR-OFFSET>
    <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
              IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
</DATABLOCK>
<DATABLOCK ID="F.ECUMEM.MEM.DATABLOCK08.ID" TYPE="DATA">
    <SHORT-NAME>DATAB_V02</SHORT-NAME>
    <FLASHDATA-REF ID-REF="F.ECUMEM.MEM.FLASHDATA08.ID" DOCTYPE="FLASH"/>
    <SEGMENTS>
        <SEGMENT ID="F.ECUMEM.MEM.DATABLOCK.SEGMENT04.ID">
            <SHORT-NAME>DATAB_SEG_V02</SHORT-NAME>
            <SOURCE-START-ADDRESS>00</SOURCE-START-ADDRESS>
            <UNCOMPRESSED-SIZE>8192</UNCOMPRESSED-SIZE>
        </SEGMENT>
    </SEGMENTS>
    <TARGET-ADDR-OFFSET xsi:type="POS-OFFSET">
        <POSITIVE-OFFSET>024000</POSITIVE-OFFSET>
    </TARGET-ADDR-OFFSET>
    <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
              IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
</DATABLOCK>
```

After the DATABLOCKS the FLASHDATAS need to be defined. As described before the source format for data memory is BINARY. The code area and the flash driver should be INTEL-HEX format in this example.

The first FLASHDATA block contains the memory for the flash driver - version “V01”:

```
        <FLASHDATAS>
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA01.ID" xsi:type="INTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_BOOT_V01</SHORT-NAME>
                <DATAFORMAT SELECTION="INTEL-HEX"/>
                <DATA>
                 :020000020000FC
                 :100000003821F64FB80000003800BF193800000052
                 :100010003821F64FB80000003821F64FB800000034
...
                </DATA>
            </FLASHDATA>
```

The second FLASHDATA block contains the memory for the code - version “V01”:

```
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA02.ID" xsi:type="INTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_CODE_V01</SHORT-NAME>
                <DATAFORMAT SELECTION="INTEL-HEX"/>
                <DATA>
                 :020000020000FC
                 :100000003821F64FB80000003800BF193800000052
                 :100010003821F64FB80000003821F64FB800000034
...
                </DATA>
            </FLASHDATA>
```

The third block contains the data for DataA in BINARY format - version “V01”. Because the binary data can be encrypted in the source (i.e. for engine control modules) an additional element ENCRYPT-COMPRESS-METHOD is necessary for the encryption in the flash tool or ECU:

```
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA03.ID" xsi:type="INTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_DATAA_V01</SHORT-NAME>
                <DATAFORMAT SELECTION="BINARY"/>
                <ENCRYPT-COMPRESS-METHOD TYPE="A_BYTEFIELD">4A</ENCRYPT-COMPRESS-METHOD>
                <DATA>
                 110300003821F64FB80000003800BF193800000052
                 113010003821F64FB80000003821F64FB800000034
                 113020003821F64FB80000003880BC81380000004D
...
                </DATA>
            </FLASHDATA>
```

The fourth block looks similar to block three and contains the data for the DataB segment - version “V01”.

```
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA04.ID" xsi:type="INTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_DATAB_V01</SHORT-NAME>
                <DATAFORMAT SELECTION="BINARY"/>
                <ENCRYPT-COMPRESS-METHOD TYPE="A_BYTEFIELD">4A</ENCRYPT-COMPRESS-METHOD>
                <DATA>
                 110300003821F64FB80000003800BF193800000052
                 113010003821F64FB80000003821F64FB800000034
                 113020003821F64FB80000003880BC81380000004D
```

```
...
                </DATA>
        </FLASHDATA>
```

It is not necessary to include the whole data and code inside the FLASHDATA elements. External source files can be referenced by the filename. This is done for the FLASHDATA at version “V02”.

```
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA05.ID" xsi:type="EXTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_BOOT_V02</SHORT-NAME>
                <DATAFORMAT SELECTION="INTEL-HEX"/>
                <DATAFILE LATEBOUND-DATAFILE="true">FLASHDATA_01.hex</DATAFILE>
            </FLASHDATA>
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA06.ID" xsi:type="EXTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_CODE_V02</SHORT-NAME>
                <DATAFORMAT SELECTION="INTEL-HEX"/>
                <DATAFILE LATEBOUND-DATAFILE="true">FLASHDATA_02.hex</DATAFILE>
            </FLASHDATA>
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA07.ID" xsi:type="EXTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_DATAA_V02</SHORT-NAME>
                <DATAFORMAT SELECTION="BINARY"/>
                <ENCRYPT-COMPRESS-METHOD TYPE="A_BYTEFIELD">4A</ENCRYPT-COMPRESS-METHOD>
                <DATAFILE LATEBOUND-DATAFILE="true">FLASHDATA_03.bin</DATAFILE>
            </FLASHDATA>
            <FLASHDATA ID="F.ECUMEM.MEM.FLASHDATA08.ID" xsi:type="EXTERN-FLASHDATA">
                <SHORT-NAME>FLASHDATA_DATAB_V02</SHORT-NAME>
                <DATAFORMAT SELECTION="BINARY"/>
                <ENCRYPT-COMPRESS-METHOD TYPE="A_BYTEFIELD">4A</ENCRYPT-COMPRESS-METHOD>
                <DATAFILE LATEBOUND-DATAFILE="true">FLASHDATA_04.bin</DATAFILE>
            </FLASHDATA>
        </FLASHDATAS>
    </MEM>
```

To advertise the physical memory segments to a programming tool the PHYS-MEM element could be used. All four memory segments of the ECU need to be defined here as shown in Figure 160.

```
    <PHYS-MEM ID="F.ECUMEM.PHYSMEM01.ID">
        <SHORT-NAME>A</SHORT-NAME>
        <PHYS-SEGMENTS>
```

Physical memory segment for the flash driver (0x00000 - 0x01FFF):

```
            <PHYS-SEGMENT ID="F.ECUMEM.PHYSMEM.PHYSSEG01.ID" xsi:type="ADDRDEF-PHYS-SEGMENT">
                <SHORT-NAME>BOOT_SEGMENT</SHORT-NAME>
                <FILLBYTE>FF</FILLBYTE>
                <BLOCK-SIZE>1024</BLOCK-SIZE>
                <START-ADDRESS>00</START-ADDRESS>
                <END-ADDRESS>1FFF</END-ADDRESS>
            </PHYS-SEGMENT>
```

Physical memory segment for the code (0x02000 - 0x1FFFF):

```
            <PHYS-SEGMENT ID="F.ECUMEM.PHYSMEM.PHYSSEG02.ID" xsi:type="ADDRDEF-PHYS-SEGMENT">
                <SHORT-NAME>CODE_SEGMENT</SHORT-NAME>
                <FILLBYTE>FF</FILLBYTE>
                <BLOCK-SIZE>1024</BLOCK-SIZE>
```

```
            <START-ADDRESS>2000</START-ADDRESS>
            <END-ADDRESS>01FFFF</END-ADDRESS>
        </PHYS-SEGMENT>
```

Physical memory segment for DataA (0x20000 - 0x23FFF):

```
        <PHYS-SEGMENT ID="F.ECUMEM.PHYSMEM.PHYSSEG03.ID" xsi:type="SIZEDEF-PHYS-SEGMENT">
            <SHORT-NAME>DATAA_SEGMENT</SHORT-NAME>
            <FILLBYTE>00</FILLBYTE>
            <BLOCK-SIZE>512</BLOCK-SIZE>
            <START-ADDRESS>020000</START-ADDRESS>
            <SIZE>8192</SIZE>
        </PHYS-SEGMENT>
```

Physical memory segment for the DataB (0x24000 - 0x27FFF):

```
        <PHYS-SEGMENT ID="F.ECUMEM.PHYSMEM.PHYSSEG04.ID" xsi:type="SIZEDEF-PHYS-SEGMENT">
            <SHORT-NAME>DATAB_SEGMENT</SHORT-NAME>
            <FILLBYTE>00</FILLBYTE>
            <BLOCK-SIZE>512</BLOCK-SIZE>
            <START-ADDRESS>024000</START-ADDRESS>
            <SIZE>8192</SIZE>
        </PHYS-SEGMENT>
    </PHYS-SEGMENTS>
</PHYS-MEM>
</ECU-MEM></ECU-MEMS>
```

An important branch inside the FLASH container is the ECU-MEM-CONNECTORS. All SESSIONs defined above are referenced here together with a link to the required flash job (DIAG-COMM object).

```
<ECU-MEM-CONNECTORS>
    <ECU-MEM-CONNECTOR ID="F.ECUMEMCON01.ID">
        <SHORT-NAME>FLASH_ECU_Connector</SHORT-NAME>
        <SESSION-DESCS>
```

The first SESSION shall be used for a complete download of all software versions “V01”. This job also has the highest PRIORITY (0). This means, that this job is executed first, if more than one job is selected.

```
            <SESSION-DESC DIRECTION="DOWNLOAD">
                <SHORT-NAME>DOWNLOAD_SOFTWARE_V01_COMPLETE</SHORT-NAME>
                <PARTNUMBER>FLASH_ECU_123</PARTNUMBER>
                <PRIORITY>0</PRIORITY>
                <SESSION-SNREF SHORT-NAME="FLASH_ECU_HW1_COMPLETE"/>
                <DIAG-COMM-SNREF SHORT-NAME="DIAGCOMM_FLASH_JOB_01"/>
                <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
                          IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
            </SESSION-DESC>
```

The second SESSION is only be used for the DataB memory update to version “V02”. This requires a different flash-job because data has different security mechanism and only one of the four memory segments need to be programmed. For the update of data memory a lower priority is defined.

```
            <SESSION-DESC DIRECTION="DOWNLOAD">
                <SHORT-NAME>DOWNLOAD_UPDATE_DATAB_V02</SHORT-NAME>
```

```
            <PARTNUMBER>FLASH_ECU_123</PARTNUMBER>
            <PRIORITY>50</PRIORITY>
            <SESSION-SNREF SHORT-NAME="UPDATE_DATAB"/>
            <DIAG-COMM-SNREF SHORT-NAME="DIAGCOMM_FLASH_JOB_02"/>
            <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
                      IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
        </SESSION-DESC>
```

The update of flash driver and code memory with version “V02” requires a further flash-job.

```
        <SESSION-DESC DIRECTION="DOWNLOAD">
            <SHORT-NAME>DOWNLOAD_UPDATE_BOOT_AND_CODE_V02</SHORT-NAME>
            <PARTNUMBER>FLASH_ECU_123</PARTNUMBER>
            <PRIORITY>10</PRIORITY>
            <SESSION-SNREF SHORT-NAME="UPDATE_BOOT_AND_CODE_HW1"/>
            <DIAG-COMM-SNREF SHORT-NAME="DIAGCOMM_FLASH_JOB_03"/>
            <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
                      IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
        </SESSION-DESC>
```

The download of DataA and DataB segment requires a new job too.

```
        <SESSION-DESC DIRECTION="DOWNLOAD">
            <SHORT-NAME>DOWNLOAD_UPDATE_DATAA_AND_DATAB_V02</SHORT-NAME>
            <PARTNUMBER>FLASH_ECU_123</PARTNUMBER>
            <PRIORITY>40</PRIORITY>
            <SESSION-SNREF SHORT-NAME="UPDATE_DATA_HW2_COMPLETE"/>
            <DIAG-COMM-SNREF SHORT-NAME="DIAGCOMM_FLASH_ECU_JOB_04"/>
            <AUDIENCE IS-MANUFACTURING="true" IS-SUPPLIER="true" IS-AFTERSALES="true"
                      IS-DEVELOPMENT="true" IS-AFTERMARKET="true"/>
        </SESSION-DESC>
    </SESSION-DESCS>
    <ECU-MEM-REF ID-REF="F.ECUMEM01.ID" DOCTYPE="FLASH"/>
    <LAYER-REFS>
        <LAYER-REF DOCTYPE="LAYER" DOCREF="ECUVARIANT01" ID-REF="ECUVARIANT01.ID"/>
    </LAYER-REFS>
  </ECU-MEM-CONNECTOR>
</ECU-MEM-CONNECTORS>
</FLASH>
```

Flash jobs can be reused in SESSION-DESCs if the sequence of diagnostic services and the security mechanisms for up- or download in the respective SESSIONs is equal.

It is also possible to describe multiple processor ECUs within one flash container. If the different memory areas of the processors inside the ECU can be programmed by using one continuous address space all SESSIONs, DATABLOCKs, FLASHDATAs and PHYSMEMs can be defined in only one ECU-MEM object.

To support more than one ECU with different physical memory structures or ECUs with multiple processors and overlapping memory areas multiple ECU-MEMs and ECU-MEM-CONNECTORS shall be defined in one flash container instance.

7.7 ECU variant coding usage scenarios

7.7.1 Overview

In the automotive electronics industry, the term “Variant Coding” describes a procedure that allows the ECU software to be adapted to a vehicle-specific (optional components, equipments and instrumentation) or a localization specific (e.g. country) environment. “Variant Coding” of an ECU is typically performed during vehicle or ECU production, in the aftermarket (upgrading vehicle equipment) and in a service environment (replacement of defective parts or equipment). For the purposes of this part of ISO 22901, the term “ECU software configuration” or “ECU configuration” in brief will be used to describe this process, because this term describes more accurately what is done during this procedure. To this end, the ECU configuration part of ODX provides a set of data elements that allow modifying subsections of the ECU's memory using diagnostic communication i.e. diagnostic services. That means ODX contains on one hand *predefined data sets* that can be directly used for configuration as well as the description of their *substructure and conversion methods* in order to change parts of the data dynamically during runtime of the diagnostic system driven by the ODX data.

The following use cases are typically handled via ECU configuration and are covered by the ODX data model:

a) adjustment of an ECU according to a specific vehicle environment;

b) country specific configuration;

c) setting of characteristic curves;

d) enabling/disabling of optional functionality.

Even though the most obvious use case for the described model is to *write* configuration data *to* an ECU, also *reading* of configuration data *from* an ECU is supported.

7.7.2 General modelling concepts

The central element of the ODX ECU configuration concept is the CONFIG-RECORD. A CONFIG-RECORD object acts as a notional memory buffer, which corresponds to a contiguous section of the ECU’s permanent memory. A CONFIG-RECORD connects the data and their description on the one hand and the ECU’s permanent memory and the diagnostic services to manipulate these data on the other hand. Like most other ODX data, configuration data can also be described in either physical or coded representation or even a combination of both.

Each CONFIG-RECORD describes its own notional buffer with a fictive start address of zero. The address in the ECU’s permanent memory is defined at DIAG-COMM-DATA-CONNECTOR/WRITE-SERVICE/PHYS-CONSTANT-VALUE[22)] which can also be used for definition of DIDs (data identifier like local ID) if appropriate.

Several CONFIG-RECORDs can be aggregated into a CONFIG-DATA element. The CONFIG-DATA is considered to contain the Configuration data description for a whole ECU. Several CONFIG-DATAs can be aggregated in an ECU-CONFIG instance. ECU-CONFIG is a separate ODX CATEGORY (like FLASH or DIAG-LAYER-CONTAINER) [23)].

22) The same applies for READ-SERVICE.

23) The aggregation of CONFIG-RECORD objects within a CONFIG-DATA object does *not* convey any further meaning, such as ordering of the notional memory buffers or read/write sequence.

Figure 161 — ECU configuration overview illustrates above description.

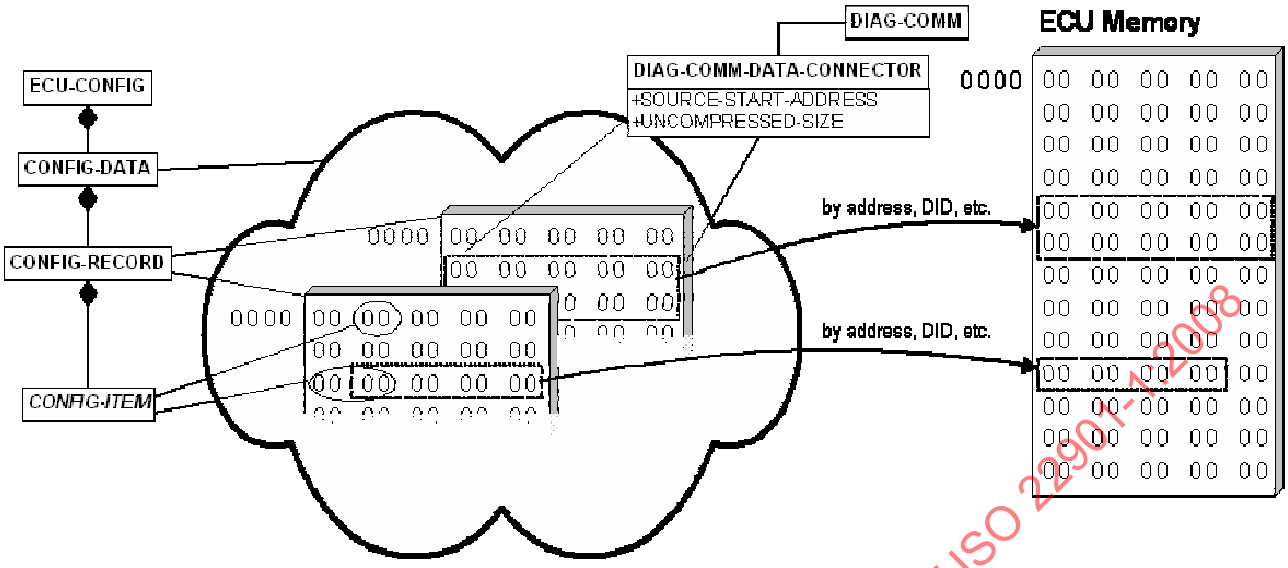


Figure 161 — ECU configuration overview[24)]

7.8 ODX data model for ECU configuration

7.8.1 Description of the ECU configuration data model

7.8.1.1 Overview

The data for ECU configuration can be described in two different ways:

a) blocks of binary data (DATA-RECORD);

b) fine grained data items that correspond to single functions of the ECU (CONFIG-ITEM, described by ITEM-VALUE if value can be changed at runtime).

7.8.1.2 Describing blocks of binary data

A CONFIG-RECORD object can optionally contain one or many DATA-RECORD objects. A DATA-RECORD object is a contiguous block of binary data. Its size shall be equal to the size of the notional memory buffer of the CONFIG-RECORD. Each DATA-RECORD object can be identified by its IDENT-VALUE (element name: DATA-ID). Each DATA-RECORD object represents *one alternative content* of the CONFIG-RECORD object it belongs to. The decision about which DATA-RECORD to use in an actual ECU configuration event shall be handled outside of ODX.

24) The UML model elements in this figure are simplified. See 7.8.4 for the comprehensive model.

Figure 162 — Using blocks of binary data via DATA-RECORD illustrates above description.

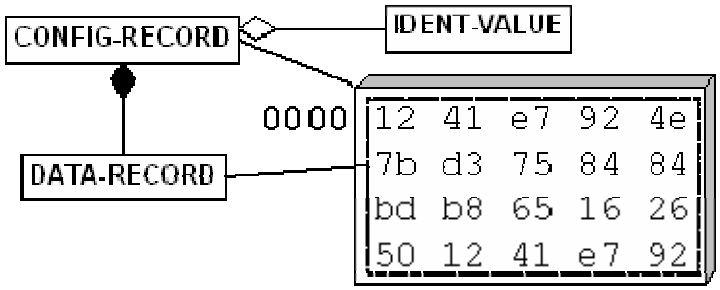


Figure 162 — Using blocks of binary data via DATA-RECORD[25)]

A DATA-RECORD can only describe coded values. The conversion of these binary data to physical values (and vice versa) is possible by using the fine grain ECU configuration data description by the CONFIG-ITEM objects (see next paragraph).

In a runtime environment, a user may be given the choice to select one of the DATA-RECORDs defined in the respective ODX file by selecting its IDENT-VALUE (DATA-ID) and to use this block data in an ECU configuration programming event.

7.8.1.3 Describing individual configuration items

A CONFIG-RECORD object may optionally be described via so-called CONFIG-ITEM elements. A CONFIG-ITEM may be thought of as a bit or a number of contiguous bits, enabling or disabling a specific ECU function or setting a value for use by some ECU function(s). The entirety of the notional memory may be defined by CONFIG-ITEM[26)] elements. These describe the conversion from a physical value to a coded value that is than stored in the notional memory buffer. Thus, CONFIG-ITEMS relay to DATA-OBJECT-PROPERTIES (DOPs) i.e. in a runtime environment an external process may supply a physical value for each of the CONFIG-ITEM elements which will be converted into the coded representation and stored in a certain section of the notional memory buffer. These sections shall not overlap. Alternatively, an OPTION-ITEM element can also have a PHYSICAL-DEFAULT-VALUE, which will be used in case no actual physical value is being supplied.

25) The UML model elements in this figure are simplified. See 7.8.4 for the comprehensive model. For more information on how to specify binary data blocks using the DATA element, see also 6.5.2.4 (FLASHDATA).

26) CONFIG-ITEM is a generalization of some specialized model elements. See Figure 167 — UML representation and structure of ECU configuration items for details.

Figure 163 — Using CONFIG-ITEMs illustrates above description.

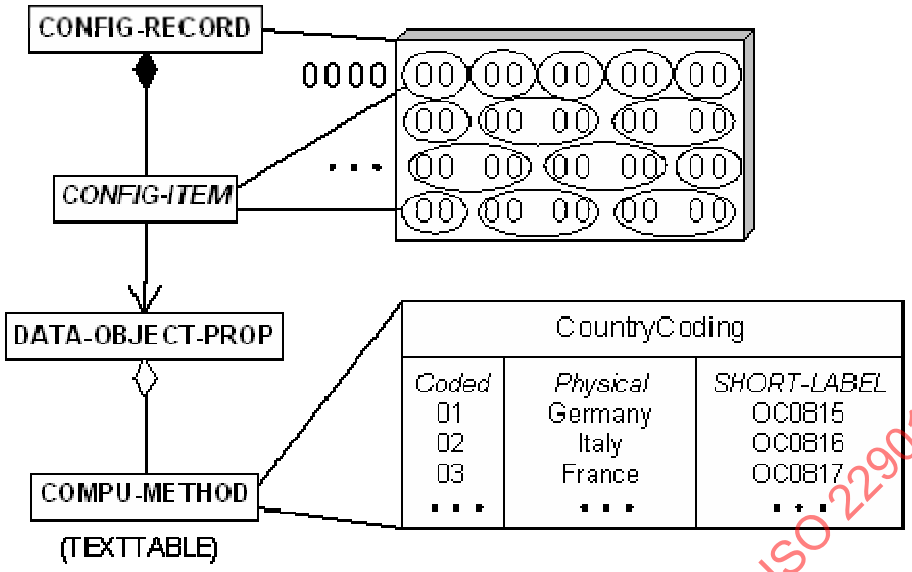


Figure 163 — Using CONFIG-ITEMs[27)]

There are four specializations of CONFIG-ITEM, as described below.

— OPTION-ITEM
This describes a e.g. function related configuration element of the ECU. The majority of CONFIG-ITEMs will be of this type. If for a certain use-case option codes are required to identify the various configuration options, this option code will be put in the SHORT-LABEL element of the corresponding COMPU-SCALE of the TEXT-TABLE being used as COMPU-METHOD in the DOP that is linked to the CONFIG-ITEM. All DATA-OBJECT-PROPs defined inside an ECU-CONFIG instance are referenced via DATA-OBJECT-PROP-SNREF. DATA-OBJECT-PROPs defined in a DIAG-LAYER-CONTAINER will be referenced via DATA-OBJECT-PROP-REF.

— SYSTEM-ITEM
This describes an instruction to the D-server to write a data element (e.g. the tester ID or the current data) to the ECU's programmable memory. This is in analogy to the SYSTEM parameter, which is described in 7.3.5.4. See also Table A.4 — Enumeration of SYSPARAM at SYSTEM for a list of predefined SYSPARAM values.

— CONFIG-ID-ITEM
There exists at most one CONFIG-ID-ITEM for every CONFIG-RECORD, which is used to define how the *identification* of that CONFIG-RECORD (given by its IDENT-VALUE/CONFIG-ID) is being located in the notional memory buffer. Using this method the identification can be written to the ECUs memory. When reading data from the ECU, this can be used to identify the matching CONFIG-RECORD in the ODX data.

— DATA-ID-ITEM
There exists at most one DATA-ID-ITEM per CONFIG-RECORD, which is used to define how the identification of the selected DATA-RECORD (given by its IDENT-VALUE/DATA-ID) is being converted to or from the notional memory buffer. Using this method the identification can be written to the ECUs memory. When reading data from the ECU, this can be used to identify the matching DATA-RECORD in the ODX data.

27) The UML model elements in this figure are simplified. See 7.8.4 for the comprehensive model.

7.8.1.4 Description of configuration values

An alternative to the use of a TEXTTABLE for the description of configuration values is the definition of ITEM-VALUES. They allow specifying the potential values of an OPTION-ITEM with some additional information:

— PHYS-CONSTANT-VALUE: contains the value to be converted by the related DOP;

— MEANING: a short description of the consequences to the ECU's functionality if this value is selected;

— KEY: a unique key value to identify this ITEM-VALUE;

— RULE: a placeholder for a kind of expression to determine if this value is valid for a certain vehicle configuration;

— DESCRIPTION: Offers an additional opportunity to describe the functionality of this certain value if selected;

— AUDIENCE: Access to a certain ITEM-VALUE can be restricted to specific user groups.

NOTE The PHYS-CONSTANT-VALUE is also mapped against the DATA-OBJECT-PROP referenced by the OPTION-ITEM.

7.8.1.5 Combining binary data blocks with ECU functions

The two methods described above can be combined in a straightforward manner as shown below. The binary data supplied by one of the DATA-RECORD object will initialize the notional memory buffer in a first step. Then one or more of the CONFIG-ITEM (type OPTION-ITEM) elements of the CONFIG-RECORD together with some PHYSICAL-DEFAULT-VALUEs or user supplied values can modify the memory buffer under control of a user. See Figure 164 — Combining CONFIG-ITEM and DATA-RECORD.

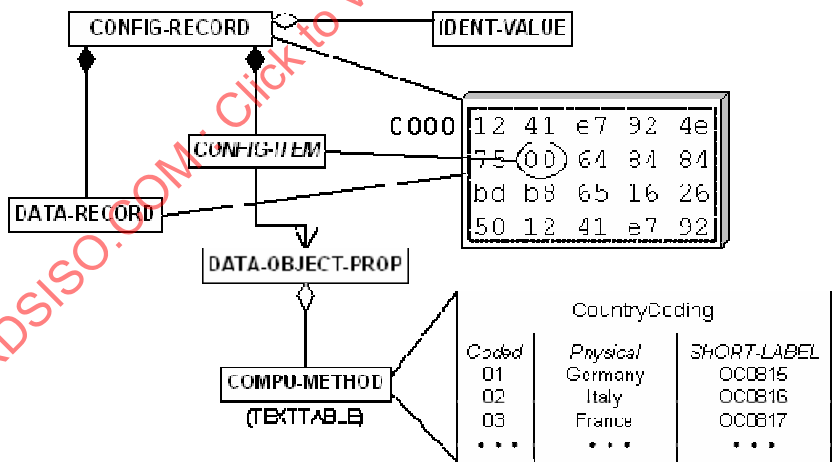


Figure 164 — Combining CONFIG-ITEM and DATA-RECORD[28)]

7.8.2 Reading and writing configuration data from and to the ECU

Figure 165 shows the general idea how to write configuration data to an ECU using a KWP2000 WriteMemoryByAddress service as an example. First, a runtime environment needs to identify the services to

28) The UML model elements in this figure are simplified. See 7.8.4 for the comprehensive model.

use for writing configuration data (in the simplified figure the WRITE-SERVICE). The D-server provides it with a cut-out of the notional memory buffer (defined by SOURCE-START-ADDRESS and UNCOMPRESSED-SIZE) through setting a certain parameter. Other PARAMs of the DIAG-COMM are set to predefined values. In the example shown the memoryAddress is set to 0x14. As all other parameters of WriteMemoryByAddress can be defined as constant values, in the example, all data necessary to execute the service has been defined by the configuration record and the DIAG-COMM-DATA-CONNECTOR. The predefined values are defined in the context of the DIAG-COMM-DATA-CONNECTOR as physical constant values. They are not limited to just memory addresses but can also be a DID number, PID number or similar.

In the event, that the WRITE-SERVICE cannot transfer the complete CONFIG-RECORD in one single programming event, the CONFIG-RECORD can be subdivided into several DIAG-COMM-DATA-CONNECTORS. For the purpose of ECU configuration, several DIAG-COMM-DATA-CONNECTORSs are used solely to split up a CONFIG-RECORD into smaller chunks of data, that can be used as a PARAM in the PDU of the read or write service. For that reason, the order of the DIAG-COMM-DATA-CONNECTORSs shall be defined.

To read the data from the ECU the DIAG-COMM-DATA-CONNECTOR can also refer to a READ-SERVICE and an OUT-PARAM-IF for reading configuration data from an ECU.

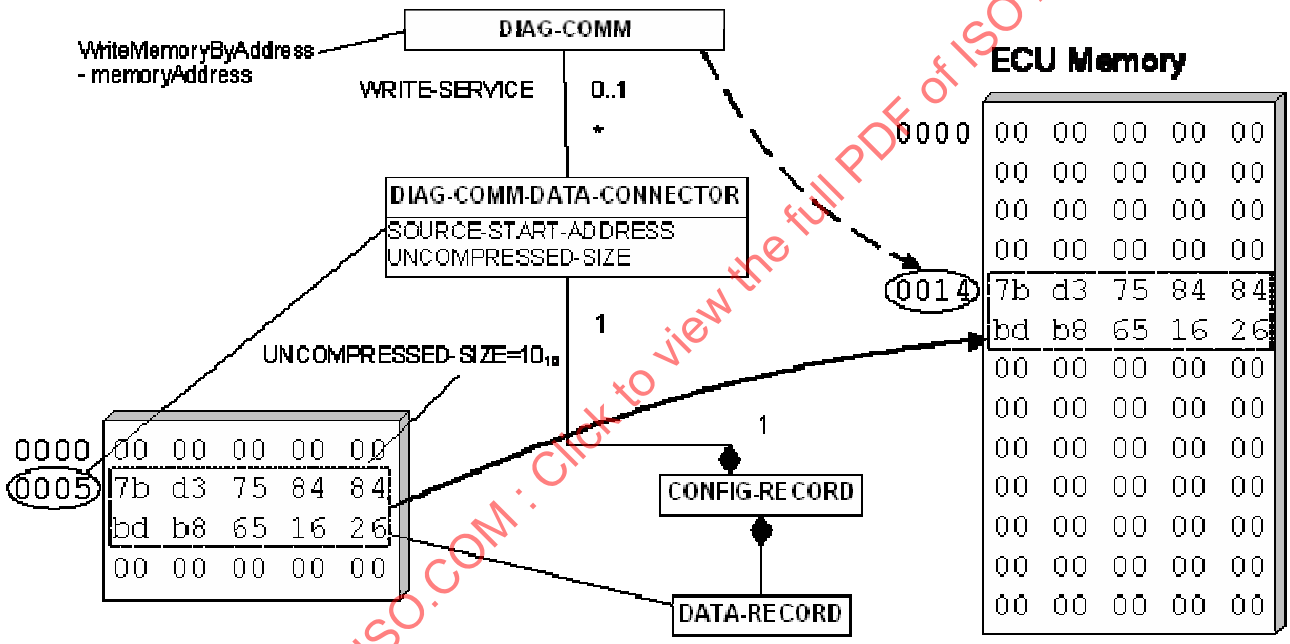


Figure 165 — Write data to the ECU by KWP2000-WriteMemoryByAddress[29)]

7.8.3 ECU-CONFIG

The following class diagrams show the structures that ODX provides for the purpose of ECU configuration. The ODX data element ECU-CONFIG of type ODX-CATEGORY aggregates all data elements for ECU configuration programming. Being of type ODX-CATEGORY means, that ECU-CONFIG is a separate XML-file of the ODX XML files family like COMPARAM-SPEC, DIAGLAYER-SPEC, FLASH, etc. with the postfix .odx-e or simply .odx. Inheritance from ODX-CATEGORY means also that authoring and versioning information can be applied (COMPANY-DATA, ADMIN-DATA).

29) The UML model elements are simplified in this figure. See 7.8.4 for the comprehensive model.

The contained CONFIG-DATA-DICTIONARY-SPEC shall contain the DATA-OBJECT-PROPs and UNIT-SPECs to be referenced from within the ECU-CONFIG. See Figure 166 — UML representation and structure of ECU configuration data for detailed modelling information.

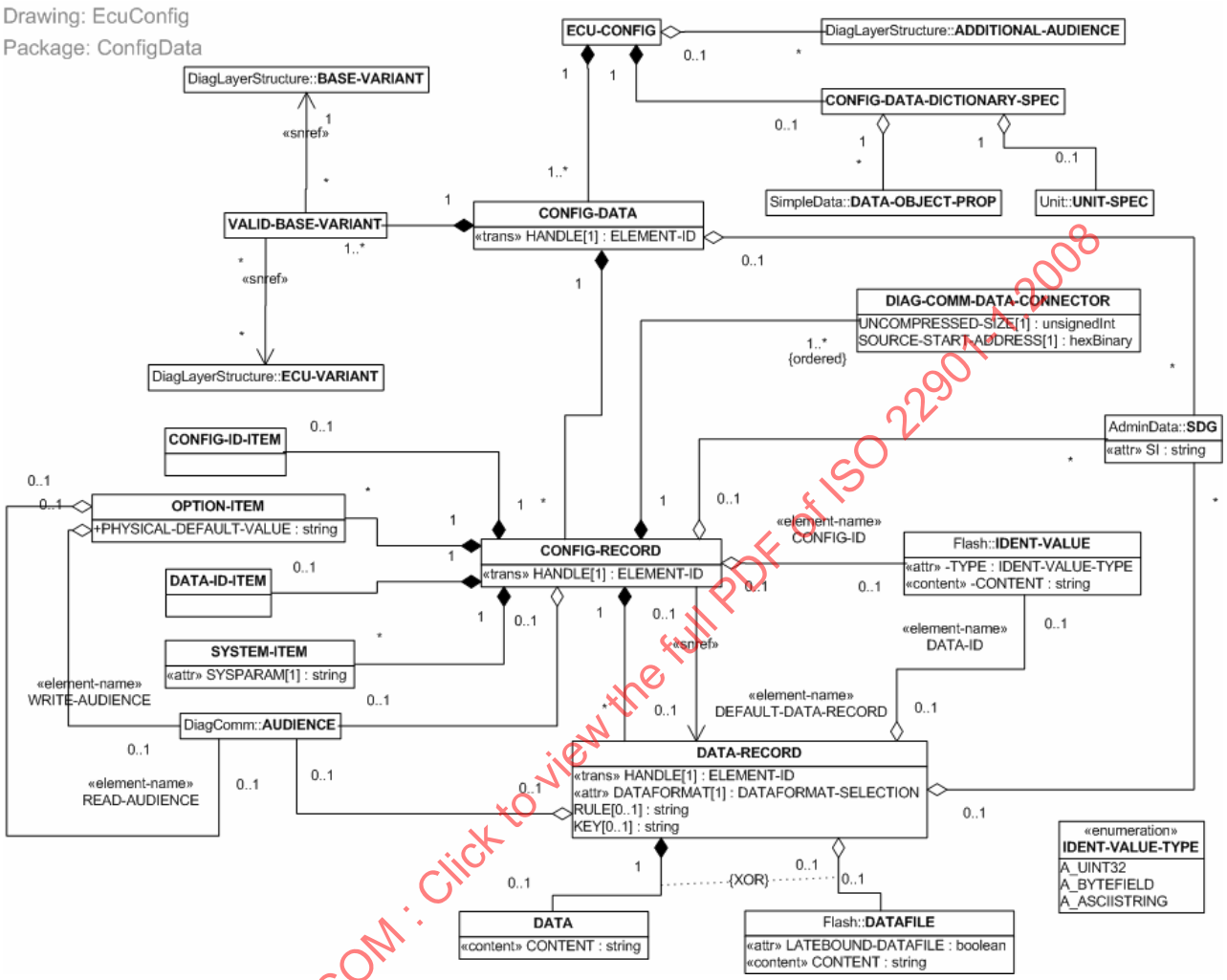


Figure 166 — UML representation and structure of ECU configuration data

The size of the notional memory buffer is defined by the DIAG-COMM-DATA-CONNECTOR elements described in 7.8.4.

7.8.4 Comprehensive UML model

The meaning of CONFIG-DATA is described at the beginning of this subclause. The VALID-BASE-VARIANT determines the Base-Variant and ECU-Variants the complete set of CONFIG-DATA is valid for. The CONFIG-DATA may be valid for any number of base variants. Related to a single base variant, the three cases described below are allowed.

a) Only one VALID-BASE-VARIANT refers to this base variant via BASE-VARIANT-SNREF without any ECU-VARIANT-SNREFs.
As a consequence the CONFIG-DATA is valid for the base variant and all variants inherited from this.

b) Only one VALID-BASE-VARIANT refers to this base variant via BASE-VARIANT-SNREF with at least one ECU-VARIANT-SNREF.
As consequence CONFIG-DATA is valid only for the variants explicitly specified, but not for the base variant itself.

c) Two VALID-BASE-VARIANTs refer to this base variant, one of them contains BASE-VARIANT-SNREF with at least one ECU-VARIANT-SNREF, the other one without any ECU-VARIANT-SNREFs. As a consequence the CONFIG-DATA is valid only for the variants explicitly specified and for the base variant self.

d) Two VALID-BASE-VARIANTs refer to this base variant, but none of them contains an ECU-VARIANT-SNREF. As a consequence the CONFIG-DATA is valid only for this BASE-VARIANT and for no ECU-VARIANTS at all.

No other combinations are allowed.

Multiple CONFIG-DATA can be valid for a certain BASE-VARIANT or ECU-VARIANT. They can be defined in the same ECU-CONFIG or in different ECU-CONFIG categories. Yet, the SHORT-NAME of each CONFIG-DATA shall be unique in the DIAG-LAYER it is valid for.

Each CONFIG-DATA contains several CONFIG-RECORDs. A CONFIG-RECORD may contain an IDENT-VALUE as CONFIG-ID. Its value shall be unique among all CONFIG-RECORDs within in a CONFIG-DATA. Each CONFIG-RECORD can contain several DATA-RECORDs which are predefined blocks of coding information and ITEM-VALUEs containing individual coding values. One of these may be marked as DEFAULT-DATA-RECORD via «snref». Its data will be used at initialization time of the configuration record. KEY and RULE contain additional information, which are presented at the 3D API. The diagnostic application can use this information for selecting or for filtering of data. For example a set of keys can be defined in RULE like “K1 K4 K6” (means logical or). If the key “K1” (Configuration 1) is given only DATA-RECORD or ITEM-VALUEs suitable for that K1 could be used. The format for KEY and RULE is not standardized.

The attribute DATAFORMAT of type DATAFORMAT-SELECTION can be either INTEL-HEX, MOTOROLA-S or BINARY (compare FLASHDATA/DATAFORMAT). DATAFILE is described at the FLASH section whereas DATA corresponds to INTERN-FLASHDATA. Segmented binary data referenced from a DATA-RECORD (intern or external) shall define values for at least all addresses from zero to size of configuration record minus 1. For each of these addresses, exactly one value shall be given, i.e. no gaps and no overlapping segments are allowed.

Each DATA-RECORD can have an IDENT-VALUE (element name DATA-ID) where the IDENT-VALUE-TYPE defines the datatype (see description of FLASH) and CONTENT the matching value. Its value shall be unique among the DATA-RECORDS within a CONFIG-RECORD. If such a DATA-RECORD identifier is part of the CONFIG-RECORD itself its position and coding is defined by the DATA-ID-ITEM component of CONFIG-RECORD. It is up to the data provider to ensure that the actual data of the DATA-RECORD encodes the same identifier value as given in the DATA-ID. The same holds for IDENT-VALUE/CONFIG-ID and CONFIG-ID-ITEM. See Figure 167 — UML representation and structure of ECU configuration items for detailed modelling information.

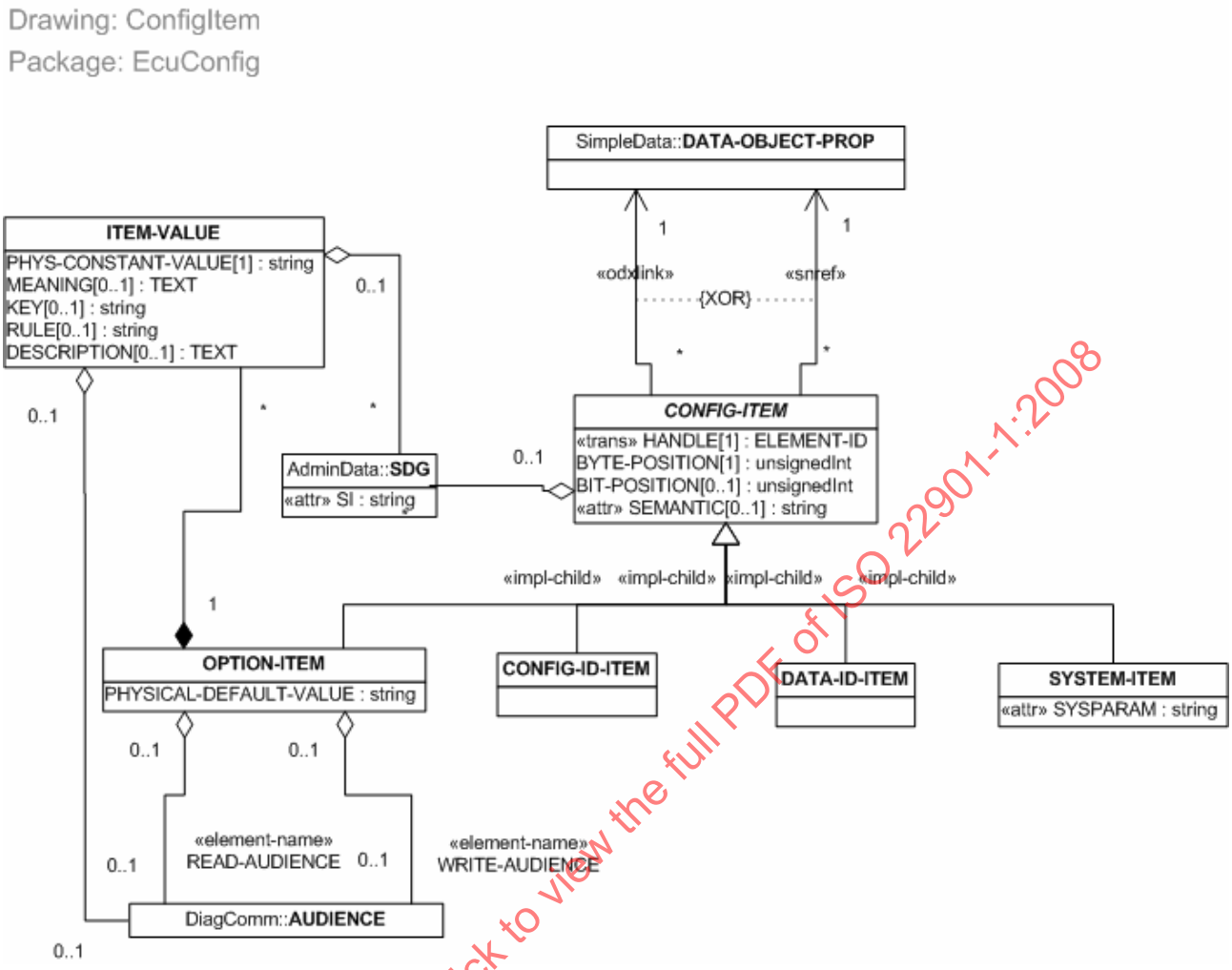


Figure 167 — UML representation and structure of ECU configuration items

Each CONFIG-ITEM refers to a DATA-OBJECT-PROP that defines transformation from physical to coded representation and vice versa. It is through these CONFIG-ITEMs that an diagnostic application can query the current configuration. CONFIG-ITEM/BYTE-POSTION and CONFIG-ITEM/BIT-POSITION refer to the position in the notional memory buffer of the CONFIG-RECORD the coded value of the CONFIG-ITEM is stored. Just like PARAMs refer to a position in the PDU.

Access to OPTION-ITEMS can be restricted via READ-AUDIENCE and WRITE-AUDIENCE. Only the groups set in the READ-AUDIENCE can read the value of the OPTION-ITEM. Only the groups set in the WRITE-AUDIENCE can change the value of the OPTION-ITEM. If no AUDIENCE is defined for reading or writing the corresponding kind of access is not restricted.

See Figure 168 — UML representation of ECU configuration: DIAG-COMM-DATA-CONNECTOR for detailed modelling information.

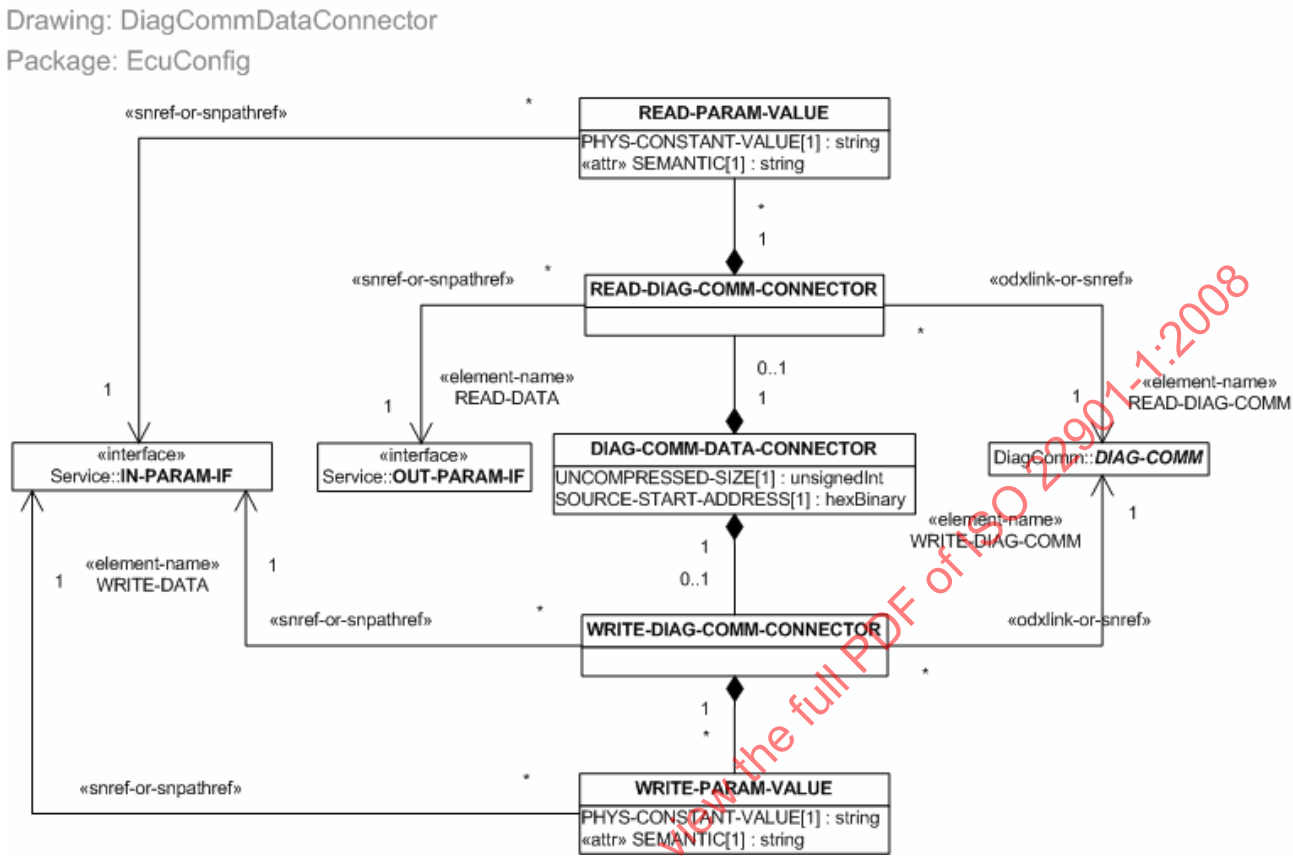


Figure 168 — UML representation of ECU configuration: DIAG-COMM-DATA-CONNECTOR

The CONFIG-RECORD references at least one DIAG-COMM-DATA-CONNECTOR. Each DIAG-COMM-DATA-CONNECTOR (see Figure 168 — UML representation of ECU configuration: DIAG-COMM-DATA-CONNECTOR) defines how to read or/and write a cut out of the data stored in the notional memory buffer of the CONFIG-RECORD. The cut out of the memory buffer starts at SOURCE-START-ADDRESS and extends over UNCOMPRESSED-SIZE bytes.

The size of the notional memory buffer is defined by the maximum of the sums of SOURCE-START-ADDRESS and UNCOMPRESSED-SIZE. Thus, the largest address used to read or write data defines the size of the buffer. Data that cannot be read from or stored in the ECU, because it can never be addressed by a DIAG-COMM-DATA-CONNECTOR, need not to be defined by the CONFIG-RECORD.

If the DIAG-COMM-DATA-CONNECTOR defines how to store data in the ECU it shall contain a WRITE-DIAG-COMM-CONNECTOR. The WRITE-DIAG-COMM-CONNECTOR defines three things, as described below.

a) A DIAG-SERVICE or SINGLE-ECU-JOB is defined that shall be used for writing. It is referenced via «odxlink-or-snref» as WRITE-DIAG-COMM. This DIAG-COMM shall be usable by all DIAG-LAYERs the enclosing CONFIG-DATA is valid for. If it is referenced via «snref» a DIAG-COMM with the given SHORT-NAME shall be visible in each layer. If it is referenced via «odxlink» for each layer the referenced DIAG-COMM shall either be defined or imported in at least on of the DIAG-LAYERs in the value inheritance scope of that layer. This is the same validity as checked by rule #58 for «odxlink»s to DIAG-COMMs defined in that DIAG-LAYER.
   As an example, assume a BASE-VARIANT BV and two ECU-VARIANTS E1 and E2 inheriting from that BASE-VARIANT. BV defines a service “WriteDataBV”, E1 defines a service “ReadDataE1”, and E2

maintains a NOT-INHERITED-DIAG-COMM with «snref» to “WriteDataBV” and defines no services of its own. A CONFIG-DATA is valid only for E1 and E2, but not BV:

1) it shall not refer via «snref» to “WriteDataBV”, because that SHORT-NAME cannot be resolved in E2;

2) it can refer via «odxlink» to the service “WriteDataBV” in BV, because BV is in the value inheritance scope of E1 and E2;

3) it shall not refer via «odxlink» to the service “ReadDataE1”, because no DIAG-LAYER in the inheritance scope of E2 defined or imported that service.

b) A PARAM of that DIAG-COMM that receives the cut out of the notional memory buffer to be written to the ECU. It is defined by «snref-or-snpathref» as WRITE-DATA. As the cut out is always a byte sequence taken from the notional memory buffer the PHYSICAL-TYPE's BASE-DATA-TYPE of that parameter's DATA-OBJECT-PROP shall be A_BYTEFIELD large enough to hold UNCOMPRESSED-SIZE many bytes.

c) Further default values for other PARAMs of that DIAG-COMM. For each such PARAM the WRITE-DIAG-COMM-CONNECTOR contains a WRITE-PARAM-VALUE. The reference from WRITE-PARAM-VALUE to an IN-PARAM-IF of the DIAG-COMM determines the PARAM that is to be set e.g. to the memory address or the DID information (e.g. local ID LID). The D-server will set that parameter to the PHYS-CONST-VALUE stored in the WRITE-PARAM-VALUE. Therefore, the PHYS-CONST-VALUE shall match the PHYSICAL-TYPE's BASE-DATA-TYPE of that parameter's DATA-OBJECT-PROP. The SEMANTIC attribute describes the meaning of the PARAM. Typical values are ADDRESS or DID.

If the DIAG-COMM-DATA-CONNECTOR defines how to read data from the ECU it shall contain a READ-DIAG-COMM-CONNECTOR. It defines nearly the same (and the same restriction apply regarding DIAG-COMMs that can be referenced), except that the READ-DATA element points to a response parameter (via OUT-PARAM-IF). The D-server will retrieve the value of that PARAM and store it in the cut out of the notional memory buffer described by SOURCE-START-ADDRESS and UNCOMPRESSED-SIZE at the DIAG-COMM-DATA-CONNECTOR. As the notional memory buffer contains a sequence of bytes, the PHYSICAL-TYPE's BASE-DATA-TYPE of that parameter's DATA-OBJECT-PROP shall be A_BYTEFIELD.

7.8.5 Example

The following sample of XML shows how a DATA-RECORD and a CONFIG-ITEM look like in ECU-CONFIG.

```
<ECU-CONFIG ID="EC0001">
  <SHORT-NAME>ECU_1</SHORT-NAME>
  <CONFIG-DATAS>
    <CONFIG-DATA>
      <SHORT-NAME>CONFIG_DATA_1</SHORT-NAME>
      <VALID-BASE-VARIANTS>
        <VALID-BASE-VARIANT>
          <ECU-VARIANT-SNREFS>
            <ECU-VARIANT-SNREF SHORT-NAME="E1"/>
          </ECU-VARIANT-SNREFS>
          <BASE-VARIANT-SNREF SHORT-NAME="BV"/>
        </VALID-BASE-VARIANT>
      </VALID-BASE-VARIANTS>
      <CONFIG-RECORDS>
        <CONFIG-RECORD>
          <SHORT-NAME>Coding</SHORT-NAME>
          <DIAG-COMM-DATA-CONNECTORS>
            <DIAG-COMM-DATA-CONNECTOR>
              <UNCOMPRESSED-SIZE>15</UNCOMPRESSED-SIZE>
              <SOURCE-START-ADDRESS>00</SOURCE-START-ADDRESS>
            </DIAG-COMM-DATA-CONNECTOR>
          </DIAG-COMM-DATA-CONNECTORS>
```

```
          <CONFIG-ID TYPE="A_UINT32">01</CONFIG-ID>
          <DATA-RECORDS>
            <DATA-RECORD DATAFORMAT="BINARY">
              <SHORT-NAME>Default_String</SHORT-NAME>
              <DATA-ID TYPE="A_ASCIISTRING">0001</DATA-ID>
              <DATA>0102030405060708090A0B0C0D0E0F</DATA>
            </DATA-RECORD>
          </DATA-RECORDS>
          <OPTION-ITEMS>
            <OPTION-ITEM>
              <SHORT-NAME>Config_ITEM_1</SHORT-NAME>
              <BYTE-POSITION>0</BYTE-POSITION>
              <BIT-POSITION>0</BIT-POSITION>
              <DATA-OBJECT-PROP-SNREF SHORT-NAME="DOP_UNIT32_8Bit"/>
              <ITEM-VALUES>
                <ITEM-VALUE>
                  <PHYS-CONSTANT-VALUE>0</PHYS-CONSTANT-VALUE>
                  <MEANING>Off</MEANING>
                  <KEY>K1</KEY>
                </ITEM-VALUE>
                <ITEM-VALUE>
                  <PHYS-CONSTANT-VALUE>1</PHYS-CONSTANT-VALUE>
                  <MEANING>on</MEANING>
                  <KEY>K2</KEY>
                </ITEM-VALUE>
              </ITEM-VALUES>
            </OPTION-ITEM>
          </OPTION-ITEMS>
        </CONFIG-RECORD>
      </CONFIG-RECORDS>
    </CONFIG-DATA>
  </CONFIG-DATAS>
  <CONFIG-DATA-DICTIONARY-SPEC>
    <DATA-OBJECT-PROPS>
      <DATA-OBJECT-PROP ID="DOP0001">
        <SHORT-NAME>DOP_UINT32_8Bit</SHORT-NAME>
        <COMPU-METHOD><CATEGORY>IDENTICAL</CATEGORY></COMPU-METHOD>
        <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" xsi:type="STANDARD-LENGTH-TYPE">
          <BIT-LENGTH>8</BIT-LENGTH>
        </DIAG-CODED-TYPE>
        <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
      </DATA-OBJECT-PROP>
    </DATA-OBJECT-PROPS>
  </CONFIG-DATA-DICTIONARY-SPEC>
</ECU-CONFIG>
```


7.9 Function dictionary

7.9.1 Terms and requirements

ODX provides in a wide range a communication-oriented view on an ECU's diagnostic functionality. However, this does not always meet today's way of designing vehicles, because many functions of a vehicle are distributed across several ECUs.

Function oriented diagnostics becomes more and more important since the functional point of view is much closer to the customer experienced symptoms a malfunction might cause.

This is covered with the structures described in this subclause (7.9) based on the aspects of communication-oriented diagnostics definitions and requirements. The following use cases were considered when designing this ODX-CATEGORY:

— definition of a hierarchical function catalogue to organize service tester user interfaces;

— references to one or several ECUs and their diagnostic data content relevant for that function;

— reference to services/jobs to make functions “executable”;

— definition of function input and output parameters with optional references to parameters of related services;

— description of ECU sub functionality- or components (e.g. LIN-slaves).

The last aspect is described inside the DIAG-LAYER-CONTAINER and is not part of the ODX-CATEGORY FUNCTION-DICTIONARY.

7.9.2 Functions and function groups in ODX

A function in ODX represents a vehicle subsystem or functionality (e.g. Indicator lights) considered from the point of view of diagnostics. A function may divide into one or more system components /subfunctions that each may consist of several parts again and so on.

A function in FUNCTION-DICTIONARY refers to the set of diagnostic information implemented in one or several ECUs that is related to this function including diagnostic services, DTCs, environment data and parameters. It is not intended (and therefore not possible) to (re-)define any information that is already available.

For example the function “Indicator Light Left” as a subfunction of “Indicator Lights” is implemented across different ECUs and related to the according fault memories, input/output controls, and measurement values and so on. Furthermore the function “Blinking left” may be valid for several model lines and model years. From a functional point of view it is interesting to know which ECUs and diagnostic elements of an ECU are part of a function.

Finally, a function may have input and output parameters to influence the behaviour of a function respectively indicating the correct function behaviour.

Functions are defined in hierarchical structures (by recursion) to allow expressing different functional granularities:

— Indicator Lights;
  — Indicator Light Left;
  — Indicator Light Right;
  — Warning Indicator Lights;
— ….

A FUNCTION-NODE-GROUP is a container for already defined FUNCTION-NODES regardless to their hierarchical context, e.g. the FUNCTION-GROUP “Lights” could contain the FUNCTION-NODEs “Indicator Lights” and “Head Lights”:

- Lights;
  - Head Lights;
    - …;
- Indicator Lights;
  - Indicator Light Left;
  - Indicator Light Right;
  - Warning Indicator Lights;
  - ….

A FUNCTION-NODE-GROUP may also contain other FUNCTION-NODE-GROUPs. For a FUNCTION-NODE-GROUP applies the same requirements as for a FUNCTION-NODE (relation to ECUs, services, DTCs, parameters…).

In the example above it is use case dependent either to choose a FUNCTION-NODE or a FUNCTION-NODE-GROUP.

7.9.3 Function dictionary data model description

Figure 169 — UML representation of FUNCTION-DICTIONARY illustrates the FUNCTION-DICTIONARY which inherits from CATEGORY and provides ADMIN-DATA and COMPANY-DATA by default.

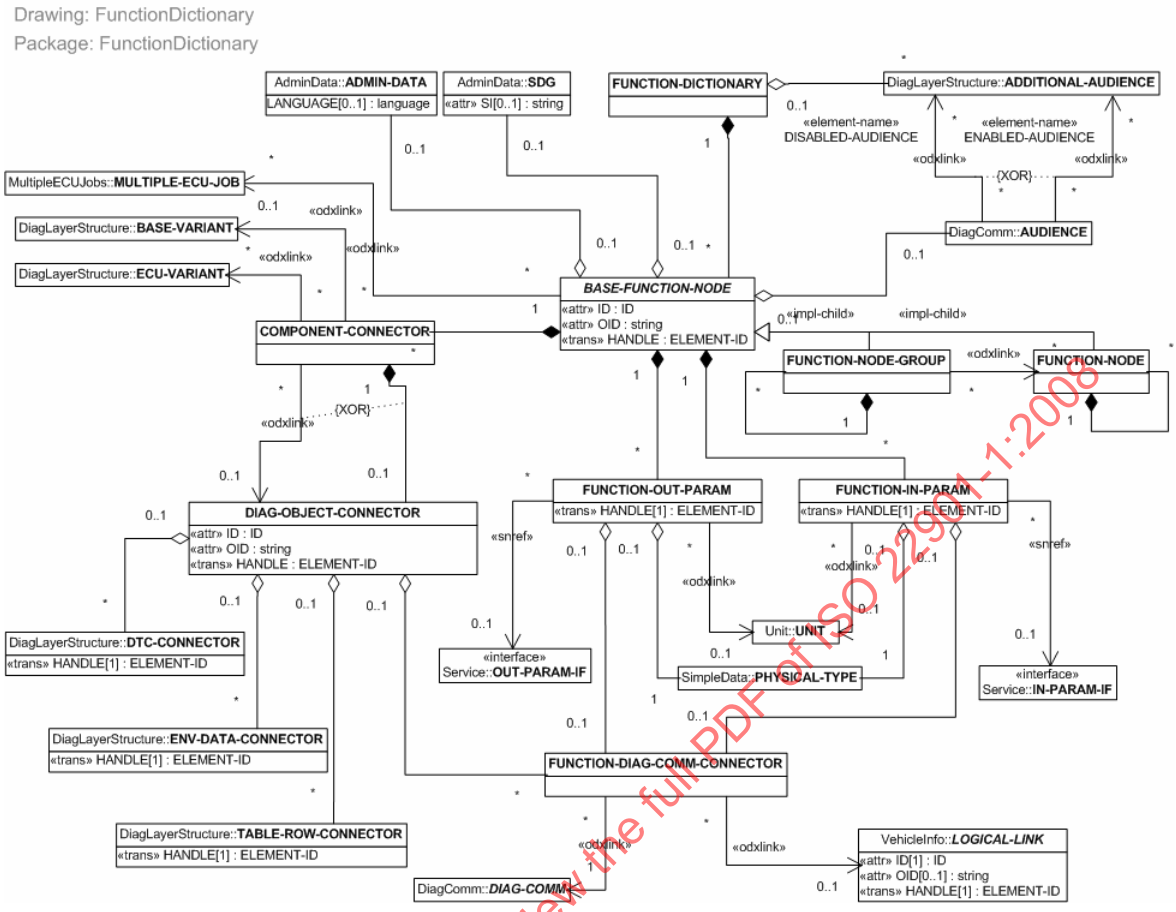


Figure 169 — UML representation of FUNCTION-DICTIONARY

A BASE-FUNCTION-NODE aggregates all common features of a FUNCTION-NODE/FUNCITON-NODE-GROUP (see 7.9.2) and therefore is implemented by the according subclasses. To be able to reflect the life cycle of BASE-FUNCTION-NODE instances the ADMIN-DATA (containing REVISION, TEAM-MEMBER,) has been added.

The hierarchy of FUNCTION-NODEs/FUNCTION-NODE-GROUPs can be considered as a function catalogue representing the functional layout of the vehicle. It might be useful to generate the function catalogue out of the same system where the vehicle's functional layout is described.

The element FUNCTION-NODE represents a function as part of a function hierarchy. The requirement of grouping functions is implemented by a FUNCTION-NODE-GROUP. FUNCTION-NODES are referenced via odxlink by a FUNCTION-NODE-GROUP.

A FUNCTION-NODE or FUNCTION-NODE-GROUP may only be relevant for or even restricted to certain departments or data customers. To reflect this, it is allowed to specify (ADDITIONAL-)AUDIENCEs, which may be used by a diagnostic application or as an data export/ conversion filter criteria by appropriate tools.

As mentioned above, a function often is distributed across ECUs and other components of the vehicle's network and therefore not only implemented in one single. The layout of the function's layout may vary from model line to model line. From that perspective it makes sense at first to describe which components (references to BASE- /ECU-VARIANTS) are contributing to a certain function and also to describe what is the right layout for a function.

A component's contribution to a function from diagnostic point of view may include DTCs, ENV-DATAs, DIAG-COMMs and TABLE-ROWS. The DIAG-OBJECT-CONNECTOR aggregates odxlink references to these

objects. The uniqueness is guaranteed by the DIAG-LAYERs that are referenced by the super ordinate COMPONENT-CONNECTOR.

For example, a referenced DTC-DOP not being available in a specific ECU-VARIANT layer would lead to unspecified behaviour of the D-server when at runtime this ECU-VARIANT is selected. At least one BASE-VARIANT or ECU-VARIANT shall be referenced by a COMPONENT-CONNECTOR. If a BASE-VARIANT and one or more ECU-VARIANTS are referenced, all ECU-VARIANTS have to inherit from the referenced BASE-VARIANT. If no BASE-VARIANT but more than one ECU-VARIANTs are referenced, all these ECU-VARIANTS have to inherit from the same BASE-VARIANT.

FUNCTION-IN-PARAMS and FUNCTION-OUT-PARAMS cover the following use cases:

— documentation of high level function input and output parameters without any technical relation, e.g. with no relevance for a D-server;

— documentation of Vehicle Message Matrix/VMM input and output signals related to a function;

— mapping of VMM-Signals to diagnostic content.

For the higher level description of a function, higher level input and output parameter descriptions can be useful, therefore the diagnostic description of the input and output PARAMs is optional. The higher level input and output parameter description could be useful, if a MULTIPLE-ECU-JOB is used for diagnosing the function.

For example, the value of one function's output signal is described as a PARAM of a measurement value, while the value of an function's input signal may be covered by a routine's service parameter. In the example the output signal and the routine's service parameter could both be accessed by diagnostic services, but for a higher level description only input "Current in V" and output "Temperature in C" is necessary.

Both FUNCTION-IN-PARAM and FUNCTION-OUT-PARAM have a PHYSICAL-TYPE and a UNIT to reflect this information without the necessity to resolve the (optionally) attached PARAMs.

Since the optional IN-/OUT-PARAM-IFs may only be referenced by SHORT-NAME, it is necessary to specify the service scope, for which the SHORT-NAME of the PARAMs shall be unique. This scope is given by the DIAG-COMM referenced by the FUCTION-IN-PARAM/FUNCTION-OUT-PARAM element. The reference to the DIAG-COMM is optional. In the case of having no DIAG-COMM referenced it is not allowed to have an IN-/OUT-PARAM-IF-SNREF specified and vice versa.

7.9.4 Function dictionary usage scenario

Figure 170 — Example four door ECUs in a vehicle network considers a vehicle with four doors, each having an ECU in it.

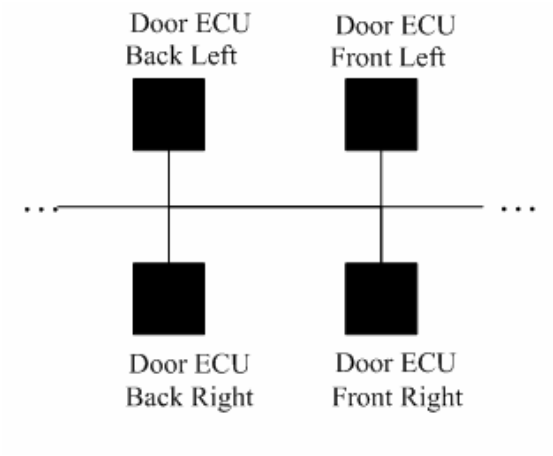


Figure 170 — Example four door ECUs in a vehicle network

The four ECUs each have their own functionality (e.g. window up and down, lock/ unlock door), but may also be addressed by a common functionality, e.g. central lock activating all locks or heat extraction opening all windows simultaneously. See Table 26 — Individual and common functionality of door ECUs.

Table 26 — Individual and common functionality of door ECUs

Functionality	Door ECU Back Left	Door ECU Back Right	Door ECU Front Left	Door ECU Front Right
Individual Functionality	Window Back Left up/ down	Window Back Right up/ down	Window Front Left up/ down	Window Front Right up/ down
	Door Lock Back Left open/close	Door Lock Back Right open/close	Door Lock Front Left open/close	Door Lock Front Right open/close
Common Functionality	Central Lock			
	Heat Extraction			

From the diagnostics point of view, these functions may be reflected in three ways:

a) the activation of the window or lock actuator is performed by an IO-Control service;

b) any problems with the actuators are expressed by error codes;

c) the current position of the window or the locker of a door is requested by a measurement service.

A FUNCTION-NODE container for the Door ECU Front Left reflecting this would look like this:

```
<ODX MODEL-VERSION="2.2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="odx.xsd">
    <FUNCTION-DICTIONARY ID="Body_Functions">
        <SHORT-NAME>Body_Functions</SHORT-NAME>
        <FUNCTION-NODES>
            <FUNCTION-NODE ID="FN_WinFrontLeftUpDown">
                <SHORT-NAME>FN_WinFrontLeftUpDown</SHORT-NAME>
```

```
        <COMPONENT-CONNECTORS>
            <COMPONENT-CONNECTOR>
                <BASE-VARIANT-REF ID-REF="DoorEcuFrontLeft"/>
                <DIAG-OBJECT-CONNECTOR ID="DOC_DoorEcuFrontLeft_Win">
                    <SHORT-NAME>DOC_DoorEcuFrontLeft_Win</SHORT-NAME>
                    <FUNCTION-DIAG-COMM-CONNECTORS>
                        <FUNCTION-DIAG-COMM-CONNECTOR>
                            <DIAG-COMM-REF ID-REF="IOCntrl_WinActuator"/>
                        </FUNCTION-DIAG-COMM-CONNECTOR>
                        <FUNCTION-DIAG-COMM-CONNECTOR>
                            <DIAG-COMM-REF ID-REF="Read_WinPos"/>
                        </FUNCTION-DIAG-COMM-CONNECTOR>
                        <FUNCTION-DIAG-COMM-CONNECTOR>
                            <DIAG-COMM-REF ID-REF="Read_WinKeyPanel"/>
                        </FUNCTION-DIAG-COMM-CONNECTOR>
                    </FUNCTION-DIAG-COMM-CONNECTORS>
                    <DTC-CONNECTORS>
                        <DTC-CONNECTOR>
                            <SHORT-NAME>A</SHORT-NAME>
                            <DTC-DOP-REF ID-REF="DTCDOP1"/>
                            <DTC-SNREF SHORT-NAME="DTC123456"/>
                        </DTC-CONNECTOR>
                        <DTC-CONNECTOR>
                            <SHORT-NAME>A</SHORT-NAME>
                            <DTC-DOP-REF ID-REF="DTCDOP1"/>
                            <DTC-SNREF SHORT-NAME="DTC234567"/>
                        </DTC-CONNECTOR>
                        <DTC-CONNECTOR>
                            <SHORT-NAME>A</SHORT-NAME>
                            <DTC-DOP-REF ID-REF="DTCDOP1"/>
                            <DTC-SNREF SHORT-NAME="DTC345678"/>
                        </DTC-CONNECTOR>
                    </DTC-CONNECTORS>
                </DIAG-OBJECT-CONNECTOR>
            </COMPONENT-CONNECTOR>
        </COMPONENT-CONNECTORS>
    </FUNCTION-NODE>
    <FUNCTION-NODE ID="FN_DoorLockFrontLeft">
        <SHORT-NAME>FN_DoorLockFrontLeft</SHORT-NAME>
        <COMPONENT-CONNECTORS>
            <COMPONENT-CONNECTOR>
                <BASE-VARIANT-REF ID-REF="DoorEcuFrontLeft"/>
                <DIAG-OBJECT-CONNECTOR ID="DOC_DoorEcuFrontLeft_Lock">
                    <SHORT-NAME>DOC_DoorEcuFrontLeft_Lock</SHORT-NAME>
                    <FUNCTION-DIAG-COMM-CONNECTORS>
                        <FUNCTION-DIAG-COMM-CONNECTOR>
                            <DIAG-COMM-REF ID-REF="Read_DoorLockFrontLeft_Status"/>
                        </FUNCTION-DIAG-COMM-CONNECTOR>
                        <FUNCTION-DIAG-COMM-CONNECTOR>
                            <DIAG-COMM-REF ID-REF="IOCntrl_DoorLockFrontLeft"/>
                        </FUNCTION-DIAG-COMM-CONNECTOR>
                    </FUNCTION-DIAG-COMM-CONNECTORS>
                    <DTC-CONNECTORS>
                        <DTC-CONNECTOR>
                            <SHORT-NAME>A2</SHORT-NAME>
                            <DTC-DOP-REF ID-REF="DTCDOP1"/>
                            <DTC-SNREF SHORT-NAME="DTC456789"/>
                        </DTC-CONNECTOR>
```

```
                    </DTC-CONNECTORS>
                </DIAG-OBJECT-CONNECTOR>
            </COMPONENT-CONNECTOR>
        </COMPONENT-CONNECTORS>
    </FUNCTION-NODE>
</FUNCTION-NODES>
<FUNCTION-NODE-GROUPS>
    <FUNCTION-NODE-GROUP ID="FNG_HeatExtraction">
        <SHORT-NAME>FNG_HeatExtraction</SHORT-NAME>
        <FUNCTION-NODE-REFS>
            <FUNCTION-NODE-REF ID-REF="FN_WinFrontLeftUpDown"/>
            <FUNCTION-NODE-REF ID-REF="FN_WinFrontRightUpDown"/>
            <FUNCTION-NODE-REF ID-REF="FN_WinBackLeftUpDown"/>
            <FUNCTION-NODE-REF ID-REF="FN_WinBackRightUpDown"/>
            <FUNCTION-NODE-REF ID-REF="...FN_Ventilation..."/>
        </FUNCTION-NODE-REFS>
    </FUNCTION-NODE-GROUP>
    <FUNCTION-NODE-GROUP ID="FNG_CentralLock">
        <SHORT-NAME>FNG_CentralLock</SHORT-NAME>
        <FUNCTION-NODE-REFS>
            <FUNCTION-NODE-REF ID-REF="FN_DoorLockFrontLeft"/>
            <FUNCTION-NODE-REF ID-REF="FN_DoorLockFrontRight"/>
            <FUNCTION-NODE-REF ID-REF="FN_DoorLockBackLeft"/>
            <FUNCTION-NODE-REF ID-REF="FN_DoorLockBackRight"/>
            <FUNCTION-NODE-REF ID-REF="...FN_StatusCentralLockRequest..."/>
        </FUNCTION-NODE-REFS>
    </FUNCTION-NODE-GROUP>
</FUNCTION-NODE-GROUPS>
</FUNCTION-DICTIONARY>
</ODX>
```

The additional comments below should be noted.

— In function hierarchies it is intentionally not predefined how FUNCTION-NODEs and their subordinates relate regarding their diagnostic functionality, i.e. the author (or the diagnostic application) decides whether the content related to a subordinate function automatically is relevant for its superior function and is automatically considered in its context.

— Function parameters are considered mainly to be used for documentation purposes, e.g. Vehicle Message Matrix (VMM) signals. As mentioned above, if there are any IN-/OUT-PARAM-IFs referenced, their SHORT-NAMEs are to be resolved in the scope of the related services referenced by the FUNCTION-DIAG-COMM-CONNECTOR aggregated to the FUNCTION-OUT-PARAM/FUNCTION-IN-PARAM.

— There is no relation defined between DTCs and ENV-DATAS referenced by a DIAG-OBJECT-CONNECTOR. Again, the author or the diagnostic application decides about whether or not making this relation and how to define such a relation.

IMPORTANT — Multiple FUNCTION-DICTIONARYs are allowed, but the SHORT-NAMEs of all BASE-FUNCTION-NODEs shall be globally unique.

7.9.5 SUB-COMPONENT data model description

Figure 171 — UML representation of SUB-COMPONENT illustrates the definition of a SUB-COMPONENT. A SUB-COMPONENT is defined at the DIAG-LAYER and is considered to be a functional unit in- or outside of an ECU that covers certain additional diagnostics relevant functionality either physically (e.g. a LIN slave) or logically.

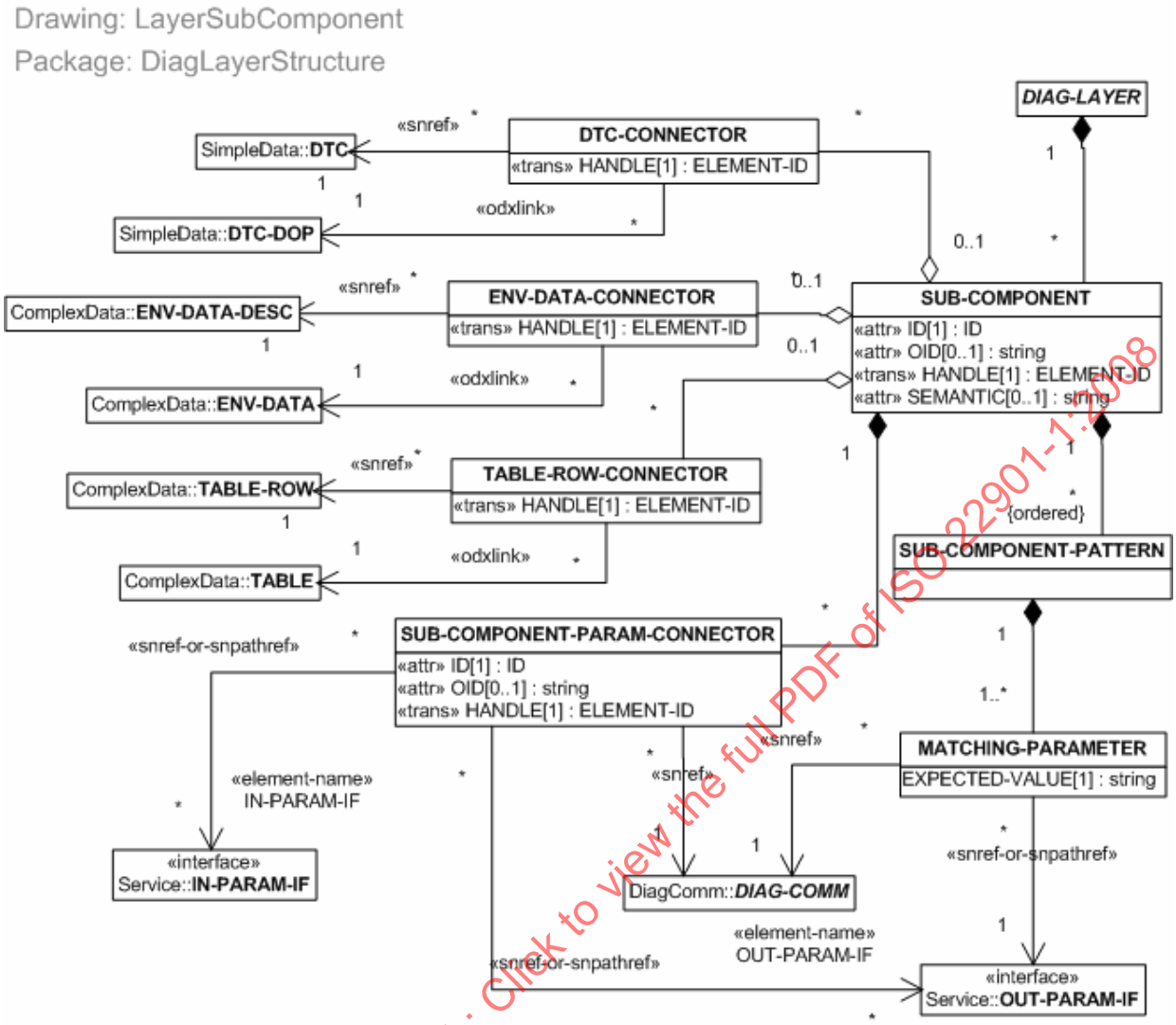


Figure 171 — UML representation of SUB-COMPONENT

To point out the use case, the SEMANTIC attribute may be used. Two SEMANTICS are predefined:

— SLAVE, if the SUB-COMPONENT describes a physical function unit;

— FUNCTION, if the SUB-COMPONENT describes a logical function unit.

The latter is interesting in context with FUNCTION-DICTIONARY, when a counterpart to a FUNCTION-NODE/-GROUP shall be defined (also see additional comments).

In opposite to a FUNCTION-NODE or a FUNCTION-NODE-GROUP a SUB-COMPONENT is always related to one explicit ECU (or even ECU-VARIANT) and can be considered as an additional layer below DIAG-LAYER. The difference is that no new data (DTC, ENV-DATA, and TABLE-ROW) is defined but only reused (referenced) from other layers. Therefore, the DTC-CONNECTOR, TABLE-ROW-CONNECTOR and ENV-DATA-CONNECTOR elements aggregated by a SUB-COMPONENT shall always only point to elements that are part of the DIAG-LAYER that the SUB-COMPONENT belongs to.

A SUB-COMPONENT-PARAM-CONNECTOR makes it possible to refer to a service and, optionally, to one or more IN- and OUT-PARAM-IFs. The related DIAG-COMM is the SHORT-NAME boundary for the PARAM-IFs.

The DIAG-COMM may be referenced without any IN-/OUT-PARAM-IF-REFS. In this case, the whole DIAG-COMM is relevant for the referencing SUB-COMPONENT.

SUB-COMPONENT-PATTERNS may be used to specify how the presence of a SUB-COMPONENT can be determined at runtime. In opposite to an ECU-VARIANTs VARIANT-PATTERNS this is not intended to be performed automatically but only for documentation purposes. However, after a SUB-COMPONENT has been selected or “identified” by a diagnostic application, the diagnostic application may provide according functionality. For example it can filter out any content that is not relevant for this SUB-COMPONENT.

There is no inheritance defined for SUB-COMPONENTs. SUB-COMPONENTs should be defined for each BASE-/ECU-VARIANT if necessary.

7.9.6 SUB-COMPONENT usage scenario

Figure 172 — Multi purpose ECU with 2 LIN slaves illustrates a multi purpose ECU with two Seat LIN slave controllers attached.

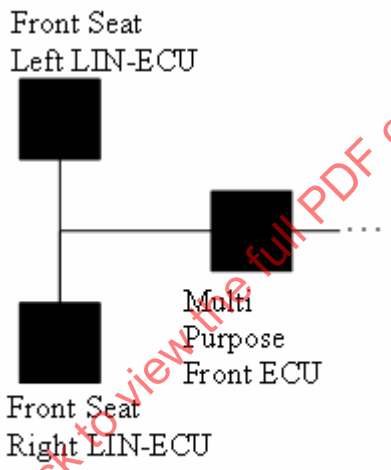


Figure 172 — Multi purpose ECU with 2 LIN slaves

Since only the multi purpose ECU (master) is attached to the CAN network, all diagnostic messages, that have an influence on the behaviour or request the status of the seat ECU, will be sent to the master. . The master then decides how to handle those diagnostic messages. On the other hand, if an error occurs inside one of the LIN slaves, this is probably indicated by a DTC activated in the master's fault memory since the slaves may not have their own fault memory.

This means that the diagnostic data description of the master ECU shall also consider the diagnostic functionality of the slaves.

This relationship can be expressed by defining a SUB-COMPONENT for each LIN-slave that:

a) reuses the content of the master ECU and therefore avoid redundancy;

b) resides inside the diagnostic description of the master ECU, to keep the SUB-COMPONENT that helps to keep it self contained;

c) may be checked for validity or presence by using its SUB-COMPONENT-PATTERNS to filter out any information that is not relevant in the current SUB-COMPONENT's context.

The following code matches the example above:

```
<SUB-COMPONENTS>
     <SUB-COMPONENT ID="_1234">
          <SHORT-NAME>Group1</SHORT-NAME>
          <DESC>
               <p>SUB-COMPONENT Example</p>
          </DESC>
          <!-- DTCs related to this SUB-COMPONENT-->
          <SUB-COMPONENT-PATTERNS>
               <SUB-COMPONENT-PATTERN>
                    <MATCHING-PARAMETERS>
                         <MATCHING-PARAMETER>
                              <EXPECTED-VALUE>A</EXPECTED-VALUE>
                              <DIAG-COMM-SNREF SHORT-NAME="ECU_Type_Read"/>
                              <OUT-PARAM-IF-SNREF SHORT-NAME="DataRecord"/>
                         </MATCHING-PARAMETER>
                    </MATCHING-PARAMETERS>
               </SUB-COMPONENT-PATTERN>
          </SUB-COMPONENT-PATTERNS>
          <SUB-COMPONENT-PARAM-CONNECTORS>
               <SUB-COMPONENT-PARAM-CONNECTOR ID="_1235">
                    <SHORT-NAME>SUB_COMPONENT_PARAM_CONNECTOR_EnableRxAndEnableTx_Control</SHORT-NAME>
                    <DIAG-COMM-SNREF SHORT-NAME="EnableRxAndEnableTx_Control"/>
               </SUB-COMPONENT-PARAM-CONNECTOR>
          </SUB-COMPONENT-PARAM-CONNECTORS>
          <DTC-CONNECTORS>
               <DTC-CONNECTOR>
                    <SHORT-NAME>DTCCONNECTORA</SHORT-NAME>
                    <DTC-DOP-REF ID-REF="DTCDOPA"/>
                    <DTC-SNREF SHORT-NAME="DTC01"/>
               </DTC-CONNECTOR>
               <DTC-CONNECTOR>
                    <SHORT-NAME>DTCCONNECTORB</SHORT-NAME>
                    <DTC-DOP-REF ID-REF="DTCDOPA"/>
                    <DTC-SNREF SHORT-NAME="DTC02"/>
               </DTC-CONNECTOR>
          </DTC-CONNECTORS>
          <!-- Description of Services related to this SUB-COMPONENT,
               i.e. In case the SUB-COMPONENT is Identified the 3D-System shall only provide the
                Services listed below.-->
     </SUB-COMPONENT>
     <SUB-COMPONENT ID="_2234">
          <!-- Within this SUB-COMPONENT, DTC01 and Service DevelopmentData_Read is available-->
          <SHORT-NAME>Group2</SHORT-NAME>
          <DESC>
               <p>SUB-COMPONENT Example</p>
          </DESC>
          <!-- DTCs related to this SUB-COMPONENT-->
          <SUB-COMPONENT-PATTERNS>
               <SUB-COMPONENT-PATTERN>
                    <MATCHING-PARAMETERS>
                         <MATCHING-PARAMETER>
                              <EXPECTED-VALUE>B</EXPECTED-VALUE>
                              <DIAG-COMM-SNREF SHORT-NAME="ECU_Type_Read"/>
                              <OUT-PARAM-IF-SNREF SHORT-NAME="DataRecord"/>
                         </MATCHING-PARAMETER>
                    </MATCHING-PARAMETERS>
               </SUB-COMPONENT-PATTERN>
```

```
        </SUB-COMPONENT-PATTERNS>
        <SUB-COMPONENT-PARAM-CONNECTORS>
            <SUB-COMPONENT-PARAM-CONNECTOR ID="_2235">
                <SHORT-NAME>SUB_COMPONENT_PARAM_CONNECTOR_DevelopmentData_Read</SHORT-NAME>
                <DIAG-COMM-SNREF SHORT-NAME="DevelopmentData_Read"/>
            </SUB-COMPONENT-PARAM-CONNECTOR>
        </SUB-COMPONENT-PARAM-CONNECTORS>
        <DTC-CONNECTORS>
            <DTC-CONNECTOR>
                <SHORT-NAME>A1</SHORT-NAME>
                <DTC-DOP-REF ID-REF="DTCDOPA"/>
                <DTC-SNREF SHORT-NAME="DTC01"/>
            </DTC-CONNECTOR>
        </DTC-CONNECTORS>
        <!-- Description of Services related to this SUB-COMPONENT,
            i.e. In case the SUB-COMPONENT is Identified the 3D-System
            shall only provide the Services listed below.-->
    </SUB-COMPONENT>
</SUB-COMPONENTS>
```

The additional comments below should be noted.

— The SUB-COMPONENT addresses primarily documentation use cases and offers another approach to the diagnostic content of the master. In other scenarios it might be more useful to have a separate DIAG-LAYER also for the LIN slaves.

— There may be the use case to have a relationship between a function defined in FUNCTION-DICTIONARY and a SUB-COMPONENT that e.g. covers this functionality in the ECU. This is not explicitly modelled but still can be covered by a diagnostic application mapping to SEMANTICS or naming conventions.

8 Data model implementation in XML

8.1 Classifier

8.1.1 Classes

8.1.1.1 Default mapping

8.1.1.1.1 Description

The default mapping for classes will result in a XML schema complex type definition for each class, using the class name as complex type name. The elements based on such a type will also use the class name as element name.

8.1.1.1.2 UML example

Figure 173 — Default class mapping illustrates the DIAG-LAYER-CONTAINER in UML.

Figure 173 — Default class mapping

8.1.1.1.3 Schema example

```
<xsd:complexType name="DIAG-LAYER-CONTAINER">
     ...
</xsd:complexType>
...
<xsd:complexType name="ODX">
    <!--Class: ODX-->
    <xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="1" type="DIAG-LAYER-CONTAINER"
        ...
    </xsd:choice>
    ...
</xsd:complexType>
```

8.1.1.1.4 ODX instance example

```
<?xml version="1.0" encoding="UTF-8"?>
<ODX MODEL-VERSION="2.2.0">
    <DIAG-LAYER-CONTAINER ID="ID0006">
        <SHORT-NAME>A</SHORT-NAME>
    </DIAG-LAYER-CONTAINER>
</ODX>
```

8.1.1.2 Transparent mapping

8.1.1.2.1 Description

If a class has the stereotype «trans», it will be mapped to a named group definition. This makes such a class transparent (invisible) with respect to an ODX instance document.

8.1.1.2.2 UML example

Figure 174 — Transparent class mapping illustrates the Transparent mapping in UML.

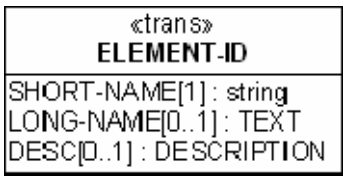


Figure 174 — Transparent class mapping

8.1.1.2.3 Schema example

```
<xsd:group name="ELEMENT-ID">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="SHORT-NAME"/>
    <xsd:element maxOccurs="1" minOccurs="0" type="TEXT" name="LONG-NAME"/>
    <xsd:element maxOccurs="1" minOccurs="0" type="DESCRIPTION" name="DESC"/>
  </xsd:sequence>
</xsd:group>
```

8.1.1.2.4 ODX instance example

```
<some-surrounding-element>
  ...
  <SHORT-NAME>SOME_SHORT_NAME</SHORT-NAME>
  <LONG-NAME>SOME_LONG_NAME</LONG-NAME>
  ...
</some-surrounding-element>
```

8.1.1.3 Interfaces

8.1.1.3.1 Description

Interfaces are identified by the «interface» stereotype. Interfaces can only be used as targets for SHORT-NAME references or odxlinks. The interface name determines the name of the linking element.

8.1.1.3.2 UML example

Figure 175 — Mapping interfaces illustrates the Mapping interface in UML.

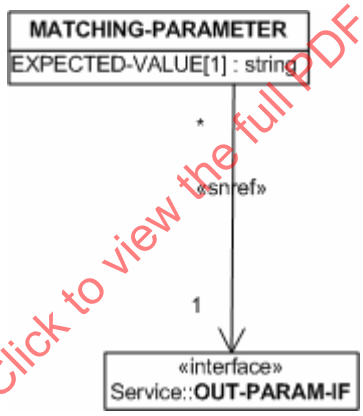


Figure 175 — Mapping interfaces

8.1.1.3.3 Schema example

```
<xsd:element maxOccurs="1" minOccurs="1" name="OUT-PARAM-IF-SNREF" type="SNREF"/>
```

8.1.1.3.4 ODX instance example

```
<OUT-PARAM-IF-SNREF SHORT-NAME="..."/>
```

8.1.1.4 Enumerations

8.1.1.4.1 Description

Enumerations are mapped to simple type definitions that are restrictions of the XML schema built-in simple type *xsd:string*.

8.1.1.4.2 UML example

Figure 176 — Enumerations illustrates the Enumerations in UML.

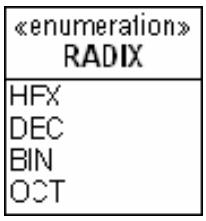


Figure 176 — Enumerations

8.1.1.4.3 Schema example

```
<xsd:simpleType name="RADIX">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="HEX"/>
        <xsd:enumeration value="DEC"/>
        <xsd:enumeration value="BIN"/>
        <xsd:enumeration value="OCT"/>
    </xsd:restriction>
</xsd:simpleType>
...
<xsd:complexType name="PHYSICAL-TYPE">
  ...
  <xsd:attribute use="optional" type="RADIX" name="DISPLAY-RADIX"/>
</xsd:complexType>
```

8.1.1.4.4 ODX instance example

```
<PHYSICAL-TYPE DISPLAY-RADIX="DEC"/>
```

8.1.1.5 Odxlink classes

8.1.1.5.1 Description

Classes with the stereotype «odxlink» can only occur in conjunction with an association having the same stereotype. This is semantically equivalent to association classes with «odxlink» stereotype and has been used to overcome a Visio deficiency that does not allow association classes to be shown on more than one page, as well as the improper handling of generalization. All association classes within the ODX model have to have the «odxlink» stereotype.

8.1.1.5.2 UML example

Figure 177 — Association class mapping illustrates the Association class mapping in UML.

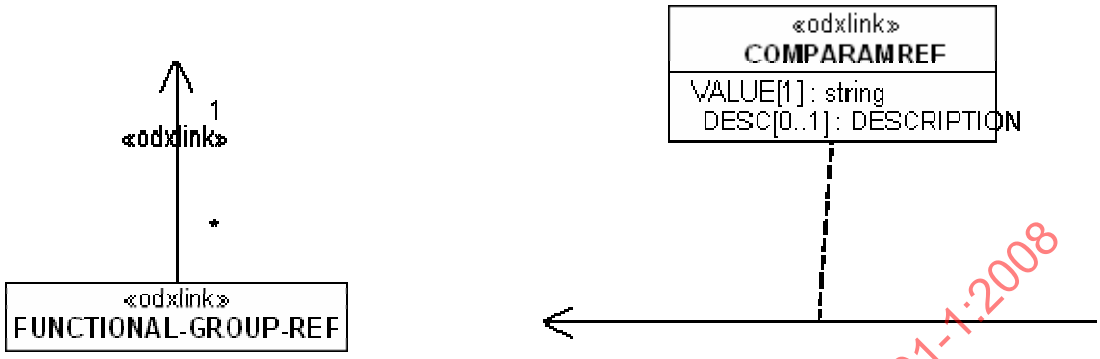


Figure 177 — Association class mapping

8.1.1.5.3 Schema example

```
<xsd:complexType name="COMPARAM-REF">
  <!--Association Class: COMPARAM-REF-->
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="VALUE"/>
    <xsd:element maxOccurs="1" minOccurs="0" type="TEXT" name="DESC"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="ODXLINK-ATTR"/>
</xsd:complexType>
...
<xsd:complexType name="FUNCTIONAL-GROUP-REF">
  <xsd:complexContent>
    <xsd:extension base="PARENT-REF"/>
  </xsd:complexContent>
</xsd:complexType>
...
<xsd:complexType name="PARENT-REF">
  <xsd:sequence>
    ...
  </xsd:sequence>
  <xsd:attributeGroup ref="ODXLINK-ATTR"/>
</xsd:complexType>
```

8.1.1.5.4 ODX instance example

```
<COMPARAM-REF DOCTYPE="CONTAINER" ID-REF="XID0001">
    <SIMPLE-VALUE>0.125</SIMPLE-VALUE>
</COMPARAM-REF>

...

<FUNCTIONAL-GROUP-REF DOCTYPE="CONTAINER" ID-REF="XID0003">
   <NOT-INHERITED-DIAG-COMMS>
   <NOT-INHERITED-DIAG-COMM>
               <DIAG-COMM-SNREF SHORT-NAME="A"></DIAG-COMM-SNREF>
           </NOT-INHERITED-DIAG-COMM>
```

```
    </NOT-INHERITED-DIAG-COMMS>
</FUNCTIONAL-GROUP-REF>
```

8.1.1.6 Imported classes

8.1.1.6.1 Description

The stereotype «import» denotes classes that are imported at the XML schema level and are therefore not described any further within the UML model. The class name will be usable as a type name within a XML schema element definition.

8.1.1.6.2 UML example

Figure 178 — Mapping of imported classes illustrates the Mapping of imported classes in UML.

Figure 178 — Mapping of imported classes

8.1.1.6.3 Schema example

```
<xsd:element maxOccurs="unbounded" minOccurs="0" type="p" name="p"/>
```

8.1.1.6.4 ODX instance example

```
<some-surrounding-element>
  ...
  <p>Some text or other XHTML elements</p>
  ...
</some-surrounding-element>
```

8.1.2 Attributes

8.1.2.1 Default mapping

8.1.2.1.1 Description

By default, UML attributes will be mapped to sub-elements of the corresponding class using the specified data type. The specified multiplicity will be transformed into the XML schema occurrence constraints.

8.1.2.1.2 UML example

Figure 179 — Default attribute mapping illustrates the Default attribute mapping in UML.

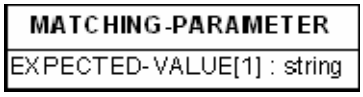


Figure 179 — Default attribute mapping

8.1.2.1.3 Schema example

```
<xsd:complexType name="MATCHING-PARAMETER">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="EXPECTED-VALUE"/>
    ...
  </xsd:sequence>
</xsd:complexType>
```

8.1.2.1.4 ODX instance example

```
<MATCHING-PARAMETER>
    <EXPECTED-VALUE>SOME_VALUE</EXPECTED-VALUE>
    <DIAG-COMM-SNREF SHORT-NAME="A"/>
    <OUT-PARAM-IF-SNREF SHORT-NAME="B"/>
</MATCHING-PARAMETER>
```

8.1.2.2 XML attribute mapping

8.1.2.2.1 Description

If an attribute has a stereotype <<attr>> it will be mapped to a XML attribute. The multiplicity is restricted to [0..1] or [1].

If the multiplicity is [0..1], a default value may be specified which will be mapped to a default constraint in XML. If, in addition to a default value, the attributes *changeability* is *frozen* it will be mapped to a fixed attribute.

8.1.2.2.2 UML example

Figure 180 — XML attribute mapping illustrates the XML attribute mapping in UML.

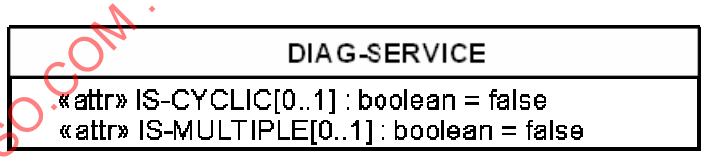


Figure 180 — XML attribute mapping

8.1.2.2.3 Schema example

```
<xsd:complexType name="DIAG-SERVICE">
  ...
  <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-CYCLIC"/>
  <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-MULTIPLE"/>
  ...
</xsd:complexType>
```

8.1.2.2.4 ODX instance example

```
<DIAG-SERVICE IS-CYCLIC="true" ID="A">
<SHORT-NAME>A</SHORT-NAME>
</DIAG-SERVICE>
```

8.1.2.3 Transparent mapping

8.1.2.3.1 Description

If an attribute has the stereotype «trans», it will be mapped to a named group definition. This makes such an attribute transparent (invisible) with respect to an ODX instance document. It is important that the referenced class also has the «trans» stereotype.

8.1.2.3.2 UML example

Figure 181 — Transparent attribute mapping illustrates the Transparent attribute mapping in UML.

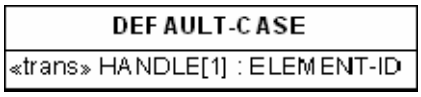


Figure 181 — Transparent attribute mapping

8.1.2.3.3 Schema example

```
<xsd:complexType name="DEFAULT-CASE">
  <xsd:sequence>
    <xsd:group ref="ELEMENT-ID"/>
    ...
  </xsd:sequence>
</xsd:complexType>
...
<xsd:group name="ELEMENT-ID">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="SHORT-NAME"/>
    <xsd:element maxOccurs="1" minOccurs="0" type="TEXT" name="LONG-NAME"/>
    <xsd:element maxOccurs="1" minOccurs="0" type="DESCRIPTION" name="DESC"/>
  </xsd:sequence>
</xsd:group>
```

8.1.2.3.4 ODX instance example

```
<DEFAULT-CASE>
  <SHORT-NAME>SOME_SHORT_NAME</SHORT-NAME>
  <LONG-NAME>SOME_LONG_NAME</LONG-NAME>
</DEFAULT-CASE>
```

8.1.2.4 XML content mapping

8.1.2.4.1 Description

If an UML attribute has the stereotype «content», it will be mapped to the content of the class' corresponding XML element. Not more than one attribute per class may have this stereotype. The multiplicity is restricted to [0..1] or [1].

8.1.2.4.2 UML example

Figure 182 — XML content mapping illustrates the XML content mapping in UML.

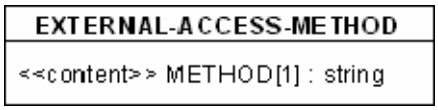


Figure 182 — XML content mapping

8.1.2.4.3 Schema example

```
<xsd:complexType name="EXTERNAL-ACCESS-METHOD">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string"/>
  </xsd:simpleContent>
</xsd:complexType>
```

8.1.2.4.4 ODX instance example

```
<EXTERNAL-ACCESS-METHOD ID="A">
    <SHORT-NAME>A</SHORT-NAME>
    <METHOD/>
</EXTERNAL-ACCESS-METHOD>
```

8.2 Relationships

8.2.1 Generalizations

8.2.1.1 Overview

The ODX model uses two different implementation schemes of generalization denoted by the «impl-parent» and «impl-child» stereotypes. The implementation scheme shall not change in a single inheritance tree. The XML schema types generated for both methods are identical however they are referenced differently.

8.2.1.2 Parent generalization

8.2.1.2.1 Description

Using the *parent generalization* concept, there are only XML elements created using the most general class of an inheritance tree. The actual derived type is being identified by the `xsi:type` attribute that is part of the XML schema instance namespace.

8.2.1.2.2 UML example

Figure 183 — UML representation of parent generalization illustrates the Parent generalization in UML.

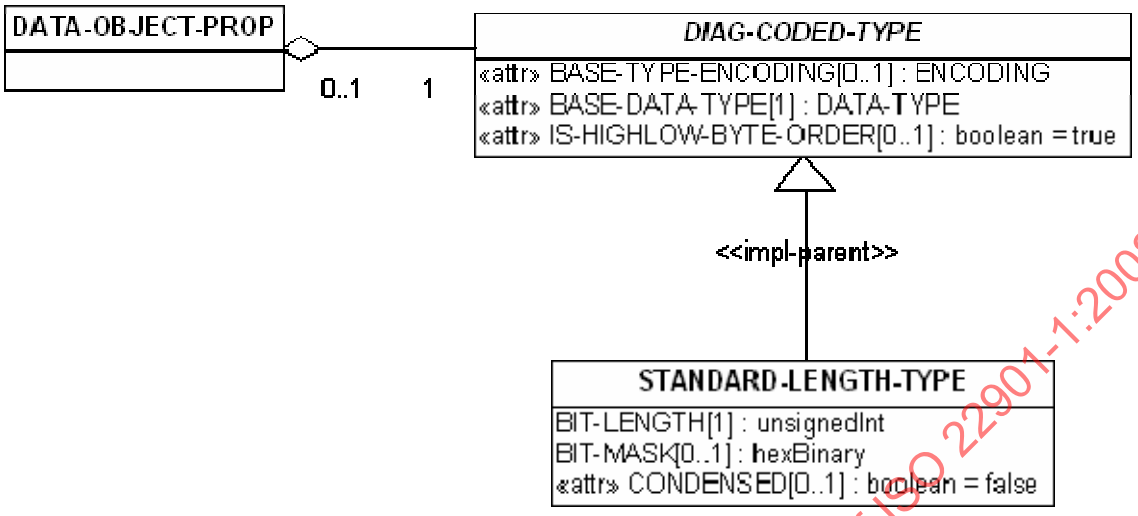


Figure 183 — UML representation of parent generalization

8.2.1.2.3 Schema example

```
<xsd:complexType name="DIAG-CODED-TYPE">
  <!--Class: DIAG-CODED-TYPE-->
  <xsd:attribute use="optional" type="ENCODING" name="BASE-TYPE-ENCODING"/>
  <xsd:attribute use="required" type="DATA-TYPE" name="BASE-DATA-TYPE"/>
  <xsd:attribute use="optional" type="TERMINATION" name="TERMINATION"/>
  <xsd:attribute default="true" use="optional" type="xsd:boolean"
                 name="IS-HIGHLOW-BYTE-ORDER"/>
</xsd:complexType>
...
<xsd:complexType name="STANDARD-LENGTH-TYPE">
  <xsd:complexContent>
  <xsd:extension base="DIAG-CODED-TYPE">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="BIT-LENGTH"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:hexBinary" name="BIT-MASK"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
...
<xsd:complexType name="DATA-OBJECT-PROP">
  <xsd:complexContent>
    <xsd:extension base="DOP-BASE">
      <xsd:sequence>
        ...
        <xsd:element maxOccurs="1" minOccurs="1" type="DIAG-CODED-TYPE"
                     name="DIAG-CODED-TYPE"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

8.2.1.2.4 ODX instance example

```
<DATA-OBJECT-PROP ID="A">
    <SHORT-NAME>DTC_HEX_VALUE</SHORT-NAME>
    <COMPU-METHOD>
        <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32" BASE-TYPE-ENCODING="2C" xsi:type="STANDARD-LENGTH-TYPE">
        <BIT-LENGTH>8</BIT-LENGTH>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE="A_UINT32"/>
</DATA-OBJECT-PROP>
```

8.2.1.3 Child generalization

8.2.1.3.1 Description

Using the *child generalization* concept, there are only XML elements created for the most specialized classes of an inheritance tree.

8.2.1.3.2 UML example

Figure 184 — UML example of child generalization illustrates the child generalization in UML.

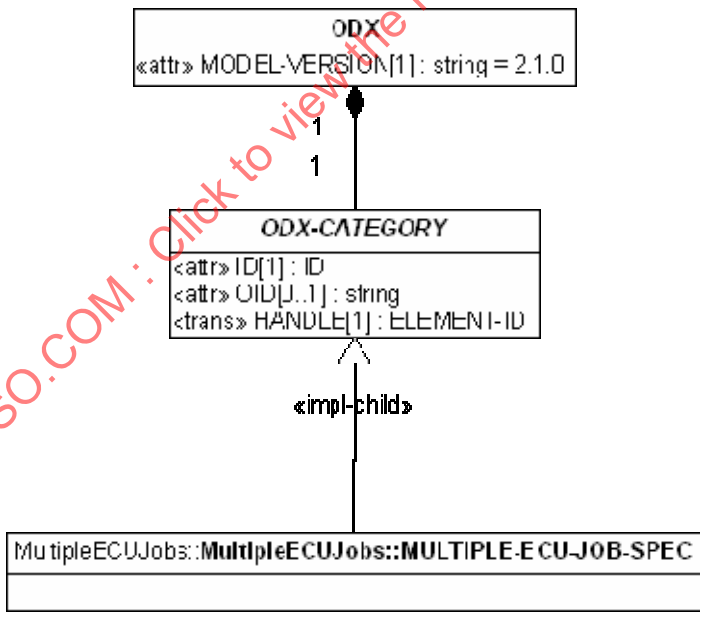


Figure 184 — UML example of child generalization

8.2.1.3.3 Schema example

```
<xsd:complexType name="ODX-CATEGORY">
  ...
</xsd:complexType>
...
<xsd:complexType name="MULTIPLE-ECU-JOB-SPEC">
  <xsd:complexContent>
    <xsd:extension base="ODX-CATEGORY">
      ...
```

```
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
...
<xsd:complexType name="ODX">
  <xsd:choice>
    ...
    <xsd:element maxOccurs="1" minOccurs="1" type="MULTIPLE-ECU-JOB-SPEC"
                 name="MULTIPLE-ECU-JOB-SPEC"/>
    ...
  </xsd:choice>
  ...
</xsd:complexType>
```

8.2.1.3.4 ODX instance example

```
<ODX>
  <MULTIPLE-ECU-JOB-SPEC>
  ...
  </MULTIPLE-ECU-JOB-SPEC>
</ODX>
```

8.2.2 Associations

8.2.2.1 Composition and aggregation

8.2.2.1.1 Description

Composition and aggregation are both mapped to identical XML structures. This is done in a way that the aggregated object becomes a sub-element of the aggregating object. The multiplicities given at the association end of the aggregated class are mapped to XML schema occurrence constraints. The multiplicities given at the other end are irrelevant for the implementation.

If the upper bound of the specified multiplicity exceeds 1, the association will be mapped to a wrapper element as described in 8.2.2.2.

8.2.2.1.2 UML example

Figure 185 — UML example of composition and aggregation illustrates the composition and aggregation in UML.

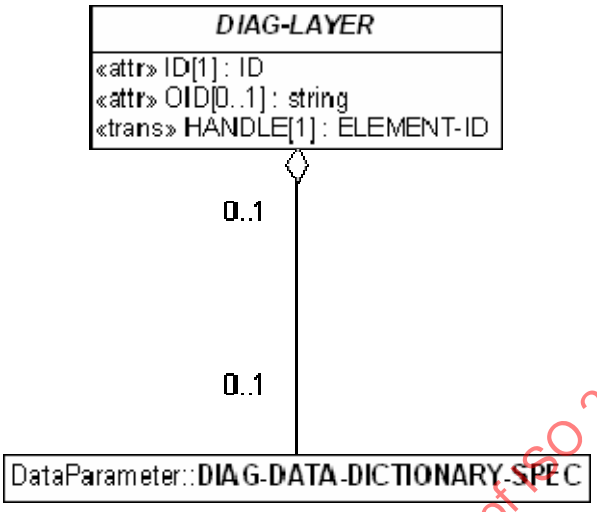


Figure 185 — UML example of composition and aggregation

8.2.2.1.3 XML schema example

```
<xsd:complexType name="DIAG-LAYER">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="0" type="DIAG-DATA-DICTIONARY-SPEC"
                 name="DIAG-DATA-DICTIONARY-SPEC"/>
  </xsd:sequence>
  ...
</xsd:complexType>
```

8.2.2.1.4 ODX instance example

```
<PROTOCOL ID="A">
    <SHORT-NAME>A</SHORT-NAME>
    <DIAG-DATA-DICTIONARY-SPEC/>
    <COMPARAM-SPEC-REF ID-REF="A"/>
</PROTOCOL>
```

8.2.2.2 Wrapper elements

8.2.2.2.1 Description

If the upper bound of the specified multiplicity of an association exceeds 1, the association will be mapped to either one or multiple wrapper elements, which are implemented by anonymous type definition. The specific implementation of the wrapper element depends on the generalization concept (see 8.2) being used for the aggregated class.

8.2.2.2.2 Wrapper element for child generalizations

8.2.2.2.2.1 Description

Due to the fact that wrapper elements have been introduced to group elements of identical type, it requires multiple wrapper elements to group the distinct elements resulting from child generalization. I.e. there will be one wrapper element for each most specialized class within an inheritance tree using child generalization.

8.2.2.2.2.2 UML example

Figure 186 — UML representation of wrapper for child generalizations illustrates the wrapper element using child generalization in UML.

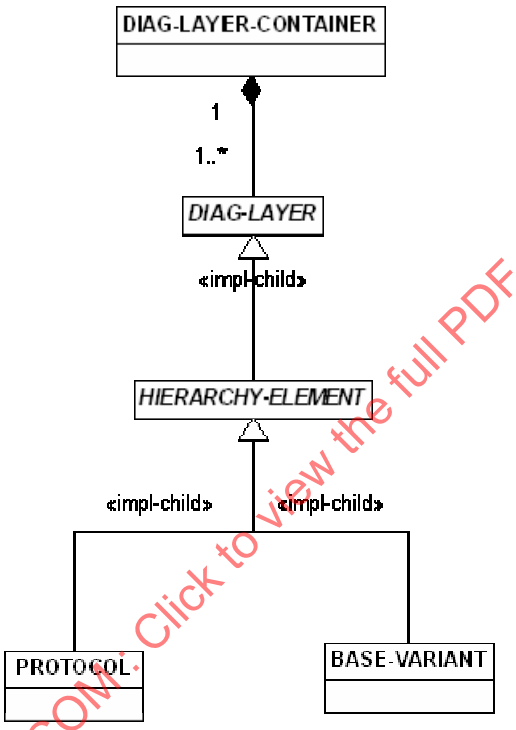


Figure 186 — UML representation of wrapper for child generalizations

8.2.2.2.2.3 XML schema example

```
<xsd:complexType name="DIAG-LAYER-CONTAINER">
  <xsd:complexContent>
    <xsd:extension base="ODX-CATEGORY">
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="PROTOCOLS">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element maxOccurs="unbounded" minOccurs="1" type="PROTOCOL"
                           name="PROTOCOL"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        ...
        <xsd:element maxOccurs="1" minOccurs="0" name="BASE-VARIANTS">
          <xsd:complexType>
            <xsd:sequence>
```

```
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="BASE-VARIANT"
                         name="BASE-VARIANT"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      ...
    </xsd:sequence>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

8.2.2.2.2.4 ODX instance example

```
<DIAG-LAYER-CONTAINER ID="A">
    <SHORT-NAME>A</SHORT-NAME>
    <PROTOCOLS>
        <PROTOCOL ID="ID04Mar26100629">
            <SHORT-NAME>B</SHORT-NAME>
            <COMPARAM-SPEC-REF ID-REF="A"/>
        </PROTOCOL>
    </PROTOCOLS>
    <BASE-VARIANTS>
        <BASE-VARIANT ID="ID04Mar26100633">
            <SHORT-NAME>C</SHORT-NAME>
        </BASE-VARIANT>
    </BASE-VARIANTS>
</DIAG-LAYER-CONTAINER>
```


8.2.2.2.3 Wrapper element for parent generalizations

8.2.2.2.3.1 Description

As stated above, wrapper elements have been introduced to group elements of identical type. In case of the parent generalization there exist only elements of the most general type. Therefore a single wrapper element for all specializations is sufficient.

8.2.2.2.3.2 UML example

Figure 187 — UML representation of wrapper for parent generalizations illustrates the wrapper for parent generalizations in UML.

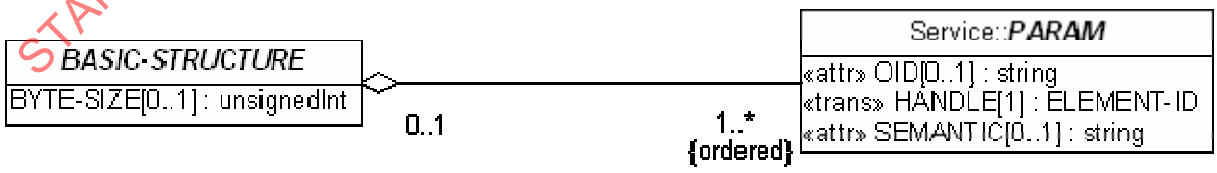


Figure 187 — UML representation of wrapper for parent generalizations

8.2.2.2.3.3 XML schema example

```
<xsd:complexType name="BASIC-STRUCTURE">
  ...
  <xsd:sequence>
```

```
    ...
    <xsd:element maxOccurs="1" minOccurs="1" name="PARAMS">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="1" type="PARAM" name="PARAM"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    ...
  </xsd:sequence>
  ...
</xsd:complexType>
```

8.2.2.2.3.4 ODX instance example

```
<STRUCTURE ID="A">
    <SHORT-NAME>A</SHORT-NAME>
    <PARAMS>
        <PARAM xsi:type="VALUE">
            <SHORT-NAME>B</SHORT-NAME>
            <DOP-SNREF SHORT-NAME="C"/>
        </PARAM>
    </PARAMS>
</STRUCTURE>
```

8.2.2.2.4 No wrapper

8.2.2.2.4.1 Description

There are some cases where the presence of a wrapper element is not desirable. This circumstance is being denoted by the **«nowrapper»** constraint.

8.2.2.2.4.2 UML example

Figure 188 — UML representation of no wrapper element illustrates no wrapper element in UML.

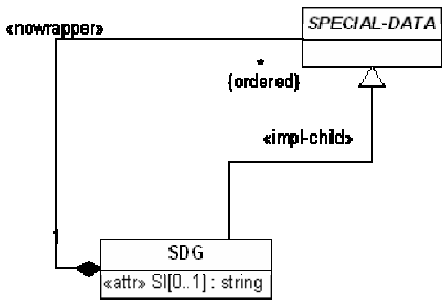


Figure 188 — UML representation of no wrapper element

8.2.2.2.4.3 XML schema Example

```
<xsd:complexType name="SDG">
  <xsd:complexContent>
    <xsd:extension base="SPECIAL-DATA">
      <xsd:sequence>
```

```
        <xsd:element maxOccurs="unbounded" minOccurs="0" type="SDG" name="SDG"/>
        ...
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

8.2.2.2.4.4 ODX instance example

```
<SDG>
  <SDG/>
  <SDG/>
</SDG>
```

8.2.2.2.5 Single wrapper for different elements

8.2.2.2.5.1 Description

In several other cases there was a requirement to collect different elements within a single wrapper element that is being shown by a constraint attached to all affected associations. The body of the constraint contains also the name of the single wrapper element and the total minimum multiplicity. The common-wrapper constraint takes precedence over the default wrapper definition.

The rules for name generation apply as usual, e.g. -REFS will be appended to the name of the wrapper element. All associations that participate in a common wrapper constraint shall have the same *prio*-number.

The common-wrapper constraint works only for identical associations.

8.2.2.2.5.2 UML example

Figure 189 — UML representation of common wrapper element illustrates the representation of common wrapper in UML.

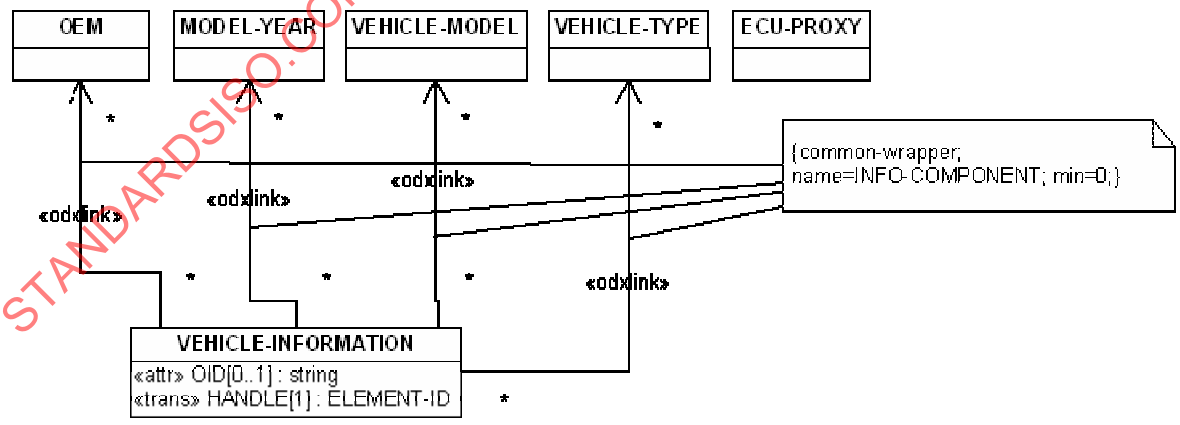


Figure 189 — UML representation of common wrapper element

8.2.2.2.5.3 XML schema example

```
<xsd:complexType name="VEHICLE-INFORMATION">
  <xsd:sequence>
    ...
    <xsd:element maxOccurs="1" minOccurs="1" name="INFO-COMPONENT-REFS">
```

```
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="1" name="INFO-COMPONENT-REF"
                       type="ODXLINK"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    ...
  </xsd:sequence>
</xsd:complexType>
```

8.2.2.2.5.4 ODX instance example

```
<VEHICLE-INFORMATION>
    <SHORT-NAME>A</SHORT-NAME>
    <INFO-COMPONENT-REFS>
        <INFO-COMPONENT-REF ID-REF="ID001"/>
        <INFO-COMPONENT-REF ID-REF="ID002"/>
    </INFO-COMPONENT-REFS>
</VEHICLE-INFORMATION>
```

9 Packaged ODX data (PDX)

9.1 Overview

9.1.1 General

The exchange of ODX data can be realized with a file container called “PDX package”. All files to be exchanged are packaged in one PDX package. A PDX package can handle ODX-files of different types with the following file extensions:

— odx-c, odx-cs (for COMPARAM-SPEC, COMPARAM-SUBSET);

— odx-d (for DIAG-LAYER-CONTAINER);

— odx-e (for ECU-CONFIG);

— odx-f (for FLASH);

— odx-fd (for FUNCTION-DICTIONARY);

— odx-m (for MULTIPLE-ECU-JOB-SPEC);

— odx-v (for VEHICLE-INFORMATION-SPEC);

— odx (alternatively all the files containing ODX data can use the extension “odx” lesser restrictive).

The PDX-package may not only contain ODX data, text or pictures, but arbitrary files of just any format.

The content of a PDX package is described in a separate XML document called “PDX package catalogue”. The PDX package catalogue itself contains no ODX data. It is used as a repository for meta-data in the exchange process.

All file names for files containing ODX data or involved in the ODX/PDX exchange process shall be handled in a case-sensitive manner by all ODX-compliant tools. The file-extensions shall always be expressed in small letters. The filename of the PDX package catalogue shall always be “index.xml” in small letters.

9.1.2 PDX package/package catalogue use cases

PDX package/package catalogue use cases are as follows:

- mutual data exchange between OEM and partners:
  - full data exchange (one way or two ways);
  - incremental (partial) data exchange: only newly created or changed data is exchanged;
- configuration management and version control:
  - exchange of revision history;
  - support for CMS-specific versioning [30]: each process partner can add CMS-internal revision information independently;
- multi-versioning: different revisions of a file may be provided in a single PDX package.

9.2 Structure of PDX package

9.2.1 Structure of PDX package catalogue

A PDX package consists of a collection of files. The content of the PDX package is described in the PDX package catalogue, which is one of the files that are stored in the PDX package. Figure 190 — UML representation and structure of the PDX package catalogue illustrates the modelling in UML.

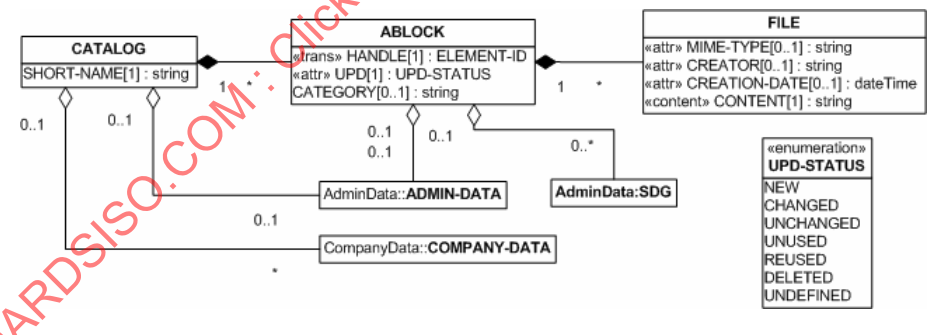


Figure 190 — UML representation and structure of the PDX package catalogue

A PDX package catalogue is represented by a CATALOG instance; it provides a list of all files that are stored in the corresponding PDX package. These file entries can be grouped logically. As an example, ODX data files may be stored in one group, and files for documentation (manuals, images, etc.) in another one.

For this purpose the CATALOG contains a list of so-called ABLOCKs; each ABLOCK represents a logical group and may contain an arbitrary number of FILE members each representing a single file in the PDX package. As an option, a CATEGORY member may be specified for an ABLOCK. This allows the definition of process-specific criteria for the categorization of files.

30) CMS: Configuration Management System.

The following values are predefined for CATEGORY:

a) ODX-DATA for all files that are derived from an ODX-CATEGORY;

b) ODX-JOB for all job code files (e.g. java-files, class-files, jar-files, dll-files);

c) LIB for all libraries that can be used or are imported by jobs;

d) PROGRAMMING-DATA for binary/hex files that are used for ECU programming and are referenced from an ECU-MEM.

The predefined list of CATEGORYs can be extended according to the actual use case.

The SHORT-NAME is a mandatory member of each ABLOCK. It identifies the logical group that the files in the ABLOCK belong to rather than the ABLOCK itself. Multiple ABLOCKs with the same SHORT-NAME may exist in one PDX package catalogue. Each ABLOCK may contain an optional ADMIN-DATA member to store revision information that is specific for the ABLOCK, i.e. it refers to all FILEs in the ABLOCK. ADMIN-DATA that is valid for all files in the PDX package and other global meta-data like COMPANY-DATA can be stored directly in the CATALOG (see 7.1.2.3 for a detailed description of ADMIN-DATA and COMPANY-DATA).

A ABLOCK has a mandatory UPD member that describes the update status of the ABLOCK's files within the data exchange process.

The UPD attribute can have one of the following values:

— NEW: new files have been introduced;

— CHANGED: the files have been changed;

— UNCHANGED: the files have not been changed;

— UNUSED: the files are no longer used at the moment but may be used again in the future;

— REUSED: the files that have been marked as UNUSED in an earlier stage of the data exchange process are used again;

— DELETED the files have been deleted;

— UNDEFINED: no update status available: this is the default value.

Each FILE member specifies the file name to locate one file within the PDX package. The MIME-TYPE, the CREATOR, and the CREATION-DATE of such a file can be specified with optional attributes. As values of MIME-TYPE the already standardized MIME types can be used. Since ODX documents are XML documents the MIME-TYPE “text/xml” can be used for them. In order to distinguish between the different types of ODX data, the MIME-TYPEs defined in Table 27 — MIME-TYPEs used in PDX packages (not normative) should be used in catalogues.

Table 27 — MIME-TYPEs used in PDX packages (not normative)

ODX category	File extension	MIME-TYPE
Comparam Spec, Comparam Subset	odx-c, odx-cs	application/x-asam.odx.odx-c, application/x-asam.odx.odx-cs
Diag-Layer Container	odx-d	application/x-asam.odx.odx-d
Vehicle Info Spec	odx-v	application/x-asam.odx.odx-v
Flash	odx-f	application/x-asam.odx.odx-f
Multiple ECU Job	odx-m	application/x-asam.odx.odx-m
ECU Config	odx-e	application/x-asam.odx.odx-e
Functional Dictionary	odx-fd	application/x-asam.odx.odx-fd
Java Jobs	Jar	application/x-asam.odx.jobs.xjar-Archive
Java Jobs Source	Jar	application/x-asam.odx.jobs.xjava-Source

EXAMPLE A catalogue for a PDX package with three files:

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG F-DTD-VERSION="ODX-2.2.0" xsi:noNamespaceSchemaLocation = "odx-cc.xsd" xmlns:xsi =
"http://www.w3.org/2001/XMLSchema-instance">
  <SHORT-NAME>ODX_CC_Example</SHORT-NAME>
  <ABLOCKS>
    <ABLOCK UPD = "NEW">
      <SHORT-NAME>DiagData</SHORT-NAME>
      <CATEGORY>ODX-DATA</CATEGORY>
      <FILES>
        <FILE MIME-TYPE = "text/xml" CREATOR = "xyz" CREATION-DATE = "2005-12-08">DiagData.odx</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK UPD = "NEW">
      <SHORT-NAME>Documentation</SHORT-NAME>
      <CATEGORY>DOCUMENTATION</CATEGORY>
      <FILES>
        <FILE MIME-TYPE = "application/pdf" CREATOR = "xyz" CREATION-DATE = "2005-12-08">odx_spec.pdf</FILE>
        <FILE MIME-TYPE = "text/html" CREATOR = "xyz" CREATION-DATE= "2005-12-08">readme.html</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```


9.2.2 Technical aspects of PDX package

A PDX package is implemented as a ZIP archive with the file extension ".pdx" (for "packaged ODX").

A PDX package contains one file with the PDX package catalogue (see example above). This file is named "index.xml" and is located in the root directory of the PDX package's internal file system hierarchy.

A PDX package is a self-contained file, i.e. the PDX package catalogue references only files within the PDX package. As a consequence, no references to external files are allowed in the PDX package catalogue.

A PDX package supports the storage of files in a hierarchical directory structure. In case of hierarchical file storage, all file paths in the PDX package catalogue are formulated as absolute paths starting with the root directory of the PDX package's internal file system hierarchy (e.g. "zipdirectory1/zipdirectory2/myfile.odx"). In

case of java jobs the full path resulting from the java package structure needs to be included in the PDX file to preserve the unambiguousness of a java job.

Naming conventions for files in PDX packages:

A file name is usually built according to the following scheme:

NAME.EXTENSION

The type of a file is specified explicitly by the MIME-TYPE member. As a consequence, there's no need for tools conforming to ODX to map the file extension to a MIME type. The file extension should be handled as a part of the file name.

To store multiple revisions of the same file within a single ODX container (which is called “multi-versioning” in ODX) each file revision is stored in a separate ABLOCK. All ABLOCKs that contain revisions of the same file share the same SHORT-NAME.

9.3 Usage scenarios

9.3.1 PDX package in the exchange process

9.3.1.1 General

The exchange of ODX data and other supporting files is the main purpose of an PDX package.

A process partner builds the PDX package by including all files he wants to transfer and by describing those in the PDX package catalogue (see Figure 191). The other process partner receives the PDX package and extracts the files; the PDX package catalogue can provide information how these files are to be handled according to the data exchange process.

Figure 191 — Usage of an ODX container in the data exchange process illustrates the ODX container in the data exchange process in a sequence chart.

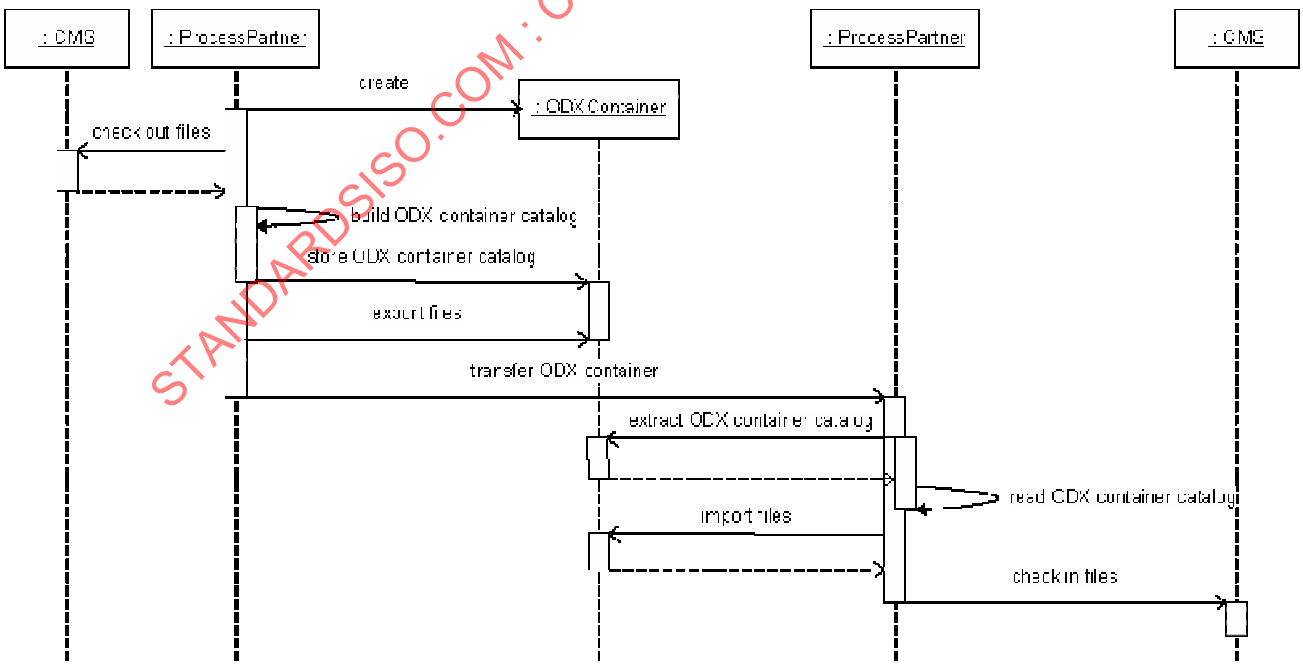


Figure 191 — Usage of an ODX container in the data exchange process

9.3.1.2 Full data exchange

All diagnostic layer instances that are needed for the diagnosis are packaged in one PDX package. Supporting files, e.g. manuals or images, are added to the PDX package.

9.3.1.3 Incremental (partial) data exchange

To reduce the amount of transferred data, only new or changed files are included in the PDX package. All unchanged files are omitted from the PDX package; they are listed in the PDX package catalogue using an ABLOCK with the UPD member set to “UNCHANGED”. Unused/reused files can be handled the same way by setting UPD to UNUSED/REUSED.

The process partners have to agree on a common set of files as a base for the data exchange process. Such a common set may be established by a full data exchange at the beginning of the data exchange process.

NOTE The process partners are free to decide whether they retain ABLOCKs in the PDX package catalogue that were not changed since the last data exchange. Including all unchanged ABLOCKs in the PDX package catalogue finally results in a list of all files involved in the data exchange process so far. Such a list is useful for the documentation of the data exchange process. However, the PDX package catalogue can become very large and difficult to maintain.

9.3.2 Configuration management and version control

9.3.2.1 Overview

ODX is designed to be independent from the used tools and the applied process to support the exchangeability of ODX data. As a consequence, process-specific information like storage structure or file revisions should be kept outside of the ODX data. The PDX package catalogue can be used to transfer this kind of information from one process partner to the other.

Generally speaking, we distinguish two types of revision information: Public revision information can be used by all process partners, while internal revision information is provided for company-internal use only (see 7.1.2.3 for details).

9.3.2.2 Exchange of revision history

In some cases, the complete public revision history of a file shall be exchanged, e.g. if a process partner gets a file for the first time that was changed several times before.

A public revision history of a file is created by adding a new DOC-REVISION member to the ADMIN-DATA of the file's ABLOCK for every new revision of the file. If an ABLOCK contains more than one file, all the files have the same revision history.

To preserve the public revision history of a file the process partners have to retain the file's ABLOCK and its associated ADMIN-DATA until the next file revision is exchanged. Otherwise any previously accumulated revision information would be lost.

EXAMPLE Exchange of revision history:

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" F-DTD-VERSION="ODX-2.2.0"
xsi:noNamespaceSchemaLocation="odx-cc.xsd">
  <SHORT-NAME>ODX_CC_DR_Example</SHORT-NAME>
  <COMPANY-DATAS>
    <COMPANY-DATA ID="abc">
      <SHORT-NAME>abc</SHORT-NAME>
      <LONG-NAME>ABC Corp.</LONG-NAME>
    </COMPANY-DATA>
    <COMPANY-DATA ID="xyz">
      <SHORT-NAME>xyz</SHORT-NAME>
```

```
      <LONG-NAME>XYZ Inc.</LONG-NAME>
      <TEAM-MEMBERS>
        <TEAM-MEMBER ID="jsmith">
          <SHORT-NAME>jsmith</SHORT-NAME>
          <LONG-NAME>John Smith</LONG-NAME>
        </TEAM-MEMBER>
      </TEAM-MEMBERS>
    </COMPANY-DATA>
  </COMPANY-DATAS>
  <ABLOCKS>
    <ABLOCK UPD="CHANGED">
      <SHORT-NAME>DiagData</SHORT-NAME>
      <CATEGORY>ODX-DATA</CATEGORY>
      <ADMIN-DATA>
        <LANGUAGE>en-UK</LANGUAGE>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.0</REVISION-LABEL>
            <STATE>draft</STATE>
            <DATE>2005-12-08T13:55:13</DATE>
          </DOC-REVISION>
          <DOC-REVISION>
            <TEAM-MEMBER-REF ID-REF="jsmith"/>
            <REVISION-LABEL>1.0.1</REVISION-LABEL>
            <STATE>release</STATE>
            <DATE>2005-12-08T15:55:14</DATE>
            <MODIFICATIONS>
              <MODIFICATION>
                <CHANGE>added diagnostic service</CHANGE>
              </MODIFICATION>
            </MODIFICATIONS>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <FILES>
        <FILE MIME-TYPE="text/xml" CREATOR="xyz" CREATION-DATE="2005-12-08">DiagData.odx</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```


9.3.2.3 CMS-specific versioning

If a process partner manages ODX files with a CMS, this CMS typically applies its own mechanism for version control. This includes an own versioning scheme etc. CMS-specific data is stored as internal revision information within the PDX package catalogue to allow accurate version tracking independently from the public revision information.

Whenever the CMS of a process partner updates its internal revision information of a file a new COMPANY-REVISION-INFO member is added to the latest DOC-REVISION of the file. The COMPANY-REF in COMPANY-REVISION-INFO is used to identify the process partner who created that internal revision information. This allows the process partners to store their CMS-specific revision histories in the PDX package catalogue without interfering with each other.

If CMS-specific revision information is needed, a separate ABLOCK should be provided for each file within the PDX package. This way, process partners can update internal revision information independently. As with the public revision history the process partners have to retain the ADMIN-DATA.

EXAMPLE CMS-specific versioning:

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" F-DTD-VERSION="ODX-2.2.0"
xsi:noNamespaceSchemaLocation="odx-cc.xsd">
  <SHORT-NAME>ODX_CC_CRI_Example</SHORT-NAME>
  <COMPANY-DATAS>
    <COMPANY-DATA ID="abc"><SHORT-NAME>abc</SHORT-NAME>
The Example is erroneous. The corrected example is:
<LONG-NAME>ABC Corp.</LONG-NAME></COMPANY-DATA>
    <COMPANY-DATA ID="xyz">
      <SHORT-NAME>xyz</SHORT-NAME>
      <LONG-NAME>XYZ Inc.</LONG-NAME>
    </COMPANY-DATA>
  </COMPANY-DATAS>
  <ABLOCKS>
    <ABLOCK UPD="CHANGED">
      <SHORT-NAME>DiagData</SHORT-NAME>
      <CATEGORY>ODX-DATA</CATEGORY>
      <ADMIN-DATA>
        <LANGUAGE>en-US</LANGUAGE>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.0</REVISION-LABEL>
            <STATE>draft</STATE>
            <DATE>2005-12-08T15:55:13</DATE>
            <COMPANY-REVISION-INFOS>
              <COMPANY-REVISION-INFO>
                <COMPANY-DATA-REF ID-REF="xyz"/>
                <REVISION-LABEL>01_00</REVISION-LABEL>
              </COMPANY-REVISION-INFO>
            </COMPANY-REVISION-INFOS>
          </DOC-REVISION>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.1</REVISION-LABEL>
            <STATE>release</STATE>
            <DATE>2005-12-08T15:55:14</DATE>
            <COMPANY-REVISION-INFOS>
              <COMPANY-REVISION-INFO>
                <COMPANY-DATA-REF ID-REF="xyz"/>
                <REVISION-LABEL>01_05</REVISION-LABEL>
              </COMPANY-REVISION-INFO>
            </COMPANY-REVISION-INFOS>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <FILES>
        <FILE MIME-TYPE="text/xml" CREATOR="xyz" CREATION-DATE="2005-12-08">DiagData.odx</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```


9.3.2.4 Using the UPD attribute for configuration management

During the export of a file to a PDX package, the container creator shall set the UPD attribute of the ABLOCK that contains the file in the PDX package catalogue. To determine the correct update status of a file the

container creator compares the current revision of the file to the last revision that was exchanged with the process partner. The revision information itself can be omitted from the file's ABLOCK if it is not needed for other purposes (e.g. for the revision history).

In an incremental data exchange process, the attribute values UNCHANGED, UNUSED, REUSED, and DELETED imply that the files listed in this ABLOCK do not have to be exported to the PDX package because the current revisions of the files were transferred during a previous data exchange. The container creator can include the files as a reference for the receiver of the PDX package.

The receiver of a PDX package may use the UPD member of an ABLOCK as a hint how to import the files in the ABLOCK (e.g. to a CMS). This is useful if the PDX package catalogue provides no other revision information.

The import of a file could result in a versioning conflict if the receiver of the PDX package has changed this file since the last data exchange. Such a conflict shall be resolved manually by the process partners.

EXAMPLE Usage of UPD attribute in a complex data exchange process.

The process partner “ABC” creates a PDX package with the revision 1.0.0 of the file “DiagData.odx” and the following PDX package catalogue:

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" F-DTD-VERSION="ODX-2.2.0"
xsi:noNamespaceSchemaLocation="odx-cc.xsd">
  <SHORT-NAME>ODX_CC_UPD_Example_1</SHORT-NAME>
  <ABLOCKS>
    <ABLOCK UPD="NEW">
      <SHORT-NAME>DiagData</SHORT-NAME>
      <CATEGORY>ODX-DATA</CATEGORY>
      <ADMIN-DATA>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.0</REVISION-LABEL>
            <DATE>2005-12-08T15:55:14</DATE>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <FILES>
        <FILE MIME-TYPE="text/xml" CREATOR="abc" CREATION-DATE="2005-12-08">DiagData.odx</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```

The process partner “XYZ” imports “DiagData.odx” from the received PDX package. He changes this file and exports it as revision 1.0.1 to the PDX package. He also adds two files “odx_spec.pdf” and “readme.html” without further revision information. The resulting PDX package catalogue may look like this:

```
    <ABLOCK UPD="CHANGED">
      <SHORT-NAME>DiagData</SHORT-NAME>
      <CATEGORY>ODX-DATA</CATEGORY>
      <ADMIN-DATA>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.0</REVISION-LABEL>
            <DATE>2005-12-08T15:55:14</DATE>
          </DOC-REVISION>
          <DOC-REVISION>
```

```
            <REVISION-LABEL>1.0.1</REVISION-LABEL>
            <DATE>2005-12-08T17:25:04</DATE>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <FILES>
        <FILE MIME-TYPE="text/xml" CREATOR="xyz" CREATION-DATE="2005-12-08">DiagData.odx</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK UPD="NEW">
      <SHORT-NAME>Documentation</SHORT-NAME>
      <ADMIN-DATA>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.0</REVISION-LABEL>
            <DATE>2005-12-08T17:25:04</DATE>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <FILES>
        <FILE MIME-TYPE="application/pdf" CREATOR="xyz" CREATION-DATE="2005-12-08">odx_spec.pdf</FILE>
        <FILE MIME-TYPE="text/html" CREATOR="xyz" CREATION-DATE="2005-12-08">readme.html</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```

Checking the UPD attribute “ABC” notices that “DiagData.odx” was changed; he saves the revision 1.0.0 of this file before he imports the new revision and the other new files from the PDX package.

After changing the file “readme.html” “ABC” stores the new revision of this file in a PDX package; the other files have not changed and are omitted from the PDX package. The content of the PDX package is described as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" F-DTD-VERSION="ODX-2.2.0"
xsi:noNamespaceSchemaLocation="odx-cc.xsd">
  <SHORT-NAME>ODX_CC_UPD_Example_3</SHORT-NAME>
  <ABLOCKS>
    <ABLOCK UPD="UNCHANGED">
      <SHORT-NAME>DiagData</SHORT-NAME>
      <CATEGORY>ODX-DATA</CATEGORY>
      <ADMIN-DATA>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.0</REVISION-LABEL>
            <DATE>2005-12-08T15:55:14</DATE>
          </DOC-REVISION>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.1</REVISION-LABEL>
            <DATE>2005-12-08T17:25:04</DATE>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <FILES>
        <FILE MIME-TYPE="text/odx" CREATOR="xyz" CREATION-DATE="2005-12-08">DiagData.odx</FILE>
      </FILES>
    </ABLOCK>
    <ABLOCK UPD="CHANGED">
```

```
      <SHORT-NAME>Documentation</SHORT-NAME>
      <ADMIN-DATA>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.0</REVISION-LABEL>
            <DATE>2005-12-08T17:25:04</DATE>
          </DOC-REVISION>
          <DOC-REVISION>
            <REVISION-LABEL>1.0.1</REVISION-LABEL>
            <DATE>2005-12-08T19:56:59</DATE>
            <MODIFICATIONS>
              <MODIFICATION>
                <CHANGE>readme.html was revised</CHANGE>
              </MODIFICATION>
            </MODIFICATIONS>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <FILES>
        <FILE MIME-TYPE="application/pdf" CREATOR="xyz" CREATION-DATE="2005-12-08">odx_spec.pdf</FILE>
        <FILE MIME-TYPE="text/html" CREATOR="abc" CREATION-DATE="2005-12-08">readme.html</FILE>
      </FILES>
    </ABLOCK>
  </ABLOCKS>
</CATALOG>
```

Annex A
(normative)

Enumerations and pre-defined values

A.1 Predefined values of SEMANTICs

Table A.1 — SEMANTIC at DIAG-COMM defines the meaning of the normative values of SEMANTIC at DIAG-COMM.

Table A.1 — SEMANTIC at DIAG-COMM

Pre-defined value	Description
COMMUNICATION	To show that communication between tester and ECU is meant (Used with DIAGNOSTIC-CLASS="STARTCOM" and DIAGNOSTIC-CLASS="STOPCOM")
SESSION	Switch a diagnostic session
SECURITY	Enable security access
IDENTIFICATION	Read ECU or flash identification
FAULTREAD	Read diagnostic trouble codes
FAULTCLEAR	Clear diagnostic trouble codes
DEFAULT-FAULT-CLEAR	Clear diagnostic trouble codes (may only appear once)
ENVREAD	Read environment data of diagnostic trouble codes
VARCODING	Job or service for variant coding
STOREDDATA	Read/write stored data
CURRENTDATA	Read current data
MEMORY	Read/write ECU memory
CONTROL	Start/stop routine, request routine results
CALIBRATION	Job or Service for ECU calibration
ROUTINE	Start/stop a routine and/or request routine results
FUNCTION	Execute function like enable/disable normal message transmission or reset ECU
DOWNLOAD	Job or service to request a download data transfer (to the ECU)
UPLOAD	Job or service to request an upload data transfer (from the ECU)
FLASHJOB	Job used for reprogramming of ECU memory
TESTERPRESENT	service to keep an active diagnostic session alive (may only appear once)
DEFINE-DYN-MESSAGE	Service defines a DYN-ID record
READ-DYN-DEF-MESSAGE	Service reads data using a DYN-ID
MEMORYREAD	Job or service for reading ECU memory, e.g. Read Memory By Address Service
MEMORYWRITE	Job or service for writing ECU memory, e.g. Write Memory By Address Service
STOREDDATAREAD	Job or service for reading stored data, e.g. Read Data By Identifier Service
STOREDDATAWRITE	Job or service for writing stored data, e.g. Read Write By Identifier Service
VARIANTCODINGREAD	Job or service for reading variant coding data

Table A.1 (*continued*)

Pre-defined value	Description
VARIANTCODINGWRITE	Job or service for writing variant coding data
EVENTREAD	Job or service for interpretation of event triggered ECU responses
EVENTCLEAR	Reset event logic in the ECU to stop event triggered responses
DEFAULT-EVENT-READ	Default interpretation of event triggered ECU responses (may only appear once)
DEFAULT-EVENT-CLEAR	Reset default event logic in the ECU (may only appear once)

Table A.2 — SEMANTIC at PARAM defines the meaning of the normative values of SEMANTIC at PARAM.

Table A.2 — SEMANTIC at PARAM

Pre-defined value	Description
ID	Identifier
SERVICE-ID	Service identifier
SUBFUNCTION	Subfunction of service identifier
CONTROL	Control parameter
DATA	Data
LOCAL-ID	Local identifier
COMMON-ID	Common identifier
CONTROLPARAMETER-ID	Control parameter identifier
DATA-ID	Data identifier
DATAPACKAGE-ID	Data package identifier
PARAMETER-ID	Parameter identifier
DYN-ID	Dynamically defined identifier within a request and a response

Table A.3 — SEMANTIC at TABLE defines the meaning of the normative values of SEMANTIC at TABLE.

Table A.3 — SEMANTIC at TABLE

Pre-defined value	Description
DATA-ID	Data identifier
PACKAGE-ID	Package identifier
PARAMETER-ID	Parameter identifier
LOCAL-ID	Local identifier
COMMON-ID	Common identifier
ADDRESS	Memory address in the ECU

A.2 Values of extendable enumerations

Table A.4 — Enumeration of SYSPARAM at SYSTEM defines the meaning of the normative values of SYSPARAM at SYSTEM.

Table A.4 — Enumeration of SYSPARAM at SYSTEM

Value	Description
TIMEZONE	Count of minutes to UTC
YEAR	Indication of year
MONTH	Indication of month
DAY	Indication of day
HOUR	Indication of hour (0-23)
MINUTE	Indication of minute (0-59)
SECOND	Indication of second (0-59)
TESTERID	Identification identifier of the tester
USERID	Identification identifier of the user
CENTURY	Indication of century
WEEK	Indication of week

Table A.5 — Enumeration of TYPE at DATABLOCK defines the meaning of the normative values of TYPE at DATABLOCK.

Table A.5 — Enumeration of TYPE at DATABLOCK

Value	Description
BOOT	Data block of boot memory
CODE	Data block of code memory
DATA	Data block of data memory

Table A.6 — Enumeration of PARAM-CLASS at COMPARAM defines the meaning of the normative values of PARAM-CLASS at COMPARAM.

Table A.6 — Enumeration of PARAM-CLASS at COMPARAM

Value	Description
TIMING	Message flow timing parameters, e.g. inter byte time or time between request and response.
INIT	Parameters for initiation of communication e.g. trigger address or wakeup pattern. These parameters shall not be overwritten within ECU-Variant layer in any way.
COM	General communication parameter
ERRHDL	Parameter defining the behaviour of the D-server in case an error occurred, e.g. D-server could either continue communication after a timeout was detected, or stop and reactivate.
BUSTYPE	This is used to define bustype specific parameters (e.g. baud rate). Most of these parameters affect the physical hardware. These parameters can only be modified by the first LOGICAL-LINK that acquired the physical resource. When a second LOGICAL-LINK is created for the same resource, these parameters that were previously set will be active for the new LOGICAL-LINK.

Table A.6 (*continued*)

Value	Description
UNIQUE_ID	This type of ComParam is used to indicate to both the ComLogicalLink and the Application that the information is used for protocol response handling from a physical or functional group of ECUs to uniquely define an ECU response.
TESTER_PRESENT	This type of communication parameter is used to indicate to a server (or servers) that a client is still connected to the vehicle and that certain diagnostic services and/or communication that have been previously activated are to remain active.

Table A.7 — Enumeration "PHYSICAL-LINK-TYPE" defines the meaning of the normative values of PHYSICAL-LINK-TYPE.

Table A.7 — Enumeration "PHYSICAL-LINK-TYPE"

Value	Description
ISO_11898_2_DWCAN	ISO 11898-2, *Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit*
ISO_11898_3_DWFTCAN	ISO 11898-3, *Road vehicles — Controller area network (CAN) — Part 3: Low-speed, fault-tolerant, medium-dependent interface*
ISO_11992_1_DWCAN	ISO 11992-1, *Road vehicles — Interchange of digital information on electrical connections between towing and towed vehicles — Part 1: Physical and data-link layers*
ISO_9141_2_UART	ISO 9141-2:1994/Amd.1:1996, *Road vehicles — Diagnostic systems — Part 2: CARB requirements for interchange of digital information — Amendment 1*
ISO_14230_1_UART	ISO 14230-1, *Road vehicles — Diagnostic systems — Keyword Protocol 2000 — Part 1: Physical layer*
ISO_11898_RAW	ISO 11898-1, *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*
SAE_J1850_VPW	SAE J1850, *Class B data communications network interface* (GM Class 2)
SAE_J1850_PWM	SAE J1850, *Class B data communications network interface* (Ford SCP)
SAE_J2610_UART	SAE J2610, *Serial data communication interface* (Chrysler SCI bus type)
SAE_J1708_UART	SAE J1708, *Serial data communications between microcomputer systems in heavy-duty vehicle applications*
SAE_J1939_11_DWCAN	SAE J1939, *Recommended practice for a serial control and communications vehicle network* — Physical Layer, 250K bits/s, Twisted Shielded Pair (Dual Wire CAN)
GMW_3089_SWCAN	GMLAN Single Wire CAN Physical and Data Link Layer Specification
XDE_5024_UART	SERIAL DATA BUS FOR COMMUNICATIONS BETWEEN MICROCOMPUTER ASSEMBLIES (UART with Delco Electronics SXR 5V interface)
CCD_UART	SAE 880586 Chrysler Collision Detection (CU2d) Bus Interface, Integrate Circuit User Manual (UART with 5V differential interface circuit at 7812.5 bits/s)

A.3 Values of non-extendable enumerations

Table A.8 — Enumeration "STANDARDISATION-LEVEL" defines all possible values and their meaning of properties of type STANDARDISATION-LEVEL.

Table A.8 — Enumeration "STANDARDISATION-LEVEL"

Value	Description
STANDARD	The communication parameter belonging to a standardized protocol (or defined by the ODX working group) shall be supported by a D-server implementing this corresponding standardized protocol. This one cannot execute diagnostic data using a protocol not supported by the D-server.
OEM-SPECIFIC	The communication parameter is part of a non-standardized OEM-specific protocol; nevertheless it is required to be implemented by the runtime system. Diagnostic data using an OEM-specific protocol not supported by the runtime system cannot be executed by the runtime system. If the runtime system detects an unsupported communication parameter of type OEM-SPECIFIC, it shall stop the execution of the diagnostic data and provide an error message.
OPTIONAL	This communication parameter does not have to be supported by the D-server. If a DIAG-COMM uses an unsupported comparam of this type the comparam can be ignored and the DIAG-COMM can be executed nevertheless.
OEM-OPTIONAL	This communication parameter covers the OEM specific comparams that need not to be supported by the runtime system.

Table A.9 — Enumeration "DIAG-CLASS-TYPE" defines all possible values and their meaning of properties of type DIAG-CLASS-TYPE.

Table A.9 — Enumeration "DIAG-CLASS-TYPE"

Value	Description
STARTCOMM	Starts the communication.
STOPCOMM	Stops the communication.
VARIANTIDENTIFICATION	Used for identification of the ECU variant.
READ-DYN-DEF-MESSAGE	Reads the data via DYN-ID.
DYN-DEF-MESSAGE	Creates a DYN-ID.
CLEAR-DYN-DEF-MESSAGE	Deletes a DYN-ID.

Table A.10 — Enumeration "ADDRESSING" defines all possible values and their meaning of properties of type ADDRESSING.

Table A.10 — Enumeration "ADDRESSING"

Value	Description
FUNCTIONAL	Only functional addressing is supported.
PHYSICAL	Only physical addressing is supported. Only an individual ECU can be addressed in this mode.
FUNCTIONAL-OR-PHYSICAL	Both modes are supported.

Table A.11 — Enumeration "ROW-FRAGMENT" defines all possible values and their meaning of properties of type ROW-FRAGMENT.

Table A.11 — Enumeration "ROW-FRAGMENT"

Value	Description
KEY	The parameter acts like a PHYS-CONST with the TABLE-ROW's KEY as value and the TABLE's KEY-DOP as DOP.
STRUCT	The parameter acts like a VALUE parameter that references the TABLE-ROW's STRUCTURE as DOP.

Table A.12 — Enumeration "COMM-RELATION-VALUE-TYPE" defines all possible values and their meaning of properties of type COMM-RELATION-VALUE-TYPE.

Table A.12 — Enumeration "COMM-RELATION-VALUE-TYPE"

Value	Description
CURRENT	The value should be retrieved from the tester/ECU every time when the DIAG-VARIABLE is accessed. The D-server sends/receives the actual value. This is the default if no VALUE-TYPE is specified.
STORED	When a changed value is retrieved from the tester, the D-server should store this value for further use. The stored value is sent to the ECU.
STATIC	The value should be retrieved once from the tester/ECU when the DIAG-VARIABLE is accessed for the first time. After that the value cannot be changed anymore. An example for a static value is an ECU-internal identifier.
SUBSTITUTED	These values are neither current nor stored but values delivered by the ECU as substituted (or intermediate) values.
MEASUREMENT	The values describes measurement services, tables etc.

Table A.13 — Enumeration "COMPU-CATEGORY" defines all possible values and their meaning of properties of type COMPU-CATEGORY.

Table A.13 — Enumeration "COMPU-CATEGORY"

Value	Description
IDENTICAL	Just passes on the input value so that the internal value and the physical one are identical.
LINEAR	COMPU-METHODs of type LINEAR multiply the internal value with a factor and add on an offset. Exactly one COMPU-SCALE shall be defined.
SCALE-LINEAR	Similar to LINEAR. But multiple COMPU-SCALEs can be defined.
TEXTTABLE	The type TEXTTABLE transforms the internal value into a textual expression via mapping defined by this computational method.
COMPUCODE	A computer program coded in Java programming language does the computation.
TAB-INTP	A COMPU-METHOD of type TAB-INTP defines a set of points and the physical value is to be calculated via linear interpolation.
RAT-FUNC	A rational function differs from the linear function in the omission of the restrictions for COMPU-NUMERATORs and COMPU-DENOMINATORs. They can have as many V-values as needed.
SCALE-RAT-FUNC	Similar to RAT-FUNC. But multiple COMPU-SCALEs can be defined.

Table A.14 — Enumeration “DATA-TYPE” defines all possible values and their meaning of properties of type DATA-TYPE.

Table A.14 — Enumeration “DATA-TYPE”

Value	Description
A_INT32	32 bit signed integer (range: -2^31.. 2^31-1)
A_UINT32	32 bit unsigned integer (range: 0 .. 2^32-1)
A_FLOAT32	32 bit float (IEEE 754 single precision)
A_FLOAT64	64 bit float (IEEE 754 double precision)
A_ASCIISTRING	ISO-LATIN-1 (ISO/IEC 8859-1) coded character field with maximum length: 2^31-1 bytes
A_UTF8STRING	UTF-8 coded character field with maximum length: 2^31-1 bytes
A_UNICODE2STRING	UNICODE-2 (ISO/IEC 10646 UCS-2) coded character field with maximum length: 2^31-1 characters
A_BYTEFIELD	Byte field with maximum length: 2^32-1

Table A.15 — Enumeration “DATAFORMAT-SELECTION” defines all possible values and their meaning of properties of type DATAFORMAT-SELECTION.

Table A.15 — Enumeration “DATAFORMAT-SELECTION”

Value	Description
INTEL-HEX	Intel hex-format
MOTOROLA-S	Motorola hex-format
BINARY	Unformatted data

Table A.16 — Enumeration “ENCODING” defines all possible values and their meaning of properties of type ENCODING.

Table A.16 — Enumeration “ENCODING”

Value	Description
BCD-P	Packed BCD format --- A BCD digit is packed in 4 bits (nibble) (47 = 0x47).
BCD-UP	Unpacked BCD format --- A BCD digit is mapped to one byte (17 = 0x0107).
1C	One’s complement
2C	Two’s complement
SM	Sign-Magnitude --- The encoding sign-magnitude means the first bit represents the sign (e.g. “1” equals “-”). In the following bits the magnitude of the value is coded.)
UTF-8	UTF-8 Encoding Form. The Unicode encoding form which assigns each Unicode scalar value to an unsigned byte sequence of one to four bytes in length, as specified in Table 3-5, UTF-8 Bit Distribution. (See definition D36 in Section 3.9, Unicode Encoding Forms.) UTF-8 Encoding Scheme. The Unicode encoding scheme that serializes a UTF-8 code unit sequence in exactly the same order as the code unit sequence itself. (See definition D39 in Section 3.10, Unicode Encoding Schemes.)
UCS-2	ISO/IEC 10646 encoding form: Universal Character Set coded in 2 octets. (See Appendix C, Relationship to ISO/IEC 10646.)

Table A.16 (*continued*)

Value	Description
ISO-8859-1	ISO/IEC 8859-1, Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1
ISO-8859-2	ISO/IEC 8859-2, Information technology — 8-bit single-byte coded graphic character sets — Part 2: Latin alphabet No. 2
WINDOWS-1252	Windows Code Page 1252 (Latin 1)
NONE	In case the BASE-TYPE-ENCODING is not defined explicitly, the “default encoding” shall be applied.

Table A.17 — Enumeration “ENCRYPT-COMPRESS-METHOD-TYPE” defines all possible values and their meaning of properties of type ENCRYPT-COMPRESS-METHOD-TYPE.

Table A.17 — Enumeration “ENCRYPT-COMPRESS-METHOD-TYPE”

Value	Description
A_UINT32	The content of the element shall be interpreted as a type A_UINT32 (e.g. for value comparison, XML interpretation as xsd:unsignedInt).
A_BYTEFIELD	The content of the element shall be interpreted as a type A_BYTEFIELD (e.g. for value comparison, XML interpretation as xsd:hexBinary).

Table A.18 — Enumeration “IDENT-VALUE-TYPE” defines all possible values and their meaning of properties of type IDENT-VALUE-TYPE.

Table A.18 — Enumeration “IDENT-VALUE-TYPE”

Value	Description
A_UINT32	The content of the element shall be interpreted as a type A_UINT32 (e.g. for value comparison, XML interpretation as xsd:unsignedInt).
A_BYTEFIELD	The content of the element shall be interpreted as a type A_BYTEFIELD (e.g. for value comparison, XML interpretation as xsd:hexBinary).
A_ASCIISTRING	The content of the element shall be interpreted as a type A_UNICODE2STRING (e.g. for value comparison, XML interpretation as xsd:string)

Table A.19 — Enumeration “DIRECTION” defines all possible values and their meaning of properties of type DIRECTION.

Table A.19 — Enumeration “DIRECTION”

Value	Description
DOWNLOAD	Data transfer from tester to ECU
UPLOAD	Data transfer from ECU to tester

Table A.20 — Enumeration “INTERVAL-TYPE” defines all possible values and their meaning of properties of type INTERVAL-TYPE.

Table A.20 — Enumeration “INTERVAL-TYPE”

Value	Description
OPEN	The edge value is not to be included.
CLOSED	The edge value is to be included.
INFINITE	The values of LOWER- and UPPER-LIMIT are ignored and -∞ will be used for LOWER-LIMIT and +∞ for UPPER-LIMIT.

Table A.21 — Enumeration “PHYSICAL-DATA-TYPE” defines all possible values and their meaning of properties of type PHYSICAL-DATA-TYPE.

Table A.21 — Enumeration “PHYSICAL-DATA-TYPE”

Value	Description
A_INT32	32 bit signed integer (range: -2^31.. 2^31-1)
A_UINT32	32 bit unsigned integer (range: 0 .. 2^32-1)
A_FLOAT32	32 bit float (IEEE 754 single precision)
A_FLOAT64	64 bit float (IEEE 754 double precision)
A_UNICODE2STRING	UNICODE-2 (ISO/IEC 10646 UCS-2) coded character field with maximum length: 2^31-1 characters
A_BYTEFIELD	Byte field with maximum length: 2^32-1

Table A.22 — Enumeration “RADIX” defines all possible values and their meaning of properties of type RADIX.

Table A.22 — Enumeration “RADIX”

Value	Description
HEX	Hexadecimal number
DEC	Decimal number
BIN	Dual number
OCT	Octal number

Table A.23 — Enumeration “SESSION-SUB-ELEM-TYPE” defines all possible values and their meaning of properties of type SESSION-SUB-ELEM-TYPE.

Table A.23 — Enumeration “SESSION-SUB-ELEM-TYPE”

Value	Description
A_UINT32	The content of the element shall be interpreted as a type A_UINT32 (e.g. for value comparison, XML interpretation as xsd:unsignedInt).
A_BYTEFIELD	The content of the element shall be interpreted as a type A_BYTEFIELD (e.g. for value comparison, XML interpretation as xsd:hexBinary).
A_ASCIISTRING	The content of the element shall be interpreted as a type A_UNICODE2STRING (e.g. for value comparison, XML interpretation as xsd:string)

Table A.24 — Enumeration “TERMINATION” defines all possible values and their meaning of properties of type TERMINATION.

Table A.24 — Enumeration “TERMINATION”

Value	Description
END-OF-PDU	Standard termination by the end of the PDU, for exceptions see 7.3.6.2.
ZERO	Standard termination by a character consisting of binary zeros (e.g. 0x00 for one byte characters, 0x0000 for two byte characters) , for exceptions see 7.3.6.2.
HEX-FF	Standard termination by a character consisting of binary zeros (e.g. 0xff for one byte characters, 0xffff for two byte characters), for exceptions see 7.3.6.2.

Table A.25 — Enumeration “UNIT-GROUP-CATEGORY” defines all possible values and their meaning of properties of type UNIT-GROUP-CATEGORY.

Table A.25 — Enumeration “UNIT-GROUP-CATEGORY”

Value	Description
COUNTRY	Group of units, which are associated to a country-specific unit system.
EQUIV-UNITS	Group of equivalent units, which are used in different countries for the same application domain.

Table A.26 — Enumeration “VALID-TYPE” defines all possible values and their meaning of properties of type VALID-TYPE.

Table A.26 — Enumeration “VALID-TYPE”

Value	Description
VALID	Valid area. Current value is within a valid range and can be presented to user.
NOT-VALID	Invalid area. The ECU cannot process the requested data.
NOT-DEFINED	Indicates an area, which is marked in a specification (e.g. as reserved). Shall usually not be set by the ECU but is used by a tester to verify correct ECU behaviour. Also prevents use of areas during editing process.
NOT-AVAILABLE	Currently invalid area. The value usually is presented by the ECU but can currently not be performed due to e.g. initialisation or temporary problems. This behaviour appears during runtime and cannot be handled while data is edited.

Table A.27 — Enumeration “PIN-TYPE” defines all possible values and their meaning of properties of type PIN-TYPE.

Table A.27 — Enumeration “PIN-TYPE”

Value	Description
HI	“High” line/wire of a two wired physical data link
LOW	“Low” line/wire of a two wired physical data link
K	“K” line/wire of a ISO_9141_2_UART or ISO_14230_1_UART physical data link
L	“L” line/wire of a ISO_9141_2_UART or ISO_14230_1_UART physical data link
TX	“High” line/wire of a two wired physical data link
RX	“High” line/wire of a two wired physical data link
PLUS	“Positive (+)” line/wire of a two wired physical data link
MINUS	“Negative (-)” line/wire of a two wired physical data link
SINGLE	“Single” line/wire of a one wired physical data link (e.g. GMW_3089_SWCAN)
IGNITION-CLAMP	“IGNITION-CLAMP” is needed for D-PDU API section “9.5.8 PDU_IOCTL_SET_PROG_VOLTAGE”

Annex B
(normative)

ODX checker rules

B.1 Overview

Although contained in an annex, this list of checker rules is an essential part of the specification. The purpose of the checker rules is to ensure the interchange ability of ODX-data.

IMPORTANT — The compliance to the checker rules is mandatory for ODX compliant data. An application checking the consistency and conformity of ODX-data should create an error message in case of violation of a checker rule.

Some of the rule descriptions are classified as “[warning]”. In that case, the application should report a warning instead. One and the same erroneous condition could be covered by more than one checker rule. The checking application shall report at least one of these rules. If these rules are classified as “[error]” as well as “[warning]”, at least one of the checker rules with error level shall be reported.

B.2 ODX checker rules

Table B.1 — ODX checker rules includes a checker rule identifier (ID) in the left column to uniquely identify each rule. Some ID numbers are missing due to cancellation of this specific checker rule. A renumbering of checker rules is not desired because of existing ODX checker rule software.

Table B.1 — ODX checker rules

ID	Location	Detailed description	Error Level
0 - 3	Reserved	This range of IDs is reserved by this document.	---
4	DIAG-COMM	The PROTOCOLs that are supported by the DIAG-SERVICE are determined as follows: If at least one PROTOCOL-SNREF exists, the referenced PROTOCOLs are supported. If no PROTOCOL-SNREF exists, all PROTOCOLs within the layer hierarchy of the layer that specifies or imports the DIAG-SERVICE are supported. Every COMPARAM-REF at the DIAG-COMM shall refer to at least one of the COMPARAMs or COMPLEX-COMPARAMs in the COMPARAM-SUBSETs that are referenced by the supported combinations of PROTOCOLs and PROT-STACKs.	error
5	PROG-CODE	For every PROG-CODE element the value of the SYNTAX sub-element is matched against the strings “JAVA”, “CLASS”, and “JAR”.	warning
6	PROG-CODE	For every PROG-CODE element the value of the SYNTAX sub-element is matched against the strings “JAVA” and “CLASS”. If the value does not match one of these strings, it will be checked if the ENTRYPOINT within the same PROG-CODE exists.	warning
7	PARAM of type MATCHING-REQUEST-PARAM	Check for every DIAG-COMM with a defined request byte length for every MATCHING-REQUEST-PARAM element if the value of the REQUEST-BYTE-POS added to BYTE-LENGTH is lesser than or equal to the byte length of the request belonging to the same DIAG-COMM.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
8	PARAM of type SYSTEM	For every SYSTEM element the value of the SYSPARAM attribute is matched against the strings in the column “SYSPARAM” of the table 5 “Coding of system parameter” in the ODX specification. If the value does match one of these strings, the BASE-DATA-TYPE at PHYSICAL-TYPE of the referred DOP shall also match. In Detail: (SYSPARAM - physical data type): + TIMEZONE - A_INT32 + YEAR - A_UINT32 + MONTH - A_UINT32 + DAY - A_UINT32 + HOUR - A_UINT32 + MINUTE - A_UINT32 + SECOND - A_UINT32 + TESTERID - A_BYTEFIELD + USERID - A_BYTEFIELD + CENTURY - A_UINT32 + WEEK - A_UINT32 + TIMESTAMP - A_BYTEFIELD	error
9 - 10	Reserved	This range of IDs is reserved by this document.	---
11	DATA-OBJECT-PROP	PHYSICAL-TYPE/DISPLAY-RADIX is only allowed if PHYSICAL-TYPE/BASE-DATA-TYPE is defined as a A_UINT32. PHYSICAL-TYPE/PRECISION is only allowed if PHYSICAL-TYPE/BASE-DATA-TYPE is defined as A_FLOAT32 or A_FLOAT64.	error
12	DATA-OBJECT-PROP	DIAG-CODED-TYPE/MIN-LENGTH shall be lower or equal than DIAG-CODED-TYPE/MAX-LENGTH.	error
13	Reserved	This range of IDs is reserved by this document.	---
14	DATA-OBJECT-PROP	Each range of SCALE-CONSTR shall not overlap any other range. Two closed range ends with the same limit are prohibited	error
15	END-OF-PDU-FIELD	END-OF-PDU-FIELD/MIN-NUMBER-OF-ITEMS shall be lower or equal than END-OF-PDU-FIELD/MAX-NUMBER-OF-ITEMS.	error
16	MUX	For every CASE element, LOWER-LIMIT/CONTENT shall be less or equal to UPPER-LIMIT/CONTENT of the same CASE. “less or equal” has different meaning for every datatype that can appear for a SWITCH-KEY. Numbers are ordered mathematically.	error
17	DYNAMIC-LENGTH-FIELD	The DATA-OBJECT-PROP referenced by the DETERMINE-NUMBER-OF-ITEMS element: The DATA-OBJECT-PROP's PHYSICAL-TYPE/BASE-DATA-TYPE shall be A_UINT32.	error
18	DTCS	A DTC-DOP holds a set of DTCs that may either be inherited from the set of another DTC-DOP (referenced by LINKED-DTC-DOP excluding the DTCs specified by NOT-INHERITED-DTC-SNREFs), may be referenced via DTC-REF, or that may be specified within the DTC-DOP. Within this set, every DTC/TROUBLE-CODE shall be unique excepting the temporary DTCs.	error
19	ENV-DATA-DESC	For every ENV-DATA-DESC element, the PARAM referenced by PARAM-SNREF shall reference a DATA-OBJECT-PROP or a DTC-DOP with PHYSICAL-TYPE/BASE-DATA-TYPE set to A_UINT 32.	error
20	ENV-DATA-DESC	An ENV-DATA-DESC element contains a set of ENV-DATA elements that are referenced via ENV-DATA-REF. For every ENV-DATA-DESC element this set contains no more than one ENV-DATA, which contains an ALL-VALUE element.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
21	ENV-DATA-DESC	An ENV-DATA-DESC element contains a set of ENV-DATA elements that are referenced via ENV-DATA-REF. DTC-VALUE shall be unique among ENV-DATAs of a common ENV-DATA-DESC.	error
22	DIAG-VARIABLE	**Case 1:** COMM-RELATION refers to a DIAG-SERVICE: The parameter referenced by IN-PARAM-IF-SNREF shall exist in the DIAG-SERVICE's request and the parameter referenced by OUT-PARAM-IF-SNREF shall exist in at least one of the DIAG-SERVICE's POS-RESPONSEs. **Case 2:** COMM-RELATION refers to a SINGLE-ECU-JOB: The parameter referenced by IN-PARAM-IF-SNREF shall exist amongst the SINGLE-ECU-JOB's INPUT-PARAMs and the parameter referenced by OUT-PARAM-IF-SNREF shall exist amongst the SINGLE-ECU-JOB's OUTPUT-PARAMs.	error
23	MATCHING-PARAMETER	MATCHING-PARAMETER references a DIAG-COMM and a OUT-PARAM-IF via SNREF. The SHORT-NAME of the OUT-PARAM-IF-SNREF shall exist in all positive responses in the case of DIAG-SERVICE.	warning
24	COMPARAM-SUBSET	Each ODXLINK shall reference an element within the same COMPARAM-SUBSET structure.	error
25	MATCHING-COMPONENT	MATCHING-COMPONENT references a DIAG-COMM or MULTIPLE-ECU-JOB via odx-link and a OUT-PARAM-IF via SNREF. The SHORT-NAME of the OUT-PARAM-IF-SNREF shall exist in all positive responses in the case of DIAG-SERVICE or in one OUTPUT-PARAM in CASE of the SINGLE-ECU-JOB or MULTIPLE-ECU-JOB.	warning
26	Reserved	This range of IDs is reserved by this document.	---
27	LOGICAL-LINK	In case a BASE-VARIANT or a FUNCTIONAL-GROUP inherits from multiple PROTOCOLs, a LOGICAL-LINK for this BASE-VARIANT or FUNCTIONAL-GROUP shall also refer to exactly one of the PROTOCOL layers. If the implicitly or explicitly selected PROTOCOL supports more than one PROT-STACK, PROT-STACK-SNREF shall specify one of them	error
28	IDENT-DESC	IDENT-DESC references a DIAG-COMM and a OUT-PARAM-IF via shortname reference. In case of a DIAG-SERVICE is referenced: The referenced parameter shall exist in all positive responses of the DIAG-SERVICE. In case of a SINGLE-ECU-JOB is referenced: The OUT-PARAM shall exist at the SINGLE-ECU-JOB.	warning
29	LAYER-REF within ECU-MEM-CONNECTOR	Only ECU-VARIANT and BASE-VARIANT are valid types for reference by the LAYER-REF attribute of ECU-MEM-CONNECTOR. Check that the list of LAYERs referred to only contain these types.	error
30	Reserved	This range of IDs is reserved by this document.	---
31	general	Targets of odx-links shall be contained in the data pool of the current layer. The data pool contains all the instances of the parent layer hierarchy and the imported layers and in case of an ECU-VARIANT all other ECU-VARIANTS belonging to the same BASE-VARIANT.	error
32	DOP TABLE	Direct and indirect cycles of odx-links are not allowed. Direct cyclic references occur when object A refers B and B refers A. Indirect cyclic references occur when A occurs in the transitive closure over A's reference chain. In both cases this would result in an infinite reference loop.	error
33	general	All references may only reference the target as specified by the ODX UML model.	error
34	Reserved	This range of IDs is reserved by this document.	---

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
35	PARAM of type VALUE	PARAM of type VALUE has an optional PHYSICAL-DEFAULT-VALUE element and a DOP-REF/DOP-SNREF which can reference both a COMPLEX-DOP, a DATA-OBJECT-PROP or a DTC-DOP. The optional PHYSICAL-DEFAULT-VALUE can only exist if the referenced DOP is a DATA-OBJECT-PROP or a DTC-DOP.	error
36 - 37	Reserved	This range of IDs is reserved by this document.	---
38	DTC-DOP	The BASE-DATA-TYPE at the PHYSICAL-TYPE shall be set to A_UINT32.	error
39	general	At least one protocol layer shall exist in the inheritance hierarchy of DIAG-LAYERS.	error
40	Reserved	This range of IDs is reserved by this document.	---
41	DIAG-SERVICE	A PARAM with SEMANTIC=SERVICE-ID shall occur at most once within a REQUEST, a POS-RESPONSE, a NEG-RESPONSE and a GLOBAL-NEG-RESPONSE, respectively.	error
42	DIAG-CODED-TYPE	If BASE-DATA-TYPE is A_BYTEFIELD, A_ASCIISTRING, A_UTF8STRING the IS-HIGHLOW-BYTE-ORDER shall not be present except DIAG-CODED-TYPE is of type LEADING-LENGTH-INFO-TYPE.	error
43	DATA-OBJECT-PROP	If INTERNAL-CONSTR is present within element DATA-OBJECT-PROP, DIAG-CODED-TYPE/BASE-DATA-TYPE shall be a ordered data type (A_BYTEFIELD, A_INT32, A_UINT32, A_FLOAT32 or A_FLOAT64).	error
44	MUX/SWITCH-KEY	DATA-OBJECT-PROP/DIAG-CODED-TYPE in element SWITCH-KEY shall be of type STANDARD-LENGTH-TYPE	error
45	Reserved	This range of IDs is reserved by this document.	---
46	MUX/CASE	If a MUX element has no CASES sub-element, it shall have a DEFAULT-CASE sub-element.	error
47	COMPARAM-REF	Separately within every COMPARAM-REFS and for any supported combination of PROTOCOL and PROT-STACK, no COMPARAM should be assigned multiple values, excepting the COMPARAM-REF references a COMPLEX-COMPARAM which defines ALLOW-MULTIPLE-VALUES = “true”.	error
49	DIAG-COMM	For each DIAG-COMM in a layer the parent hierarchy is searched for a DIAG-COMM that has the same SHORT-NAME as the overwriting DIAG-COMM. If a DIAG-COMM with the same SHORT-NAME is found, this DIAG-COMM's attribute IS-FINAL shall be false.	error
50	DIAG-LAYER-CONTAINER	Only the following PARENT-REF relationships are allowed: — PROTOCOL -PARENT-REF-> ECU-SHARED-DATA — FUNCTIONAL-GROUP -PARENT-REF-> ECU-SHARED-DATA — FUNCTIONAL-GROUP -PARENT-REF-> PROTOCOL — BASE-VARIANT -PARENT-REF-> ECU-SHARED-DATA — BASE-VARIANT -PARENT-REF-> PROTOCOL — BASE-VARIANT -PARENT-REF-> FUNCTIONAL-GROUP — ECU-VARIANT -PARENT-REF-> ECU-SHARED-DATA — ECU-VARIANT -PARENT-REF-> BASE-VARIANT In all cases, the PARENT-REF source layer can have multiple PARENT-REFs to DIAG-LAYER types given above. Exception: An ECU-VARIANT can only have one BASE-VARIANT as a parent.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
51	REQUEST	Every bit within a request shall be covered by at most one PARAM.	error
52	COMPU-METHOD of type TAB-INTP	Each COMPU-SCALE of type TAB-INTP shall contain exactly one LOWER-LIMIT and no UPPER-LIMIT.	error
53	COMPU-METHOD of type COMPUCODE	The sub elements COMPU-INTERNAL-TO-PHYS and COMPU-PHYS-TO-INTERNAL shall have the sub element PROG-CODE and only this.	error
54	UNIT	PHYSICAL-DIMENSION-REF, FACTOR-SI-TO-UNIT, OFFSET-SI-TO-UNIT: If one of these elements exists within the UNIT, the both remaining elements shall also exist within the UNIT.	error
55	DIAG-LAYER	A DIAG-LAYER shall not import an ECU-SHARED-DATA by IMPORT-REF, if the same ECU-SHARED-DATA is already inherited by PARENT-REF inside the same DIAG-LAYER or any inherited DIAG-LAYER.	error
56	DIAG-LAYER	Within the intersection of a specialization level and a namespace (see chapter 7.3.2.4 “Value inheritance”) and a boundary (see 7.3.13.5 Boundary for uniqueness of short-name, Table “Boundary for uniqueness of short-name”), any SHORT-NAME shall uniquely define a data object. This rule extends to objects referenced via odx-link within proxy wrappers as well. The ODX data model contains the following proxy wrappers: DIAG-COMM-PROXY, DTC-PROXY, DIAG-VARIABLE-PROXY and TABLE-ROW.	error
57	DATA-OBJECT-PROP	Each range of SCALE-CONSTR shall fit into the INTERNAL-CONSTR range.	error
58	COMM-RELATION DIAG-COMM TABLE-DIAG-COMM-CONNECTOR	The enclosed ODX-LINK reference element may only refer to DIAG-COMMs that are either defined or imported by ODX-LINK within at least one of the DIAG-COMMS wrappers of the enclosing DIAG-LAYERs value inheritance scope or of the same DIAG-LAYER. The names of the reference elements are: — In case of COMM-RELATION: DIAG-COMM-REF; — In case of DIAG-COMM: RELATED-DIAG-COMM-REF; — In case of TABLE-DIAG-COMM-CONNECTOR: DIAG-COMM-REF.	error
59	COMPU-METHOD of type IDENTICAL	Within a DATA-OBJECT-PROP containing a COMPU-METHOD of COMPU-CATAGORY IDENTICAL the PHYSICAL-TYPE/BASE-DATA-TYPE and the DIAG-CODED-TYPE/BASE-DATA-TYPE shall be identical. Exception if the DIAG-CODED-TYPE/BASE-DATA-TYPE is A_ASCIISTRING or A_UTF8STRING the PHYSICAL-TYPE/BASE-DATA-TYPE is A_UNICODE2STRING.	error
60	COMPU-METHOD of type LINEAR, TAB-INTP, SCALE-LINEAR, RAT-FUNC or SCALE-RAT-FUNC	Only datatypes A_INT32, A_UINT32, A_FLOAT32 or A_FLOAT64 are allowed as internal or physical type.	error
61	COMPU-METHOD of type TEXTTABLE	Within a DATA-OBJECT-PROP containing a COMPU-METHOD of COMPU-CATAGORY TEXTTABLE the PHYSICAL-TYPE/BASE-DATA-TYPE shall be A_UNICODE2STRING.	error
62 - 63	Reserved	This range of IDs is reserved by this document.	---

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
64	COMPU-METHOD	For COMPU-METHODs of type IDENTICAL the objects COMPU-PHYS-TO-INTERNAL and COMPU-INTERNAL-TO-PHYS are not allowed. In all other cases at least one of these objects shall be present. The cardinality of COMPU-SCALEs in dependence of the compu-method shall be in both cases as follows: — LINEAR, RAT-FUNC: exactly 1 — SCALE-LINEAR, SCALE-RAT-FUNC: 1..* — TEXTTABLE: 1..* — TAB-INTP: 2..* — IDENTICAL, COMPU-CODE: 0	error
65 - 67	Reserved	This range of IDs is reserved by this document.	---
68	IDENT-DESC	The parameter referenced by OUT-PARAM-IF either shall be CODED-CONST or shall refer to a parameter using DATA-OBJECT-PROP.	error
69	IDENT-IF	The values of attributes TYPE at all IDENT-VALUEs of one EXPECTED-IDENT shall be identical.	error
70	IDENT-DESC	The values of the attributes TYPE at IDENT-VALUEs of the referenced IDENT shall match the PHYSICAL-TYPE/BASE-DATA-TYPE defined in the DOP of the parameter referenced by OUT-PARAM-IF. In the case of a referenced parameter of type CODED-CONST, a compumethod IDENTICAL is assumed to determine the expected physical type. The ident-value type A_ASCIISTRING matches with the physical datatype A_UNICODE2STRING.	error
71	PHYS-SEGMENT of type ADDRDEF-PHYS-SEGMENT	The START-ADDRESS shall contain a value less or equal to END-ADDRESS.	error
72	CHECKSUM	If the CHECKSUM contains SOURCE-END-ADDRESS, the SOURCE-START-ADDRESS shall contain a value less or equal the SOURCE-END-ADDRESS.	error
73	FILTER of type ADDRDEF-FILTER	FILTER-START shall contain a value less or equal to FILTER-END	error
74	SEGMENT	If the SEGMENT contains SOURCE-END-ADDRESS, SOURCE-START-ADDRESS shall contain a value less or equal to SOURCE-END-ADDRESS.	error
75	SECURITY-METHOD, FW-CHECKSUM, VALIDITY-FOR, FW-SIGNATURE, CHECKSUM-RESULT, IDENT-VALUE, ENCRYPT-COMPRESS-METHOD, CONFIG-ID, DATA-ID	The content of these elements shall correspond with the datatype defined by the attribute TYPE. A_UINT32 : xsd:unsignedInt A_BYTEFIELD : xsd:hexBinary else xsd:string	error
76	DATABLOCK	Address ranges of two different SEGMENTs in one DATABLOCK shall not overlap.	error
77	DATABLOCK	Address ranges of two different FILTERs in one DATABLOCK shall not overlap.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
78	DATABLOCK	In case of hex formatted data (INTEL-HEX, MOTOROLA-S) and if at least one SEGMENT and at least one FILTER are defined, each SEGMENT shall correspond to existing data after applying the FILTERs. This means that each SEGMENT shall lie in the interval of one or more adjacent FILTERs defined in the same DATABLOCK.	error
79	Reserved	This range of IDs is reserved by this document.	---
80	CHECKSUM	If the CHECKSUM contains UNCOMPRESSED-SIZE, UNCOMPRESSED-SIZE shall contain a value greater than zero.	error
81	SEGMENT	If the SEGMENT contains UNCOMPRESSED-SIZE, UNCOMPRESSED-SIZE shall contain a value greater than zero.	error
82	Reserved	This range of IDs is reserved by this document.	---
83	PHYS-MEM	Address ranges of two different PHYS-SEGMENTs in one PHYS-MEM shall not overlap.	error
84	DATAFILE	The wildcards “*” and “?” can only be used in DATAFILE/CONTENT if attribute LATEBOUND-DATAFILE=true.	error
85	TABLE	If KEY-DOP is defined it shall refer to a simple DATA-OBJECT-PROP.	error
86 - 88	Reserved	This range of IDs is reserved by this document.	---
89	TABLE	The TABLE-STRUCT parameter shall reference a parameter of type TABLE-KEY in the same request or response. TABLE-STRUCT and TABLE-KEY shall reside in the same STRUCTURE if one of them is an element of a STRUCTURE.	error
90	TABLE	A STRUCTURE containing a parameter TABLE-ENTRY shall not be referred by any parameter except by TABLE-ROW.	error
91	DIAG-COMM	A DIAG-COMM is able to refer to FUNCT-CLASSes defined in different Diag-Layers. The SHORT-NAMEs of the FUNCT-CLASSes referenced by one DIAG-COMM shall be unique.	error
92	SESSION-DESC	The SHORT-NAMEs of all FLASH-CLASSes referenced by one SESSION-DESC shall be unique.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
93	REQUEST POS-RESPONSE, NEG-RESPONSE, GLOBAL-NEG-RESPONSE	If a COMPU-METHOD is used in a response and COMPU-INTERNAL-TO-PHYS is not present, COMPU-PHYS-TO-INTERNAL shall be invertible. If a COMPU-METHOD is used in a request and COMPU-PHYS-TO-INTERNAL is not present, COMPU-INTERNAL-TO-PHYS shall be invertible. 1. IDENTICAL: Condition for invertibility: Always 2. LINEAR: Condition for invertibility: VN1 != 0 or COMPU-INVERSE-VALUE shall be defined. 3. SCALE-LINEAR: Condition for invertibility: The transition of neighbouring intervals is strictly monotonic, in particular in case of discrete values. Adjacent COMPU-SCALES shall have the same values on their common boundaries AND the VN1 of all intervals shall have the same sign or shall be 0. If VN1 = 0, COMPU-INVERSE-VALUE shall be specified. 4. RAT-FUNC: Inversion is not allowed. 5. SCALE-RAT-FUNC: Inversion is not allowed. 6. TEXTTABLE: Condition for invertibility: — COMPU-INTERNAL-TO-PHYS: If the value inside COMPU-CONST is not unique, or LOWER- and UPPER-LIMIT define a range, COMPU-INVERSE-VALUE shall be defined in this COMPU-SCALE. — COMPU-PHYS-TO-INTERNAL: Always 7. TAB-INTP: Condition for invertibility: Always 8. COMPU-CODE: Inversion is not allowed.	error
94 - 95	Reserved	This range of IDs is reserved by this document.	---
96	PROTOCOL	PROTOCOL-SNREFS shall not exist at DIAG-COMMs within PROTOCOL-Layer.	error
97	MUX	The cases belonging to the same MUX have one defined type of SWITCH-KEY. The datatype of the SWITCH-KEY is determined by the PHYSICAL-DATA-TYPE of the DATA-OBJECT-PROP assigned to the SWITCH-KEY. The CONTENT of all CASEs LIMIT elements shall correspond with this datatype of the SWITCH-KEY. For example: If the PHYSICAL-DATA-TYPE/BASE-DATA-TYPE of the SWITCH-KEY is A_INT32, all CASE/LIMIT/CONTENT elements belonging to the same MUX as this SWITCH-KEY shall be interpretable as an A_INT32 value.	error
98	MUX	If the CONTENT of LOWER- and UPPER-LIMIT contains a value of data type A_UNICODE2STRING in context of MUX/CASE both values shall be identical.	error
99	TABLE	The value of TABLE-ROW/KEY shall be unique inside a TABLE. (including referenced TABLE-ROWs).	error
100	MATCHING-PARAMETER	The DIAGNOSTIC-CLASS of a DIAG-COMM referenced by MATCHING-PARAMETER shall be “VARIANTIDENTIFICATION”.	error
101 - 103	Reserved	This range of IDs is reserved by this document.	---
104	DIAG-LAYER	Within one DIAG-LAYER DIAG-COMM with the values STARTCOMM or STOPCOMM as DIAGNOSTIC-CLASS may only exist once, regardless of whether they are inherited, imported or locally defined.	error
105	DIAG-LAYER	Within one DIAG-LAYER only one DIAG-COMM with SEMANTIC = DEFAULT-FAULT-READ may exist, regardless of whether it is inherited, imported or locally defined.	warning

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
106	Reserved	This range of IDs is reserved by this document.	---
107	SESSION	It is prohibited to reference one DATABLOCK more than once within the same SESSION.	error
108	DIAG-COMM GLOBAL-NEG-RESPONSE	The type of all PARAMs inside the request, response and all including complex DOPs and the allowed DOP references are restricted by table 6 “Positioning of PARAM and DOP types” in the ODX specification.	error
109	Reserved	This range of IDs is reserved by this document.	---
110	MATCHING-PARAMETER	**Case 1:** DIAG-COMM-SNREF refers to a DIAG-SERVICE: The parameter referenced by OUT-PARAM-IF-SNREF shall exist in at least one of the DIAG-SERVICE's POS-RESPONSEs or NEG-RESPONSEs. **Case 2:** DIAG-COMM-SNREF refers to a SINGLE-ECU-JOB: The parameter referenced by OUT-PARAM-IF-SNREF shall exist amongst the SINGLE-ECU-JOB's OUTPUT-PARAMs or NEG-OUTPUT-PARAMs. **WARNING — This rule is to be evaluated on the BASE-VARIANT).**	error
111	Reserved	This range of IDs is reserved by this document.	---
112	DATA-OBJECT-PROP	In case DATA-OBJECT-PROP has a UNIT-REF to a UNIT and UNIT references a FACTOR-SI-TO-UNIT or OFFSET-SI-TO-UNIT, DATA-OBJECT-PROP/PHYSICAL-TYPE/BASE-DATA-TYPE shall be numeric (A_INT32, A_UINT32, A_FLOAT32 or A_FLOAT64).	error
113	PROG-CODE	Within one SINGLE-ECU-JOB and one MULTIPLE-ECU-JOB the contained PROG-CODE elements shall be disjoint regarding their value for the SYNTAX attribute. JAVA, CLASS and JAR exclude each other.	warning
114	DIAG-CODED-TYPE	For DIAG-CODED-TYPE with BASE-DATA-TYPE, BASE-DATA-TYPE is numeric {A_INT32, A_UINT32, A_FLOAT32, A_FLOAT64} only STANDARD-LENGTH-TYPE and PARAM-LENGTH-INFO-TYPE can be used.	error
115	LENGTH-KEY-REF	LENGTH-KEY-REF is only allowed to reference PARAM of type LENGTH-KEY within the same PDU.	error
116	LENGTH-KEY	Within PARAM of type LENGTH-KEY only simple DOPs with PHYSICAL-TYPE/BASE-DATA-TYPE = A_UNIT32 are allowed.	error
117	COMPU-METHOD of type TEXTTABLE	If the DIAG-CODED-TYPE of a DATA-OBJECT-PROP is defined as a string type the definition of a range for COMPU-METHOD/CATEGORY [TEXTTABLE]/COMPU-SCALES/COMPU-SCALE is not allowed: If UPPER-LIMIT is explicitly specified, its content shall be equal to the content of LOWER-LIMIT.	error
118	Reserved	This range of IDs is reserved by this document.	---
119	DYN-DEFINED-SPEC	Within one DYN-DEFINED-SPEC all referenced TABLEs with the same SEMANTIC shall not define or import multiple TABLE-ROWs with the same KEY.	error
120 - 121	Reserved	This range of IDs is reserved by this document.	---
122	INTERNAL-CONSTR SCALE-CONSTR COMPU-SCALE MUX/CASES/CASE PHYS-CONSTR	LOWER-LIMIT/CONTENT shall be less or equal to UPPER-LIMIT/CONTENT. If the CONTENT values are equal the INTERVAL-TYPE of both shall be set to CLOSED.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
123	PARAM of type VALUE	The value of PHYSICAL-DEFAULT-VALUE shall be interpretable with the type of the referenced DATA-OBJECT-PROP/PHYSICAL-DATA-TYPE/BASE-DATA-TYPE and shall be a valid value with respect to DATA-OBJECT-PROP/PHYS-CONSTR.	error
124	PARAM of type VALUE	The corresponding internal value of PHYSICAL-DEFAULT-VALUE shall be within the range of the referenced DATA-OBJECT-PROP/INTERNAL-CONSTR.	error
125	PARAM of type PHYS-CONST	The value of PHYS-CONSTANT-VALUE shall be interpretable with the type of the referenced DATA-OBJECT-PROP/PHYSICAL-DATA-TYPE/BASE-DATA-TYPE and shall be a valid value with respect to DATA-OBJECT-PROP/PHYS-CONSTR.	error
126	PARAM of type PHYS-CONST	The corresponding internal value of PHYS-CONSTANT-VALUE shall be within the range of the referenced DATA-OBJECT-PROP/INTERNAL-CONSTR.	error
127	Reserved	This range of IDs is reserved by this document.	---
128	Param of type SYSTEM	Value of SYSPARAM shall match one of the values in the predefined value set.	warning
129	Reserved	This range of IDs is reserved by this document.	---
130	COMPU-METHOD	LOWER-LIMIT shall be defined in any case of a COMPU-SCALE definition except the COMPU-METHOD is of CATEGORY LINEAR or RAT-FUNC, where both LIMITs can be omitted.	error
131	DDLID-DEF-MODE-INFO	A DDLID-DEF-MODE-INFO shall reference services of DIAGNOSTIC-CLASS CLEAR-DYN-DEF-MESSAGE via odx-link CLEAR-DYN-DEF-MESSAGE-REF, and services of DIAGNOSTIC-CLASS READ-DYN-DEF-MESSAGE via odx-link READ-DYN-DEF-MESSAGE-REF, and services of DIAGNOSTIC-CLASS DYN-DEF-MESSAGE via odx-link DYN-DEF-MESSAGE-REF.	error
132	Reserved	This range of IDs is reserved by this document.	---
133	DIAG-VARIABLE SUB-COMPONENT, POS-RESPONSE-SUPPRESSABLE, ENV-DATA-DESC, COMM-RELATION, STATE-TRANSITION-REF, PRE-CONDITION-STATE-REF, IDENT-DESC, READ-DIAG-COMM-CONNECTOR, WRITE-DIAG-COMM-CONNECTOR, FUNCTION-OUT-PARAM, FUNCTION-IN-PARAM	The referenced IN-PARAM-IF/OUT-PARAM-IF shall not be contained (directly or indirectly) within a FIELD.	error
134	Reserved	This range of IDs is reserved by this document.	---

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
135	DIAG-LAYER	If an element of SHORT-NAME A is declared somewhere in the transitive closure of parent relationships of a DIAG-LAYER D, but no element A is valid in D through inheritance (because it is eliminated somewhere within the inheritance relationships), the SHORT-NAME A cannot be reused in D in the same namespace. If the element A is valid in D through inheritance, its SHORT-NAME can be reused to override the inherited element (except that inherited element has IS-FINAL set to true). If A is not declared anywhere in the transitive closure of parent relationships of a DIAG-LAYER D, A can be freely used in D.	error
136	NOT-INHERITED-DIAG-COMM	If an IS-MANDATORY DIAG-COMM of SHORT-NAME A is inherited into a DIAG-LAYER D, it may not be completely eliminated. In a single-inheritance case it means it cannot be part of any NOT-INHERITED list amongst the transitive closure of inheritance relationships of D. In a multiple-inheritance case, where element with SHORT-NAME A is inherited from multiple sources, it means that exactly one inheritance path shall remain open or as an alternative the element with SHORT-NAME A shall be redefined and thus overridden at layer D. If some of the inherited elements of SHORT-NAME A are not mandatory, the open inheritance path shall inherit one of the mandatory elements into D or the element with SHORT-NAME A might be overridden as well.	error
137	General	A DIAG-LAYER shall not inherit several objects of the same type and with the same SHORT-NAME from multiple parent layers with the same specialization level if the objects are not overridden at that layer. In case an object with the same SHORT-NAME is defined at the DIAG-LAYER ambiguity is resolved and no conflict exists. In case the conflicting objects are of type DIAG-COMM, DOP, TABLE or DIAG-VARIABLE all but one shall be excluded using a NOT-INHERITED clause or as an alternative the object shall be redefined and thus overridden.	error
138	COMPU-METHOD of type TEXTTABLE	If only COMPU-INTERNAL-TO-PHYS is available the COMPU-INVERSE-VALUE of all the COMPU-SCALEs with same physical string value shall be identical.	warning
139	LOGICAL-LINK	A LOGICAL-LINK can either reference a BASE-VARIANT alone or a PROTOCOL alone or a PROTOCOL and a BASE-VARIANT or a PROTOCOL and a FUNCTIONAL-GROUP. All other combinations are invalid.	error
140	COMPU-SCALES	Applicable to a DATA-OBJECT-PROP with STANDARD-LENGTH-TYPE with an ordered BASE-DATA-TYPE and a COMPU-METHOD without COMPU-DEFAULT-VALUE: At least the valid range of a DATA-OBJECT-PROP shall be covered by the COMPU-SCALEs of the COMPU-METHOD. Valid range is defined in section “Simple Data - Data Object Property”.	error
141 - 142	Reserved	This range of IDs is reserved by this document.	---
143	SESSION-DESC	SESSION-SNREF shall reference to a SESSION that exists within the ECU-MEM referenced by the ECU-MEM-CONNECTOR the SESSION-DESC is contained in.	error
144	Reserved	This range of IDs is reserved by this document.	---
145	general	In case if DIAG-CODED-TYPE is A_FLOAT32 the BIT-LENGTH shall be equal to 32. In case of A_FLOAT64, the BIT-LENGTH shall be equal to 64. That means that the DIAG-CODED-TYPE shall be of type STANDARD-LENGTH-TYPE if BASE-DATA-TYPE is set to A_FLOATxx.	error
146 - 148	Reserved	This range of IDs is reserved by this document.	---

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
149	IDENT-DESC	IDENT-IF-SNREF shall reference to a IDENT-IF that exists within the namespace of the ECU-MEM referenced by the ECU-MEM-CONNECTOR the IDENT-DESC is contained in.	error
150	Reserved	This range of IDs is reserved by this document.	---
151	COMPU-METHOD	Each COMPU-SCALE shall not overlap each other COMPU-SCALE inside the same wrapper COMPU-SCALES, except the COMPU-METHOD is of category TEXTTABLE.	error
152	DIAG-SERVICE	If the POS-RESPONSE-SUPPRESSABLE element is defined at a DIAG-SERVICE the parameter referenced by CODED-CONST-SNREF, VALUE-SNREF, TABLE-KEY-SNREF or PHYS-CONST-SNREF shall exist within the request of the DIAG-SERVICE containing the POS-RESPONSE-SUPPRESSABLE element.	error
153 - 155	Reserved	This range of IDs is reserved by this document.	---
156	STRUCTURE	The optional attribute BYTE-SIZE gives the size of the whole structure in bytes. It shall not be less than the total size of the included parameters. If the STRUCTURE has any dynamic components (e.g. MUX or any FIELD with dynamic length) the BYTE-SIZE shall not be specified.	warning
157	odxlink	DOCREF and DOCTYPE shall be specified either jointly or not at all.	error
158	ScopeOutOfXml ODX-CATAGORY	The content of ODX files shall be in accordance to their file extension.	error
159	ScopeOutOfXml PROG-CODE	The referenced file shall exist.	error
160	ScopeOutOfXml DATAFILE	If the attribute LATEBOUND-DATA-FILE equals “false” the referenced file shall exist.	error
161	INPUT-PARAM	The optional PHYSICAL-DEFAULT-VALUE can only exist if the referenced DOP is a DATA-OBJECT-PROP.	error
162	Reserved	This range of IDs is reserved by this document.	---
163	INPUT-PARAM	The corresponding internal value of PHYSICAL-DEFAULT-VALUE shall be within the range of the referenced DATA-OBJECT-PROP/INTERNAL-CONSTR.	error
164	Reserved	This range of IDs is reserved by this document.	---
165	COMPU-METHOD	If the CATEGORY is LINEAR or SCALE-LINEAR the number of V at COMPU-SCALE/COMPU-NUMERATOR shall be 1 or 2 and the number of V at COMPU-DENOMINATOR shall be 0 or 1. If the CATEGORY is RAT-FUNC, SCALE-RAT-FUNC the number of V at COMPU-SCALE/COMPU-NUMERATOR shall be greater than 0. If DENOMINATOR value is missing, it is set to 1 per default.	error
166 - 171	Reserved	This range of IDs is reserved by this document.	---
172	COMPU-METHOD	If no COMPU-SCALE/UPPER-LIMIT is present and COMPU-SCALE/LOWER-LIMIT is present, the INTERVAL-TYPE at LOWER-LIMIT of the same COMPU-SCALE shall be defined as CLOSED.	error
173 - 175	Reserved	This range of IDs is reserved by this document.	---

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
176	DIAG-COMM-DATA-CONNECTOR	The SERVICE-DATA-CONNECTOR/READ-SERVICE-SNREF and SERVICE-DATA-CONNECTOR/WRITE-SERVICE-SNREF shall only reference DIAG-COMMs that reside within Variants referenced by CONFIG-DATA/BASE-VARIANT-DATA for the CONFIG-DATA element in which the SERVICE-DATA-CONNECTOR is contained through a CONFIG-RECORD.	error
177	DIAG-COMM-DATA-CONNECTOR	The IN-PARAM-IF referenced from a READ-PARAM-VALUE or a WRITE-PARAM-VALUE shall be a parameter with a DATA-OBJECT-PROP, except PHYS-CONST and SYSTEM.	error
178	Parameter	A data object of DIAG-CODED-TYPE A_BYTEFIELD, A_ASCIISTRING, A_UTF8STRING, and A_UNICODE2STRING as well as complex data objects shall cover whole bytes, i.e. the bit position shall be zero and the length shall be a multiple of 8 (16 in case of A_UNICODE2STRING). This is valid even if the bit-length of the data object is dynamically defined by MIN-MAX-LENGTH-type, LEADING-LENGTH-INFO or PARAM-LENGTH-INFO-TYPE.	error
179	TABLE	If a TABLE A references a TABLE-ROW from TABLE B, the KEY-DOP-REF of A shall point to the identical DATA-OBJECT-PROP element as the KEY-DOP-REF of B or B has no KEY-DOP-REF at all.	error
180	FUNCTION-DICTIONARY	FUNCTION-DICTIONARY: The BASE-VARIANTs and ECU-VARIANTs referenced by a VEHICLE-INFORMATION-CONNECTOR shall be valid within the VEHICLE-INFORMATION referenced from this VEHICLE-INFORMATION-CONNECTOR, if such a VEHICLE-INFORMATION is referenced.	error
181	FUNCTION-DICTIONARY	BASE-FUNCTION-NODE/FUNCTION-OUT-PARAM/OUT-PARAM-IF shall exist within the POS-RESPONSE PARAMS or OUT-PARAMs of a MULTIPLE-ECU-JOB, DIAG-COMM or DIAG-VARIABLE referenced through BASE-FUNCTION-NODE/VEHICLE-INFORMATION-CONNECTOR/DIAG-LAYER-CONNECTOR/EXECUTABLE-REF. The same applies for IN-PARAM-REF.	error
182	FUNCTION-DICTIONARY	If BASE-FUNCTION-NODE/FUNCTION-OUT-PARAM/OUT-PARAM-IF or BASE-FUNCTION-NODE/FUNCTION-IN-PARAM/IN-PARAM-IF exists, a MULTIPLE-ECU-JOB, DIAG-COMM or DIAG-VARIABLE shall also exist within BASE-FUNCTION-NODE/VEHICLE-INFORMATION-CONNECTOR/DIAG-LAYER-CONNECTOR/EXECUTABLE-REF.	error
183	Reserved	This range of IDs is reserved by this document.	---
184	BIT-MASK	The BIT-MASK length shall match the length of the parameter.	error
185	DTC-DOP	Two DTCs with the same TROUBLE-CODE shall have the same SHORT-NAME and vice versa.	error
186	COMPARAM-REF	If the COMPARAM-REF refers to a COMPARAM, SIMPLE-VALUE shall be specified. Otherwise, COMPLEX-VALUE shall have the same structure as the COMPLEX-COMPARAM it references. COMPLEX-VALUE may only be omitted, if the COMPLEX-COMPARAM is marked with ALLOW-MULTIPLE-VALUES = “true”.	error
187	IDENT-DESC	**Case 1:** DIAG-COMM-SNREF refers to a DIAG-SERVICE: The parameter referenced by OUT-PARAM-IF-SNREF shall exist in at least one of the DIAG-SERVICE's POS-RESPONSEs. **Case 2:** DIAG-COMM-SNREF refers to a SINGLE-ECU-JOB: The parameter referenced by OUT-PARAM-IF-SNREF shall exist amongst the SINGLE-ECU-JOB's OUTPUT-PARAMs.	error
188	SESSION-DESC	SHORT-NAMEs of SESSION-DESCs in all ECU-MEM-CONNECTORs shall be unique in the scope of each supported logical link as given by LAYER-REFs and ALL-VARIANTS-REFs.	error
189	STATE-TRANSITION	Within a STATE-TRANSITION it is only allowed to reference STATEs that are contained in the same STATE-CHART as the STATE-TRANSITION itself.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
190 - 191	Reserved	This range of IDs is reserved by this document.	---
192	DIAG-LAYER	Only one STATE-CHART per SEMANTIC value is allowed within one DIAG-LAYER.	error
193	STATE-TRANSITION-REF PRE-CONDITION-STATE-REF	If IN-PARAM-IF-SNREF is not present or if it refers to a parameter of type CODED-CONST, PHYS-CONST, or TABLE-KEY, VALUE shall not be present. Otherwise, it shall be present and shall be a valid physical value of the parameter regarding the physical and internal constraints.	error
194	STATE-TRANSITION-REF PRE-CONDITION-STATE-REF	The Parameters referenced by STATE-TRANSITION-REF or PRE-CONDITION-STATE-REF through IN-PARAM-IF shall be of type TABLE-KEY, CODED-CONST, VALUE, PHYS-CONST. In case of VALUE, this parameter shall reference a DATA-OBJECT-PROP.	error
195	ECU-GROUP	Either a PHYS-RESOLUTION-LINK or a FUNCT-RESOLUTION-LINK shall exist for every GROUP-MEMBER. The FUNCT-RESOLUTION-LINK points to a LOGICAL-LINK that references a FUNCTIONAL-GROUP and no BASE-VARIANT. The PHYS-RESOLUTION-LINK points to a LOGICAL-LINK that references a BASE-VARIANT.	error
196	Reserved	This range of IDs is reserved by this document.	---
197	VEHICLE-CONNECTOR	Within a VEHICLE-CONNECTOR all PIN-NUMBERs shall be unique.	error
198	FUNCTIONAL-GROUP	Every FUNCTIONAL-GROUP needs to override the value of the COMPLEX-COMPARAM named “CP_UniqueRespIdTable”.	warning
199	COMPLEX-COMPARAM	The ALLOW_MULTIPLE_VALUES attribute of the COMPLEX-COMPARAM CP_UniqueRespIdTable shall be set to “true”.	error
200	Reserved	This range of IDs is reserved by this document.	---
201	COMPLEX-COMPARAM	COMPLEX-COMPARAM “CP_UniqueRespIdTable” always shall contain a COMPARAM with SHORT-NAME CP_ECULayerShortName. The CPUSAGE of this communication parameter shall be set to APPLICATION.	warning
202	COMPLEX-COMPARAM	The PARAM-CLASS of the COMPLEX-COMPARAM “CP_UniqueRespIdTable” and all contained COMPARAMs except for the “CP_ECULayerShortName” shall be set to UNIQUE_ID.	error
203	GROUP-MEMBER	At least one of the two link types (PHYS-RESOLUTION-LINK, FUNCT-RESOLUTION-LINK) shall be defined for every GROUP-MEMBER.	error
204	GROUP-MEMBER	The FUNCT-RESOLUTION-LINK points to a LOGICAL-LINK that references a FUNCTIONAL-GROUP and no BASE-VARIANT.	error
205	GROUP-MEMBER	The PHYS-RESOLUTION-LINK points to a LOGICAL-LINK that references a BASE-VARIANT only, a PROTOCOL only or a BASE-VARIANT and a PROTOCOL.	error
206	BASE-VARIANT-PATTERN	If the BASE-VARIANT-PATTERN of BASE-VARIANT contains at least one MATCHING-PARAMETER with USE-PHYSICAL-ADDRESSING = false, the BASE-VARIANT shall have a FUNCTIONAL-RESOLUTION-LINK specified in every ECU-GROUP it appears in.	warning
207	BASE-VARIANT-PATTERN	If the BASE-VARIANT-PATTERN of BASE-VARIANT has at least one MATCHING-PARAMETER with USE-PHYSICAL-ADDRESSING = true, the BASE-VARIANT shall have a PHYS-RESOLUTION-LINK specified in every ECU-GROUP it appears in.	warning

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
208	PARAM	If a parameter reaches the end of the PDU by definition (either END-OF-PDU-FIELD or DATA-OBJECT-PROP with TERMINATION “END-OF-PDU”) or if a parameter is a complex DOP containing such a parameter at any position, the next parameter (in the sense of the order in the XML file) shall explicitly specify the BYTE-POSITION. ESG, feb08: BASE-VARIANT-PATTERN does not refer to MATCHING-BASE-VARIANT-PARAMETER but contains. Therefore “references” is replaced by “has”.	error
209	LOGICAL-LINK	A LOGICAL-LINK L shall reference a PROT-STACK (either directly or indirectly through the associated PROTOCOL) whose PHYSICAL-LINK-TYPE is equal to the value of TYPE at the PHYSICAL-VEHICLE-LINK referenced by L.	error
210	GROUP-MEMBER	The BASE-VARIANT referenced by GROUP-MEMBER shall be equal to the BASE-VARIANT referenced through PHYS-RESOLUTION-LINK/LOGICAL-LINK of this same GROUP-MEMBER, if this LOGICAL-LINK references a BASE-VARIANT.	warning
211	INPUT-PARAM	The physical value of PHYSICAL-DEFAULT-VALUE shall be within the range of the referenced DATA-OBJECT-PROP/PHYS-CONSTR.	error
212	VEHICLE-CONNECTOR	Every VEHICLE-CONNECTOR shall have at most one VEHICLE-CONNECTOR-PIN with PIN-TYPE “IGNITION-CLAMP”.	error
213	DIAG-CODED-TYPE	If CONDENSED=true a BIT-MASK shall be set.	error
214	global	The content of LOWER-LIMIT, UPPER-LIMIT, V, VT, PHYSICAL-DEFAULT-VALUE, PHYS-CONSTANT-VALUE, CODED-VALUE, KEY, VALUE, EXPECTED-VALUE, TERMINATION-VALUE (of type xsd:string) shall match the corresponding datatype: — A_INT32 : xsd:int — A_UINT32 : xsd:unsignedInt — A_FLOAT32 : xsd:float — A_FLOAT64 : xsd:double — A_BYTEFIELD : xsd:hexBinary — else xsd:string It shall be ensured even when the DOP is overridden (e.g. a PHYS-CONST with a DOP-SNREF)	error
215	TABLE	A TABLE that corresponds to a TABLE-KEY shall have KEY-DOP.	error
216	TABLE	A TABLE that corresponds to a TABLE-ENTRY with a ROW-FRAGMENT value of KEY, shall have a KEY-DOP.	error
217	ENV-DATA-DESC	The SHORT-NAME of ENV-DATAs shall be unique within the ENV-DATA-DESC.	error
218 - 219	Reserved	This range of IDs is reserved by this document.	---
220	global	SHORT-NAMEs of CONFIG-DATA in all ECU-CONFIGs shall be unique in the scope of each supported logical link as given by BASE-VARIANT-SNREFs and ECU-VARIANT-SNREFs.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
221	FUNCTION-DICTIONARY	The DIAG-COMM referenced from FUNCTION-(IN/OUT)-PARAM shall also be referenced from one of the DIAG-OBJECT-CONNECTORs.	error
223	DATABLOCK	The attribute USER-SELECTION at DATAFORMAT shall be present, if and only if the attribute SELECTION at DATAFORMAT element is set to USER-DEFINED.	error
224	global	All SNREF references shall be valid and may only reference the target as specified by the ODX UML model. For each SNREF the context is considered by this rule. In case if the context is a DIAG-LAYER, the reference shall be valid in each layer that inherits the referencing object.	error
225	BASE-DATA-TYPE	Depending on the BASE-DATA-TYPE only the BASE-TYPE-ENCODING values are allowed as listed in the ODX specification, Table “BASE-DATA-TYPE encodings”.	error
226	COMPU-METHOD of type TEXTTABLE	At COMPU-METHODs of type TEXTTABLE the following restrictions exist: COMPU-INTERNAL-TO-PHYS direction: — COMPU-CONST/VT shall be defined for every COMPU-SCALE. Use of COMPU-CONST/V is not allowed. — If COMPU-DEFAULT-VALUE is present, it shall contain VT. Use of V is not allowed at this object. — COMPU-PHYS-TO-INTERNAL direction: — If a COMPU-SCALE defines a COMPU-INVERSE-VALUE, it shall contain VT. Use of V is not allowed at this object. — If COMPU-DEFAULT-VALUE/COMPU-INVERSE-VALUE is present, it shall contain VT. Use of V is not allowed at this object.	error
227	global	OIDs shall be globally unique.	error
228	COMPU-METHOD of type LINEAR or RAT-FUNC	The type of (scale) linear and rational calculation depends on both physical and coded data type. If at least one of them is A_FLOAT32 or A_FLOAT64, the calculation type is float, else if both are A_UINT32, it is an unsigned integer, else it is signed integer. The coefficients in V inside COMPU-NUMERATOR and COMPU-DENOMINATOR shall match the Schema datatypes xsd:double, xsd:unsignedInt, and xsd:int, respectively. Exception: In the case of (scale) linear calculation, the content of V inside COMPU-DENOMINATOR shall match the Schema datatype xsd:unsignedInt regardless of the datatypes.	error
229	global	Memory addresses of type hexBinary shall not be longer than 4 bytes.	warning
230	CONFIG-DATA	The CONFIG-DATA may be valid for any number of base variants. Related to a single base variant, the following four cases are allowed: a) Only one VALID-BASE-VARIANT refers to this base variant, BASE-VARIANT-SNREF without any ECU-VARIANT-SNREFs. As consequence CONFIG-DATA is valid for the base variant and all variants inherited from this. b) Only one VALID-BASE-VARIANT refers to this base variant, BASE-VARIANT-SNREF with at least one ECU-VARIANT-SNREF. As consequence CONFIG-DATA is valid only for the variants explicitly specified, but not for the base variant itself. c) Two VALID-BASE-VARIANTs refer to this base variant, one of them contains BASE-VARIANT-SNREF with at least one ECU-VARIANT-SNREF, the other one without any ECU-VARIANT-SNREFs. As consequence CONFIG-DATA is valid only for the variants explicitly specified and for the base variant self. d) Two VALID-BASE-VARIANTs refer to this base variant, but none of them contains an ECU-VARIANT-SNREF. As a consequence the CONFIG-DATA is valid only for this BASE-VARIANT and for no ECU-VARIANTS at all.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
231	CONFIG-RECORD	No bit shall be occupied inside the CONFIG-RECORD by more than one CONFIG-ITEM according to its BYTE-POSITION, BIT-POSITION, byte order, bit mask, and the length of the parameter.	error
232	WRITE-DIAG-COMM-CONNECTOR	The parameter referenced by WRITE-DATA-SNREF or WRITE-DATA-SNPATHREF inside the REQUEST of the write service referenced by WRITE-DIAG-COMM-REF or WRITE-DIAG-COMM-SNREF shall be of type VALUE, and shall use a DATA-OBJECT-PROP with the physical datatype A_BYTEFIELD. The length of this parameter shall be either dynamical or equal to UNCOMPRESSED-SIZE of the nesting DIAG-COMM-DATA-CONNECTOR.	error
233	READ-DIAG-COMM-CONNECTOR	The parameter referenced by READ-DATA-SNREF or READ-DATA-SNPATHREF inside all POS-RESPONSEs of the read service referenced by READ-DIAG-COMM-REF or READ-DIAG-COMM-SNREF shall be of type VALUE, and shall use a DATA-OBJECT-PROP with the physical datatype A_BYTEFIELD. The length of this parameter shall be either dynamical or equal to UNCOMPRESSED-SIZE of the nesting DIAG-COMM-DATA-CONNECTOR.	error
234	SUB-COMPONENT	Only DIAG-LAYERS of type BASE-VARIANT and ECU-VARIANT can specify objects of type SUB-COMPONENT	error
235	Reserved	This range of IDs is reserved by this document.	---
236	COMPU-METHOD of type TEXTTABLE	At COMPU-METHODs of type TEXTTABLE the following restrictions exist: COMPU-INTERNAL-TO-PHYS direction: — COMPU-CONST/VT shall be defined for every COMPU-SCALE. Use of COMPU-SCALE/V is not allowed. — if COMPU-DEFAULT-VALUE is present, it shall contain VT. Use of V is not allowed at this object.	error
237	COMPU-METHOD of type TEXTTABLE	At COMPU-METHODs of type TEXTTABLE with the internal datatype A_BYTEFIELD, A_ASCIISTRING, A_UTF8STRING, or A_UNICODE2STRING the following restrictions exist: a) COMPU-INTERNAL-TO-PHYS direction: — If a COMPU-SCALE defines a COMPU-INVERSE-VALUE, it shall contain VT. Use of V is not allowed at this object. — If COMPU-DEFAULT-VALUE/COMPU-INVERSE-VALUE is present, it shall contain VT. Use of V is not allowed at this object. b) COMPU-PHYS-TO-INTERNAL direction: — COMPU-CONST/VT shall be defined for every COMPU-SCALE. Use of COMPU-CONST/V is not allowed. — If COMPU-DEFAULT-VALUE is present, it shall contain VT. Use of V is not allowed at this object.	error

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
238	COMPU-METHOD of type LINEAR, SCALE-LINEAR, RAT-FUNCT, SCALE-RAT-FUNCT	At COMPU-METHODs of type LINEAR, SCALE-LINEAR, RAT-FUNCT, and SCALE-RAT-FUNCT the following restrictions exist: a) COMPU-INTERNAL-TO-PHYS direction: — If a COMPU-SCALE defines a COMPU-INVERSE-VALUE, it shall contain V. Use of VT is not allowed at this object. — If COMPU-DEFAULT-VALUE/COMPU-INVERSE-VALUE is present, it shall contain V. Use of VT is not allowed at this object. — COMPU-RATIONAL-COEFFS shall be present. b) COMPU-PHYS-TO-INTERNAL direction: — If COMPU-DEFAULT-VALUE is present, it shall contain V. Use of VT is not allowed at this object. — COMPU-RATIONAL-COEFFS shall be present.	error
239	COMPU-METHOD of type TAB-INTP	At COMPU-METHODs of type TAB-INTP the following restrictions exist: — COMPU-CONST/V shall be defined for every COMPU-SCALE. — Use of COMPU-CONST/VT or COMPU-RATIONAL-COEFFS is not allowed. — Use of COMPU-INVERSE-VALUE is not allowed.	error
240	COMPARAM-REF	If a COMPLEX-COMPARAM defines ALLOW-MULTIPLE-VALUE="true", it is not allowed to reference a COMPARAM of COMPLEX-COMPARAM that is inside this.	error
241	COMPARAM-REF	A COMPLEX-COMPARAM shall not include a COMPLEX-PHYSICAL-DEFAULT-VALUE if ALLOW-MULTIPLE-VALUES="false".	error
242	COMPARAM-REF	Only COMPARAM-REFs referencing a COMPLEX-COMPARAM with attribute ALLOW-MULTIPLE-VALUES="true" may be empty.	error
243	TABLE	If a TABLE defines a TABLE-ROW with IS-FINAL=true and this table is overridden in a lower layer, the new TABLE shall not define a TABLE-ROW with the same KEY, but it can refer to the original TABLE-ROW. Objects referenced by TABLE-ROWs are ignored.	error
244	TABLE	If a TABLE defines a TABLE-ROW with IS-MANDATORY=true and this table is overridden in a lower layer, the new TABLE shall contain a TABLE-ROW with the same KEY (no matter whether it is a TABLE-ROW-REF or a new TABLE-ROW). Objects referenced by TABLE-ROWs are ignored.	error
245	DIAG-VARIABLE/COMM-RELATION DIAG-COMM with DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION	All REQUEST-PARAMs of the referenced DIAG-COMM which are not referenced by IN-PARAM-IF shall fulfil the following restrictions: a) PARAMs of type VALUE within the whole transitive closure which refer to DATA-OBJECT-PROP or DTC-DOP shall have PHYSICAL-DEFAULT-VALUE set. b) PARAMs of type VALUE shall not reference an END-OF-PDU-FIELD. c) PARAMs of type TABLE-KEY shall reference a TABLE-ROW. d) PARAMs of type LENGTH-KEY are NOT allowed.	error
246	DIAG-VARIABLE/COMM-RELATION	No REQUEST-PARAMs of the referenced DIAG-COMM shall be of type SYSTEM.	warning

Table B.1 (*continued*)

ID	Location	Detailed description	Error Level
247	DIAG-SERVICE	A DIAG-SERVICE having DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION shall not have the flag IS-CYCLIC=true.	error
248	DIAG-COMM with DIAGNOSTIC-CLASS =VARIANTIDENTIFICATION	A DIAG-COMM with DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION shall neither be defined nor overridden on ECU-VARIANT level.	error
249	DIAG-COMM with DIAGNOSTIC-CLASS =VARIANTIDENTIFICATION	Any element of the transitive closure of a DIAG-COMM's REQUEST, POS-RESPONSE, NEG-RESPONSE with DIAGNOSTIC-CLASS=VARIANTIDENTIFICATION shall not be overridden in ECU-VARIANTs.	error
250	DIAG-COMM with DIAGNOSTIC-CLASS =VARIANTIDENTIFICATION	The SHORT-NAME of the response parameter used for variant identification shall not be part of any valid GLOBAL-NEG-RESPONSEs in the BASE-VARIANT and all ECU-VARIANTs.	error
251	DIAG-LAYER of type ECU-VARIANT	It is forbidden to overwrite a COMPARAM with PARAM-CLASS=INIT on ECU-VARIANT layer.	error
252 - 253	Reserved	This range of IDs is reserved by this document.	---
254	PARAM	A PARAM which is referenced by another PARAM (e.g. LENGTH-KEY and TABLE-KEY) shall be specified before the referencing PARAM (in order of appearance in the XML document) or shall specify BYTE-POSITION.	error
255	DATA-OBJECT-PROP	If PHYS-CONSTR is present within element DATA-OBJECT-PROP, PHYSICAL-TYPE/BASE-DATA-TYPE shall be a ordered data type (A_BYTEFIELD, A_INT32, A_UINT32, A_FLOAT32 or A_FLOAT64).	error
256	ECU-MEM-CONNECTOR	ALL-VARIANT-REF shall refer to BASE-VARIANT.	error
257	INTERNAL-CONSTR	No SCALE-CONSTR shall cover a range outside of INTERNAL-CONSTR/LOWER-LIMIT and /UPPER-LIMIT.	warning
258	PHYS-CONSTR	No SCALE-CONSTR shall cover a range outside of PHYS-CONSTR/LOWER-LIMIT and /UPPER-LIMIT.	warning
259 - 260	Reserved	This range of IDs is reserved by this document.	---
261	DIAG-SERVICE	The SHORT-NAMEs of all POS-RESPONSEs and NEG-RESPONSEs referenced by one DIAG-SERVICE shall be unique and shall differ from the SHORT-NAMEs of all GLOBAL-NEG-RESPONSEs of any inherited layer.	warning
262 and on-ward	Reserved	This range of IDs is reserved by this document.	---

Annex C
(normative)

XML schema

C.1 XML schema of ODX (odx.xsd)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--Version: 2.2.0-->
<!--Generated: Thu Feb 28 22:29:06 CET 2008-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:include schemaLocation="odx-xhtml.xsd"/>
    <xsd:complexType abstract="false" name="ADDITIONAL-AUDIENCE">
        <!--Class: ADDITIONAL-AUDIENCE-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType name="ADDITIONAL-AUDIENCES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="ADDITIONAL-AUDIENCE" name="ADDITIONAL-AUDIENCE"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ADDRDEF-FILTER">
        <!--Class: ADDRDEF-FILTER-->
        <xsd:complexContent>
            <xsd:extension base="FILTER">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="FILTER-END"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ADDRDEF-PHYS-SEGMENT">
        <!--Class: ADDRDEF-PHYS-SEGMENT-->
        <xsd:complexContent>
            <xsd:extension base="PHYS-SEGMENT">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="END-ADDRESS"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="ADDRESSING">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FUNCTIONAL"/>
            <xsd:enumeration value="PHYSICAL"/>
            <xsd:enumeration value="FUNCTIONAL-OR-PHYSICAL"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType abstract="false" name="ADMIN-DATA">
```

```
        <!--Class: ADMIN-DATA-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" name="LANGUAGE">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:language">
                        <xsd:pattern value="[a-z]{2}(-[A-Z]{2})?"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element type="COMPANY-DOC-INFOS" name="COMPANY-DOC-INFOS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="DOC-REVISIONS" name="DOC-REVISIONS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ALL-VALUE">
        <!--Class: ALL-VALUE-->
    </xsd:complexType>
    <xsd:complexType name="ALL-VARIANT-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="ALL-VARIANT-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="AUDIENCE">
        <!--Class: AUDIENCE-->
        <xsd:choice>
            <xsd:element type="ENABLED-AUDIENCE-REFS" name="ENABLED-AUDIENCE-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="DISABLED-AUDIENCE-REFS" name="DISABLED-AUDIENCE-REFS" minOccurs="0" maxOccurs="1"/>
        </xsd:choice>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-SUPPLIER"/>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-DEVELOPMENT"/>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-MANUFACTURING"/>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-AFTERSALES"/>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-AFTERMARKET"/>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="BASE-COMPARAM">
        <!--Class: BASE-COMPARAM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
        <xsd:attribute use="required" type="xsd:string" name="PARAM-CLASS"/>
        <xsd:attribute use="required" type="STANDARDISATION-LEVEL" name="CPTYPE"/>
        <xsd:attribute use="optional" type="xsd:unsignedInt" name="DISPLAY-LEVEL"/>
        <xsd:attribute use="required" type="USAGE" name="CPUSAGE"/>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="BASE-FUNCTION-NODE">
        <!--Class: BASE-FUNCTION-NODE-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="AUDIENCE"/>
            <xsd:element type="FUNCTION-IN-PARAMS" name="FUNCTION-IN-PARAMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="FUNCTION-OUT-PARAMS" name="FUNCTION-OUT-PARAMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="COMPONENT-CONNECTORS" name="COMPONENT-CONNECTORS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="MULTIPLE-ECU-JOB-REFS" name="MULTIPLE-ECU-JOB-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="SDG" name="SDG"/>
        </xsd:sequence>
```

```
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="BASE-VALUE">
        <!--Class: BASE-VALUE-->
    </xsd:complexType>
    <xsd:complexType abstract="false" name="BASE-VARIANT">
        <!--Class: BASE-VARIANT-->
        <xsd:complexContent>
            <xsd:extension base="HIERARCHY-ELEMENT">
                <xsd:sequence>
                    <xsd:element type="DIAG-VARIABLES" name="DIAG-VARIABLES" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="VARIABLE-GROUPS" name="VARIABLE-GROUPS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="DYN-DEFINED-SPEC" name="DYN-DEFINED-SPEC"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="BASE-VARIANT-PATTERN" name="BASE-VARIANT-
PATTERN"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="PARENT-REFS" name="PARENT-REFS"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="BASE-VARIANT-PATTERN">
        <!--Class: BASE-VARIANT-PATTERN-->
        <xsd:sequence>
            <xsd:element type="MATCHING-BASE-VARIANT-PARAMETERS" name="MATCHING-BASE-VARIANT-PARAMETERS"
minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="BASE-VARIANT-REF">
        <!--Class: BASE-VARIANT-REF-->
        <xsd:complexContent>
            <xsd:extension base="PARENT-REF"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="BASE-VARIANTS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="BASE-VARIANT" name="BASE-VARIANT"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="BASIC-STRUCTURE">
        <!--Class: BASIC-STRUCTURE-->
        <xsd:complexContent>
            <xsd:extension base="COMPLEX-DOP">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="BYTE-SIZE"/>
                    <xsd:element type="PARAMS" name="PARAMS" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="CASE">
        <!--Class: CASE-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:choice maxOccurs="1" minOccurs="0">
                <xsd:element name="STRUCTURE-REF" type="ODXLINK"/>
                <xsd:element name="STRUCTURE-SNREF" type="SNREF"/>
```

```
        </xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="1" type="LIMIT" name="LOWER-LIMIT"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="LIMIT" name="UPPER-LIMIT"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CASES">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="CASE" name="CASE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="CHECKSUM">
    <!--Class: CHECKSUM-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:hexBinary" name="FILLBYTE"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="SOURCE-START-ADDRESS"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="COMPRESSED-SIZE"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="CHECKSUM-ALG"/>
        <xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="1" type="SOURCE-END-ADDRESS" name="SOURCE-END-ADDRESS"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="UNCOMPRESSED-SIZE" name="UNCOMPRESSED-SIZE"/>
        </xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="1" type="CHECKSUM-RESULT" name="CHECKSUM-RESULT"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType abstract="false" name="CHECKSUM-RESULT">
    <!--Class: CHECKSUM-RESULT-->
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute use="required" type="SESSION-SUB-ELEM-TYPE" name="TYPE"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="CHECKSUMS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="CHECKSUM" name="CHECKSUM"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="CODED-CONST">
    <!--Class: CODED-CONST-->
    <xsd:complexContent>
        <xsd:extension base="POSITIONABLE-PARAM">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="CODED-VALUE"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="DIAG-CODED-TYPE" name="DIAG-CODED-TYPE"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="CODED-VALUES">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="xsd:string" name="CODED-VALUE"/>
    </xsd:sequence>
```

```
</xsd:complexType>
<xsd:complexType abstract="false" name="COMM-RELATION">
    <!--Class: COMM-RELATION-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" type="DESCRIPTION" name="DESC"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="RELATION-TYPE"/>
        <xsd:choice maxOccurs="1" minOccurs="1">
            <xsd:element name="DIAG-COMM-REF" type="ODXLINK"/>
            <xsd:element name="DIAG-COMM-SNREF" type="SNREF"/>
        </xsd:choice>
        <xsd:choice>
            <xsd:choice maxOccurs="1" minOccurs="0">
                <xsd:element name="IN-PARAM-IF-SNREF" type="SNREF"/>
                <xsd:element name="IN-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
            </xsd:choice>
            <xsd:choice maxOccurs="1" minOccurs="0">
                <xsd:element name="OUT-PARAM-IF-SNREF" type="SNREF"/>
                <xsd:element name="OUT-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
            </xsd:choice>
        </xsd:choice>
    </xsd:sequence>
    <xsd:attribute use="optional" type="COMM-RELATION-VALUE-TYPE" name="VALUE-TYPE"/>
</xsd:complexType>
<xsd:complexType name="COMM-RELATIONS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMM-RELATION" name="COMM-RELATION"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="COMM-RELATION-VALUE-TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="CURRENT"/>
        <xsd:enumeration value="STORED"/>
        <xsd:enumeration value="STATIC"/>
        <xsd:enumeration value="SUBSTITUTED"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="false" name="COMPANY-DATA">
    <!--Class: COMPANY-DATA-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element type="ROLES" name="ROLES" minOccurs="0" maxOccurs="1"/>
        <xsd:element type="TEAM-MEMBERS" name="TEAM-MEMBERS" minOccurs="0" maxOccurs="1"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="COMPANY-SPECIFIC-INFO" name="COMPANY-SPECIFIC-INFO"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType name="COMPANY-DATAS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPANY-DATA" name="COMPANY-DATA"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPANY-DOC-INFO">
    <!--Class: COMPANY-DOC-INFO-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="COMPANY-DATA-REF" type="ODXLINK"/>
```

```
            <xsd:element maxOccurs="1" minOccurs="0" name="TEAM-MEMBER-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="DOC-LABEL"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="COMPANY-DOC-INFOS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPANY-DOC-INFO" name="COMPANY-DOC-INFO"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="COMPANY-REVISION-INFO">
        <!--Class: COMPANY-REVISION-INFO-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="COMPANY-DATA-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="REVISION-LABEL"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="STATE"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="COMPANY-REVISION-INFOS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPANY-REVISION-INFO" name="COMPANY-REVISION-
INFO"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="COMPANY-SPECIFIC-INFO">
        <!--Class: COMPANY-SPECIFIC-INFO-->
        <xsd:sequence>
            <xsd:element type="RELATED-DOCS" name="RELATED-DOCS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="COMPARAM">
        <!--Class: COMPARAM-->
        <xsd:complexContent>
            <xsd:extension base="BASE-COMPARAM">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="PHYSICAL-DEFAULT-VALUE"/>
                    <xsd:element maxOccurs="1" minOccurs="1" name="DATA-OBJECT-PROP-REF" type="ODXLINK"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="COMPARAM-REF">
        <!--Class: COMPARAM-REF-->
        <xsd:sequence>
            <xsd:choice>
                <xsd:element maxOccurs="1" minOccurs="0" type="SIMPLE-VALUE" name="SIMPLE-VALUE"/>
                <xsd:element maxOccurs="1" minOccurs="0" type="COMPLEX-VALUE" name="COMPLEX-VALUE"/>
            </xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="0" type="DESCRIPTION" name="DESC"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="PROTOCOL-SNREF" type="SNREF"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="PROT-STACK-SNREF" type="SNREF"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="ODXLINK-ATTR"/>
    </xsd:complexType>
    <xsd:complexType name="COMPARAM-REFS">
```

```
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPARAM-REF" name="COMPARAM-REF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="COMPARAMS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPARAM" name="COMPARAM"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="COMPARAM-SPEC">
        <!--Class: COMPARAM-SPEC-->
        <xsd:complexContent>
            <xsd:extension base="ODX-CATEGORY">
                <xsd:sequence>
                    <xsd:element type="PROT-STACKS" name="PROT-STACKS" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="COMPARAM-SUBSET">
        <!--Class: COMPARAM-SUBSET-->
        <xsd:complexContent>
            <xsd:extension base="ODX-CATEGORY">
                <xsd:sequence>
                    <xsd:element type="COMPARAMS" name="COMPARAMS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="COMPLEX-COMPARAMS" name="COMPLEX-COMPARAMS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="DATA-OBJECT-PROPS" name="DATA-OBJECT-PROPS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="UNIT-SPEC" name="UNIT-SPEC"/>
                </xsd:sequence>
                <xsd:attribute use="required" type="xsd:string" name="CATEGORY"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="COMPARAM-SUBSET-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="COMPARAM-SUBSET-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="COMPLEX-COMPARAM">
        <!--Class: COMPLEX-COMPARAM-->
        <xsd:complexContent>
            <xsd:extension base="BASE-COMPARAM">
                <xsd:sequence>
                    <xsd:choice maxOccurs="unbounded" minOccurs="1">
                        <xsd:element maxOccurs="1" minOccurs="1" type="COMPARAM" name="COMPARAM"/>
                        <xsd:element maxOccurs="1" minOccurs="1" type="COMPLEX-COMPARAM" name="COMPLEX-COMPARAM"/>
                    </xsd:choice>
                    <xsd:element maxOccurs="1" minOccurs="0" type="COMPLEX-PHYSICAL-DEFAULT-VALUE" name="COMPLEX-
PHYSICAL-DEFAULT-VALUE"/>
                </xsd:sequence>
                <xsd:attribute default="false" use="optional" type="xsd:boolean" name="ALLOW-MULTIPLE-VALUES"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="COMPLEX-COMPARAMS">
```

```
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPLEX-COMPARAM" name="COMPLEX-COMPARAM"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="true" name="COMPLEX-DOP">
    <!--Class: COMPLEX-DOP-->
    <xsd:complexContent>
        <xsd:extension base="DOP-BASE"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPLEX-PHYSICAL-DEFAULT-VALUE">
    <!--Class: COMPLEX-PHYSICAL-DEFAULT-VALUE-->
    <xsd:sequence>
        <xsd:element type="COMPLEX-VALUES" name="COMPLEX-VALUES" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPLEX-VALUE">
    <!--Class: COMPLEX-VALUE-->
    <xsd:complexContent>
        <xsd:extension base="BASE-VALUE">
            <xsd:choice maxOccurs="unbounded" minOccurs="1">
                <xsd:element maxOccurs="1" minOccurs="1" type="SIMPLE-VALUE" name="SIMPLE-VALUE"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="COMPLEX-VALUE" name="COMPLEX-VALUE"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="COMPLEX-VALUES">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPLEX-VALUE" name="COMPLEX-VALUE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPONENT-CONNECTOR">
    <!--Class: COMPONENT-CONNECTOR-->
    <xsd:sequence>
        <xsd:element type="ECU-VARIANT-REFS" name="ECU-VARIANT-REFS" minOccurs="0" maxOccurs="1"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="BASE-VARIANT-REF" type="ODXLINK"/>
        <xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="0" type="DIAG-OBJECT-CONNECTOR" name="DIAG-OBJECT-CONNECTOR"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="DIAG-OBJECT-CONNECTOR-REF" type="ODXLINK"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="COMPONENT-CONNECTORS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPONENT-CONNECTOR" name="COMPONENT-CONNECTOR"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="COMPU-CATEGORY">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="IDENTICAL"/>
        <xsd:enumeration value="LINEAR"/>
        <xsd:enumeration value="SCALE-LINEAR"/>
        <xsd:enumeration value="TEXTTABLE"/>
        <xsd:enumeration value="COMPUCODE"/>
```

```
        <xsd:enumeration value="TAB-INTP"/>
        <xsd:enumeration value="RAT-FUNC"/>
        <xsd:enumeration value="SCALE-RAT-FUNC"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="false" name="COMPU-CONST">
    <!--Class: COMPU-CONST-->
    <xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="1" type="V" name="V"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="VT" name="VT"/>
    </xsd:choice>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPU-DEFAULT-VALUE">
    <!--Class: COMPU-DEFAULT-VALUE-->
    <xsd:sequence>
        <xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="1" type="V" name="V"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="VT" name="VT"/>
        </xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="0" type="COMPU-INVERSE-VALUE" name="COMPU-INVERSE-VALUE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="COMPU-DENOMINATOR">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="V" name="V"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPU-INTERNAL-TO-PHYS">
    <!--Class: COMPU-INTERNAL-TO-PHYS-->
    <xsd:sequence>
        <xsd:choice>
            <xsd:element type="COMPU-SCALES" name="COMPU-SCALES" minOccurs="1" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="PROG-CODE" name="PROG-CODE"/>
        </xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="0" type="COMPU-DEFAULT-VALUE" name="COMPU-DEFAULT-VALUE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPU-INVERSE-VALUE">
    <!--Class: COMPU-INVERSE-VALUE-->
    <xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="1" type="V" name="V"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="VT" name="VT"/>
    </xsd:choice>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPU-METHOD">
    <!--Class: COMPU-METHOD-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" type="COMPU-CATEGORY" name="CATEGORY"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="COMPU-INTERNAL-TO-PHYS" name="COMPU-INTERNAL-TO-PHYS"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="COMPU-PHYS-TO-INTERNAL" name="COMPU-PHYS-TO-INTERNAL"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="COMPU-NUMERATOR">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="V" name="V"/>
    </xsd:sequence>
```

```
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPU-PHYS-TO-INTERNAL">
    <!--Class: COMPU-PHYS-TO-INTERNAL-->
    <xsd:sequence>
        <xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="1" type="PROG-CODE" name="PROG-CODE"/>
            <xsd:element type="COMPU-SCALES" name="COMPU-SCALES" minOccurs="1" maxOccurs="1"/>
        </xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="0" type="COMPU-DEFAULT-VALUE" name="COMPU-DEFAULT-VALUE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPU-RATIONAL-COEFFS">
    <!--Class: COMPU-RATIONAL-COEFFS-->
    <xsd:sequence>
        <xsd:element type="COMPU-NUMERATOR" name="COMPU-NUMERATOR" minOccurs="1" maxOccurs="1"/>
        <xsd:element type="COMPU-DENOMINATOR" name="COMPU-DENOMINATOR" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="COMPU-SCALE">
    <!--Class: COMPU-SCALE-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" type="TEXT" name="SHORT-LABEL"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="DESCRIPTION" name="DESC"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="LIMIT" name="LOWER-LIMIT"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="LIMIT" name="UPPER-LIMIT"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="COMPU-INVERSE-VALUE" name="COMPU-INVERSE-VALUE"/>
        <xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="1" type="COMPU-CONST" name="COMPU-CONST"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="COMPU-RATIONAL-COEFFS" name="COMPU-RATIONAL-COEFFS"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="COMPU-SCALES">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="COMPU-SCALE" name="COMPU-SCALE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="CONFIG-DATA">
    <!--Class: CONFIG-DATA-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element type="VALID-BASE-VARIANTS" name="VALID-BASE-VARIANTS" minOccurs="1" maxOccurs="1"/>
        <xsd:element type="CONFIG-RECORDS" name="CONFIG-RECORDS" minOccurs="0" maxOccurs="1"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="CONFIG-DATA-DICTIONARY-SPEC">
    <!--Class: CONFIG-DATA-DICTIONARY-SPEC-->
    <xsd:sequence>
        <xsd:element type="DATA-OBJECT-PROPS" name="DATA-OBJECT-PROPS" minOccurs="0" maxOccurs="1"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="UNIT-SPEC" name="UNIT-SPEC"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CONFIG-DATAS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="CONFIG-DATA" name="CONFIG-DATA"/>
```

```
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="CONFIG-ID-ITEM">
        <!--Class: CONFIG-ID-ITEM-->
        <xsd:complexContent>
            <xsd:extension base="CONFIG-ITEM"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="CONFIG-ITEM">
        <!--Class: CONFIG-ITEM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="BYTE-POSITION"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="BIT-POSITION"/>
            <xsd:choice>
                <xsd:element maxOccurs="1" minOccurs="1" name="DATA-OBJECT-PROP-REF" type="ODXLINK"/>
                <xsd:element maxOccurs="1" minOccurs="1" name="DATA-OBJECT-PROP-SNREF" type="SNREF"/>
            </xsd:choice>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="optional" type="xsd:string" name="SEMANTIC"/>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="CONFIG-RECORD">
        <!--Class: CONFIG-RECORD-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="CONFIG-ID-ITEM" name="CONFIG-ID-ITEM"/>
            <xsd:element type="DIAG-COMM-DATA-CONNECTORS" name="DIAG-COMM-DATA-CONNECTORS" minOccurs="1"
maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="IDENT-VALUE" name="CONFIG-ID"/>
            <xsd:element type="DATA-RECORDS" name="DATA-RECORDS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="AUDIENCE"/>
            <xsd:element type="SYSTEM-ITEMS" name="SYSTEM-ITEMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="DATA-ID-ITEM" name="DATA-ID-ITEM"/>
            <xsd:element type="OPTION-ITEMS" name="OPTION-ITEMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="DEFAULT-DATA-RECORD-SNREF" type="SNREF"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CONFIG-RECORDS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="CONFIG-RECORD" name="CONFIG-RECORD"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DATA">
        <!--Class: DATA-->
        <xsd:simpleContent>
            <xsd:extension base="xsd:string"/>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DATABLOCK">
        <!--Class: DATABLOCK-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:hexBinary" name="LOGICAL-BLOCK-INDEX"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="FLASHDATA-REF" type="ODXLINK"/>
            <xsd:element type="FILTERS" name="FILTERS" minOccurs="0" maxOccurs="1"/>
```

```
        <xsd:element type="SEGMENTS" name="SEGMENTS" minOccurs="0" maxOccurs="1"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="TARGET-ADDR-OFFSET" name="TARGET-ADDR-OFFSET"/>
        <xsd:element type="OWN-IDENTS" name="OWN-IDENTS" minOccurs="0" maxOccurs="1"/>
        <xsd:element type="SECURITYS" name="SECURITYS" minOccurs="0" maxOccurs="1"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="AUDIENCE"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    <xsd:attribute use="required" type="xsd:string" name="TYPE"/>
</xsd:complexType>
<xsd:complexType name="DATABLOCK-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="DATABLOCK-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DATABLOCKS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="DATABLOCK" name="DATABLOCK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="DATAFILE">
    <!--Class: DATAFILE-->
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute use="required" type="xsd:boolean" name="LATEBOUND-DATAFILE"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="DATAFORMAT">
    <!--Class: DATAFORMAT-->
    <xsd:attribute use="required" type="DATAFORMAT-SELECTION" name="SELECTION"/>
    <xsd:attribute use="optional" type="xsd:string" name="USER-SELECTION"/>
</xsd:complexType>
<xsd:simpleType name="DATAFORMAT-SELECTION">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="INTEL-HEX"/>
        <xsd:enumeration value="MOTOROLA-S"/>
        <xsd:enumeration value="BINARY"/>
        <xsd:enumeration value="USER-DEFINED"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="false" name="DATA-ID-ITEM">
    <!--Class: DATA-ID-ITEM-->
    <xsd:complexContent>
        <xsd:extension base="CONFIG-ITEM"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="DATA-OBJECT-PROP">
    <!--Class: DATA-OBJECT-PROP-->
    <xsd:complexContent>
        <xsd:extension base="DOP-BASE">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" type="COMPU-METHOD" name="COMPU-METHOD"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="DIAG-CODED-TYPE" name="DIAG-CODED-TYPE"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="PHYSICAL-TYPE" name="PHYSICAL-TYPE"/>
```

```
                <xsd:element maxOccurs="1" minOccurs="0" type="INTERNAL-CONSTR" name="INTERNAL-CONSTR"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="UNIT-REF" type="ODXLINK"/>
                <xsd:element maxOccurs="1" minOccurs="0" type="INTERNAL-CONSTR" name="PHYS-CONSTR"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="DATA-OBJECT-PROPS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="DATA-OBJECT-PROP" name="DATA-OBJECT-PROP"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="DATA-RECORD">
    <!--Class: DATA-RECORD-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="RULE"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="KEY"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="IDENT-VALUE" name="DATA-ID"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        <xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="0" type="DATAFILE" name="DATAFILE"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="DATA" name="DATA"/>
        </xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="AUDIENCE"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="DATAFORMAT-SELECTION" name="DATAFORMAT"/>
</xsd:complexType>
<xsd:complexType name="DATA-RECORDS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="DATA-RECORD" name="DATA-RECORD"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="DATA-TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="A_INT32"/>
        <xsd:enumeration value="A_UINT32"/>
        <xsd:enumeration value="A_FLOAT32"/>
        <xsd:enumeration value="A_FLOAT64"/>
        <xsd:enumeration value="A_ASCIISTRING"/>
        <xsd:enumeration value="A_UTF8STRING"/>
        <xsd:enumeration value="A_UNICODE2STRING"/>
        <xsd:enumeration value="A_BYTEFIELD"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="false" name="DEFAULT-CASE">
    <!--Class: DEFAULT-CASE-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:choice maxOccurs="1" minOccurs="0">
            <xsd:element name="STRUCTURE-REF" type="ODXLINK"/>
            <xsd:element name="STRUCTURE-SNREF" type="SNREF"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="DESCRIPTION">
```

```
        <!--Class: DESCRIPTION-->
        <xsd:sequence>
            <xsd:group maxOccurs="unbounded" minOccurs="0" ref="block"/>
            <xsd:element type="EXTERNAL-DOCS" name="EXTERNAL-DOCS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="optional" type="xsd:string" name="TI"/>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DETERMINE-NUMBER-OF-ITEMS">
        <!--Class: DETERMINE-NUMBER-OF-ITEMS-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="BYTE-POSITION"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="BIT-POSITION">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:unsignedInt">
                        <xsd:maxExclusive value="8"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element maxOccurs="1" minOccurs="1" name="DATA-OBJECT-PROP-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="DIAG-CLASS-TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="STARTCOMM"/>
            <xsd:enumeration value="STOPCOMM"/>
            <xsd:enumeration value="VARIANTIDENTIFICATION"/>
            <xsd:enumeration value="READ-DYN-DEF-MESSAGE"/>
            <xsd:enumeration value="DYN-DEF-MESSAGE"/>
            <xsd:enumeration value="CLEAR-DYN-DEF-MESSAGE"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType abstract="true" name="DIAG-CODED-TYPE">
        <!--Class: DIAG-CODED-TYPE-->
        <xsd:attribute use="optional" type="ENCODING" name="BASE-TYPE-ENCODING"/>
        <xsd:attribute use="required" type="DATA-TYPE" name="BASE-DATA-TYPE"/>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-HIGHLOW-BYTE-ORDER"/>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="DIAG-COMM">
        <!--Class: DIAG-COMM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="FUNCT-CLASS-REFS" name="FUNCT-CLASS-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="AUDIENCE"/>
            <xsd:element type="PROTOCOL-SNREFS" name="PROTOCOL-SNREFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="RELATED-DIAG-COMM-REFS" name="RELATED-DIAG-COMM-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="PRE-CONDITION-STATE-REFS" name="PRE-CONDITION-STATE-REFS" minOccurs="0"
maxOccurs="1"/>
            <xsd:element type="STATE-TRANSITION-REFS" name="STATE-TRANSITION-REFS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
        <xsd:attribute use="optional" type="xsd:string" name="SEMANTIC"/>
        <xsd:attribute use="optional" type="DIAG-CLASS-TYPE" name="DIAGNOSTIC-CLASS"/>
        <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-MANDATORY"/>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-EXECUTABLE"/>
        <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-FINAL"/>
```

```
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DIAG-COMM-DATA-CONNECTOR">
        <!--Class: DIAG-COMM-DATA-CONNECTOR-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="UNCOMPRESSED-SIZE"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="SOURCE-START-ADDRESS"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="READ-DIAG-COMM-CONNECTOR" name="READ-DIAG-COMM-
CONNECTOR"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="WRITE-DIAG-COMM-CONNECTOR" name="WRITE-DIAG-COMM-
CONNECTOR"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DIAG-COMM-DATA-CONNECTORS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="DIAG-COMM-DATA-CONNECTOR" name="DIAG-COMM-DATA-
CONNECTOR"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DIAG-COMMS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:group maxOccurs="unbounded" minOccurs="1" ref="DIAG-COMM-PROXY"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DIAG-DATA-DICTIONARY-SPEC">
        <!--Class: DIAG-DATA-DICTIONARY-SPEC-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
            <xsd:element type="DTC-DOPS" name="DTC-DOPS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="ENV-DATA-DESCS" name="ENV-DATA-DESCS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="DATA-OBJECT-PROPS" name="DATA-OBJECT-PROPS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="STRUCTURES" name="STRUCTURES" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="STATIC-FIELDS" name="STATIC-FIELDS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="DYNAMIC-LENGTH-FIELDS" name="DYNAMIC-LENGTH-FIELDS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="DYNAMIC-ENDMARKER-FIELDS" name="DYNAMIC-ENDMARKER-FIELDS" minOccurs="0"
maxOccurs="1"/>
            <xsd:element type="END-OF-PDU-FIELDS" name="END-OF-PDU-FIELDS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="MUXS" name="MUXS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="ENV-DATAS" name="ENV-DATAS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="UNIT-SPEC" name="UNIT-SPEC"/>
            <xsd:element type="TABLES" name="TABLES" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="DIAG-LAYER">
        <!--Class: DIAG-LAYER-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
            <xsd:element type="COMPANY-DATAS" name="COMPANY-DATAS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="FUNCT-CLASSS" name="FUNCT-CLASSS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="DIAG-DATA-DICTIONARY-SPEC" name="DIAG-DATA-DICTIONARY-
SPEC"/>
            <xsd:element type="DIAG-COMMS" name="DIAG-COMMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="REQUESTS" name="REQUESTS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="POS-RESPONSES" name="POS-RESPONSES" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="NEG-RESPONSES" name="NEG-RESPONSES" minOccurs="0" maxOccurs="1"/>
```

```
            <xsd:element type="GLOBAL-NEG-RESPONSES" name="GLOBAL-NEG-RESPONSES" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="IMPORT-REFS" name="IMPORT-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="STATE-CHARTS" name="STATE-CHARTS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="ADDITIONAL-AUDIENCES" name="ADDITIONAL-AUDIENCES" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="SUB-COMPONENTS" name="SUB-COMPONENTS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="LIBRARYS" name="LIBRARYS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DIAG-LAYER-CONTAINER">
        <!--Class: DIAG-LAYER-CONTAINER-->
        <xsd:complexContent>
            <xsd:extension base="ODX-CATEGORY">
                <xsd:sequence>
                    <xsd:element type="PROTOCOLS" name="PROTOCOLS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="FUNCTIONAL-GROUPS" name="FUNCTIONAL-GROUPS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="ECU-SHARED-DATAS" name="ECU-SHARED-DATAS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="BASE-VARIANTS" name="BASE-VARIANTS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="ECU-VARIANTS" name="ECU-VARIANTS" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="DIAG-LAYER-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="DIAG-LAYER-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DIAG-OBJECT-CONNECTOR">
        <!--Class: DIAG-OBJECT-CONNECTOR-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element type="FUNCTION-DIAG-COMM-CONNECTORS" name="FUNCTION-DIAG-COMM-CONNECTORS" minOccurs="0"
maxOccurs="1"/>
            <xsd:element type="TABLE-ROW-CONNECTORS" name="TABLE-ROW-CONNECTORS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="ENV-DATA-CONNECTORS" name="ENV-DATA-CONNECTORS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="DTC-CONNECTORS" name="DTC-CONNECTORS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DIAG-SERVICE">
        <!--Class: DIAG-SERVICE-->
        <xsd:complexContent>
            <xsd:extension base="DIAG-COMM">
                <xsd:sequence>
                    <xsd:element type="COMPARAM-REFS" name="COMPARAM-REFS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" name="REQUEST-REF" type="ODXLINK"/>
                    <xsd:element type="POS-RESPONSE-REFS" name="POS-RESPONSE-REFS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="NEG-RESPONSE-REFS" name="NEG-RESPONSE-REFS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="POS-RESPONSE-SUPPRESSABLE" name="POS-RESPONSE-
SUPPRESSABLE"/>
                </xsd:sequence>
                <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-CYCLIC"/>
                <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-MULTIPLE"/>
```

```
                <xsd:attribute default="PHYSICAL" use="optional" type="ADDRESSING" name="ADDRESSING"/>
                <xsd:attribute default="SEND-AND-RECEIVE" use="optional" type="TRANS-MODE" name="TRANSMISSION-
MODE"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DIAG-VARIABLE">
        <!--Class: DIAG-VARIABLE-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="VARIABLE-GROUP-REF" type="ODXLINK"/>
            <xsd:element type="SW-VARIABLES" name="SW-VARIABLES" minOccurs="0" maxOccurs="1"/>
            <xsd:choice>
                <xsd:element type="COMM-RELATIONS" name="COMM-RELATIONS" minOccurs="0" maxOccurs="1"/>
                <xsd:element maxOccurs="1" minOccurs="0" type="SNREF-TO-TABLEROW" name="SNREF-TO-TABLEROW"/>
            </xsd:choice>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
        <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-READ-BEFORE-WRITE"/>
    </xsd:complexType>
    <xsd:complexType name="DIAG-VARIABLES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:group maxOccurs="unbounded" minOccurs="1" ref="DIAG-VARIABLE-PROXY"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="DIRECTION">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="DOWNLOAD"/>
            <xsd:enumeration value="UPLOAD"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="DISABLED-AUDIENCE-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="DISABLED-AUDIENCE-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DOC-REVISION">
        <!--Class: DOC-REVISION-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" name="TEAM-MEMBER-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="REVISION-LABEL"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="STATE"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:dateTime" name="DATE"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="TOOL"/>
            <xsd:element type="COMPANY-REVISION-INFOS" name="COMPANY-REVISION-INFOS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="MODIFICATIONS" name="MODIFICATIONS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DOC-REVISIONS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="DOC-REVISION" name="DOC-REVISION"/>
        </xsd:sequence>
```

```
</xsd:complexType>
<xsd:simpleType name="DOCTYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="FLASH"/>
        <xsd:enumeration value="CONTAINER"/>
        <xsd:enumeration value="LAYER"/>
        <xsd:enumeration value="MULTIPLE-ECU-JOB-SPEC"/>
        <xsd:enumeration value="COMPARAM-SPEC"/>
        <xsd:enumeration value="VEHICLE-INFO-SPEC"/>
        <xsd:enumeration value="COMPARAM-SUBSET"/>
        <xsd:enumeration value="ECU-CONFIG"/>
        <xsd:enumeration value="FUNCTION-DICTIONARY-SPEC"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="true" name="DOP-BASE">
    <!--Class: DOP-BASE-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType abstract="false" name="DTC">
    <!--Class: DTC-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="SHORT-NAME">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:pattern value="[a-zA-Z0-9_]{1,128}"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="TROUBLE-CODE"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="DISPLAY-TROUBLE-CODE"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="TEXT" name="TEXT"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedByte" name="LEVEL"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-TEMPORARY"/>
</xsd:complexType>
<xsd:complexType abstract="false" name="DTC-CONNECTOR">
    <!--Class: DTC-CONNECTOR-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="DTC-DOP-REF" type="ODXLINK"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="DTC-SNREF" type="SNREF"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DTC-CONNECTORS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="DTC-CONNECTOR" name="DTC-CONNECTOR"/>
    </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType abstract="false" name="DTC-DOP">
    <!--Class: DTC-DOP-->
    <xsd:complexContent>
        <xsd:extension base="DOP-BASE">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" type="DIAG-CODED-TYPE" name="DIAG-CODED-TYPE"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="PHYSICAL-TYPE" name="PHYSICAL-TYPE"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="COMPU-METHOD" name="COMPU-METHOD"/>
                <xsd:element type="DTCS" name="DTCS" minOccurs="1" maxOccurs="1"/>
                <xsd:element type="LINKED-DTC-DOPS" name="LINKED-DTC-DOPS" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-VISIBLE"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="DTC-DOPS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="DTC-DOP" name="DTC-DOP"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DTCS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:group maxOccurs="unbounded" minOccurs="1" ref="DTC-PROXY"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="DTC-VALUE">
    <!--Class: DTC-VALUE-->
    <xsd:simpleContent>
        <xsd:extension base="xsd:unsignedInt"/>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="DTC-VALUES">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="DTC-VALUE" name="DTC-VALUE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="DYNAMIC">
    <!--Class: DYNAMIC-->
    <xsd:complexContent>
        <xsd:extension base="POSITIONABLE-PARAM"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="DYNAMIC-ENDMARKER-FIELD">
    <!--Class: DYNAMIC-ENDMARKER-FIELD-->
    <xsd:complexContent>
        <xsd:extension base="FIELD">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" type="DYN-END-DOP-REF" name="DATA-OBJECT-PROP-REF"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="DYNAMIC-ENDMARKER-FIELDS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
```

```
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="DYNAMIC-ENDMARKER-FIELD" name="DYNAMIC-ENDMARKER-
FIELD"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DYNAMIC-LENGTH-FIELD">
        <!--Class: DYNAMIC-LENGTH-FIELD-->
        <xsd:complexContent>
            <xsd:extension base="FIELD">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="OFFSET"/>
                    <xsd:element maxOccurs="1" minOccurs="1" type="DETERMINE-NUMBER-OF-ITEMS" name="DETERMINE-
NUMBER-OF-ITEMS"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="DYNAMIC-LENGTH-FIELDS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="DYNAMIC-LENGTH-FIELD" name="DYNAMIC-LENGTH-
FIELD"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DYN-DEFINED-SPEC">
        <!--Class: DYN-DEFINED-SPEC-->
        <xsd:sequence>
            <xsd:element type="DYN-ID-DEF-MODE-INFOS" name="DYN-ID-DEF-MODE-INFOS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DYN-END-DOP-REF">
        <!--Association Class: DYN-END-DOP-REF-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="TERMINATION-VALUE"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="ODXLINK-ATTR"/>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="DYN-ID-DEF-MODE-INFO">
        <!--Class: DYN-ID-DEF-MODE-INFO-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="DEF-MODE"/>
            <xsd:choice maxOccurs="1" minOccurs="1">
                <xsd:element name="CLEAR-DYN-DEF-MESSAGE-REF" type="ODXLINK"/>
                <xsd:element name="CLEAR-DYN-DEF-MESSAGE-SNREF" type="SNREF"/>
            </xsd:choice>
            <xsd:choice maxOccurs="1" minOccurs="1">
                <xsd:element name="READ-DYN-DEF-MESSAGE-REF" type="ODXLINK"/>
                <xsd:element name="READ-DYN-DEF-MESSAGE-SNREF" type="SNREF"/>
            </xsd:choice>
            <xsd:choice maxOccurs="1" minOccurs="1">
                <xsd:element name="DYN-DEF-MESSAGE-REF" type="ODXLINK"/>
                <xsd:element name="DYN-DEF-MESSAGE-SNREF" type="SNREF"/>
            </xsd:choice>
            <xsd:element type="SUPPORTED-DYN-IDS" name="SUPPORTED-DYN-IDS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="SELECTION-TABLE-REFS" name="SELECTION-TABLE-REFS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DYN-ID-DEF-MODE-INFOS">
        <!-- Automatically generated wrapper element type -->
```

```
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="DYN-ID-DEF-MODE-INFO" name="DYN-ID-DEF-MODE-
INFO"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ECU-CONFIG">
        <!--Class: ECU-CONFIG-->
        <xsd:complexContent>
            <xsd:extension base="ODX-CATEGORY">
                <xsd:sequence>
                    <xsd:element type="CONFIG-DATAS" name="CONFIG-DATAS" minOccurs="1" maxOccurs="1"/>
                    <xsd:element type="ADDITIONAL-AUDIENCES" name="ADDITIONAL-AUDIENCES" minOccurs="0"
maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="CONFIG-DATA-DICTIONARY-SPEC" name="CONFIG-DATA-
DICTIONARY-SPEC"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ECU-GROUP">
        <!--Class: ECU-GROUP-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element type="GROUP-MEMBERS" name="GROUP-MEMBERS" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType name="ECU-GROUPS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="ECU-GROUP" name="ECU-GROUP"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ECU-MEM">
        <!--Class: ECU-MEM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="MEM" name="MEM"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="PHYS-MEM" name="PHYS-MEM"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ECU-MEM-CONNECTOR">
        <!--Class: ECU-MEM-CONNECTOR-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
            <xsd:element type="FLASH-CLASSS" name="FLASH-CLASSS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="SESSION-DESCS" name="SESSION-DESCS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="IDENT-DESCS" name="IDENT-DESCS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="1" name="ECU-MEM-REF" type="ODXLINK"/>
            <xsd:element type="LAYER-REFS" name="LAYER-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="ALL-VARIANT-REFS" name="ALL-VARIANT-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
```

```
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType name="ECU-MEM-CONNECTORS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="ECU-MEM-CONNECTOR" name="ECU-MEM-CONNECTOR"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ECU-MEMS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="ECU-MEM" name="ECU-MEM"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="ECU-PROXY">
    <!--Class: ECU-PROXY-->
    <xsd:complexContent>
        <xsd:extension base="INFO-COMPONENT"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ECU-PROXY-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="ECU-PROXY-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="ECU-SHARED-DATA">
    <!--Class: ECU-SHARED-DATA-->
    <xsd:complexContent>
        <xsd:extension base="DIAG-LAYER">
            <xsd:sequence>
                <xsd:element type="DIAG-VARIABLES" name="DIAG-VARIABLES" minOccurs="0" maxOccurs="1"/>
                <xsd:element type="VARIABLE-GROUPS" name="VARIABLE-GROUPS" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="ECU-SHARED-DATA-REF">
    <!--Class: ECU-SHARED-DATA-REF-->
    <xsd:complexContent>
        <xsd:extension base="PARENT-REF"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ECU-SHARED-DATAS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="ECU-SHARED-DATA" name="ECU-SHARED-DATA"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="ECU-VARIANT">
    <!--Class: ECU-VARIANT-->
    <xsd:complexContent>
        <xsd:extension base="HIERARCHY-ELEMENT">
            <xsd:sequence>
                <xsd:element type="DIAG-VARIABLES" name="DIAG-VARIABLES" minOccurs="0" maxOccurs="1"/>
                <xsd:element type="VARIABLE-GROUPS" name="VARIABLE-GROUPS" minOccurs="0" maxOccurs="1"/>
```

```
                    <xsd:element type="ECU-VARIANT-PATTERNS" name="ECU-VARIANT-PATTERNS" minOccurs="0"
maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="DYN-DEFINED-SPEC" name="DYN-DEFINED-SPEC"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="PARENT-REFS" name="PARENT-REFS"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="ECU-VARIANT-PATTERN">
        <!--Class: ECU-VARIANT-PATTERN-->
        <xsd:sequence>
            <xsd:element type="MATCHING-PARAMETERS" name="MATCHING-PARAMETERS" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ECU-VARIANT-PATTERNS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="ECU-VARIANT-PATTERN" name="ECU-VARIANT-PATTERN"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ECU-VARIANT-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="ECU-VARIANT-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ECU-VARIANTS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="ECU-VARIANT" name="ECU-VARIANT"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ECU-VARIANT-SNREFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="ECU-VARIANT-SNREF" type="SNREF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ENABLED-AUDIENCE-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="ENABLED-AUDIENCE-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="ENCODING">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="BCD-P"/>
            <xsd:enumeration value="BCD-UP"/>
            <xsd:enumeration value="1C"/>
            <xsd:enumeration value="2C"/>
            <xsd:enumeration value="SM"/>
            <xsd:enumeration value="UTF-8"/>
            <xsd:enumeration value="UCS-2"/>
            <xsd:enumeration value="ISO-8859-1"/>
            <xsd:enumeration value="ISO-8859-2"/>
            <xsd:enumeration value="WINDOWS-1252"/>
            <xsd:enumeration value="NONE"/>
        </xsd:restriction>
```

```
</xsd:simpleType>
<xsd:complexType abstract="false" name="ENCRYPT-COMPRESS-METHOD">
    <!--Class: ENCRYPT-COMPRESS-METHOD-->
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute use="required" type="ENCRYPT-COMPRESS-METHOD-TYPE" name="TYPE"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:simpleType name="ENCRYPT-COMPRESS-METHOD-TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="A_UINT32"/>
        <xsd:enumeration value="A_BYTEFIELD"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="false" name="END-OF-PDU-FIELD">
    <!--Class: END-OF-PDU-FIELD-->
    <xsd:complexContent>
        <xsd:extension base="FIELD">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="MAX-NUMBER-OF-ITEMS"/>
                <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="MIN-NUMBER-OF-ITEMS"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="END-OF-PDU-FIELDS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="END-OF-PDU-FIELD" name="END-OF-PDU-FIELD"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="ENV-DATA">
    <!--Class: ENV-DATA-->
    <xsd:complexContent>
        <xsd:extension base="BASIC-STRUCTURE">
            <xsd:choice>
                <xsd:element maxOccurs="1" minOccurs="1" type="ALL-VALUE" name="ALL-VALUE"/>
                <xsd:element type="DTC-VALUES" name="DTC-VALUES" minOccurs="1" maxOccurs="1"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="ENV-DATA-CONNECTOR">
    <!--Class: ENV-DATA-CONNECTOR-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="ENV-DATA-DESC-REF" type="ODXLINK"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="ENV-DATA-SNREF" type="SNREF"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ENV-DATA-CONNECTORS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="ENV-DATA-CONNECTOR" name="ENV-DATA-CONNECTOR"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="ENV-DATA-DESC">
```

```
    <!--Class: ENV-DATA-DESC-->
    <xsd:complexContent>
        <xsd:extension base="COMPLEX-DOP">
            <xsd:sequence>
                <xsd:choice maxOccurs="1" minOccurs="1">
                    <xsd:element name="PARAM-SNREF" type="SNREF"/>
                    <xsd:element name="PARAM-SNPATHREF" type="SNPATHREF"/>
                </xsd:choice>
                <xsd:element type="ENV-DATA-REFS" name="ENV-DATA-REFS" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ENV-DATA-DESCS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="ENV-DATA-DESC" name="ENV-DATA-DESC"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ENV-DATA-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="ENV-DATA-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ENV-DATAS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="ENV-DATA" name="ENV-DATA"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="EXPECTED-IDENT">
    <!--Class: EXPECTED-IDENT-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element type="IDENT-VALUES" name="IDENT-VALUES" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType name="EXPECTED-IDENTS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="EXPECTED-IDENT" name="EXPECTED-IDENT"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="EXTERNAL-ACCESS-METHOD">
    <!--Class: EXTERNAL-ACCESS-METHOD-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="METHOD"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType abstract="false" name="EXTERNAL-DOC">
    <!--Class: EXTERNAL-DOC-->
    <xsd:simpleContent>
```

```
            <xsd:extension base="xsd:string">
                <xsd:attribute use="required" type="xsd:anyURI" name="HREF"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="EXTERNAL-DOCS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="EXTERNAL-DOC" name="EXTERNAL-DOC"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="EXTERN-FLASHDATA">
        <!--Class: EXTERN-FLASHDATA-->
        <xsd:complexContent>
            <xsd:extension base="FLASHDATA">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="DATAFILE" name="DATAFILE"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="FIELD">
        <!--Class: FIELD-->
        <xsd:complexContent>
            <xsd:extension base="COMPLEX-DOP">
                <xsd:choice>
                    <xsd:choice maxOccurs="1" minOccurs="1">
                        <xsd:element name="BASIC-STRUCTURE-REF" type="ODXLINK"/>
                        <xsd:element name="BASIC-STRUCTURE-SNREF" type="SNREF"/>
                    </xsd:choice>
                    <xsd:choice maxOccurs="1" minOccurs="1">
                        <xsd:element name="ENV-DATA-DESC-REF" type="ODXLINK"/>
                        <xsd:element name="ENV-DATA-DESC-SNREF" type="SNREF"/>
                    </xsd:choice>
                </xsd:choice>
                <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-VISIBLE"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="FILTER">
        <!--Class: FILTER-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="FILTER-START"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FILTERS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FILTER" name="FILTER"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FLASH">
        <!--Class: FLASH-->
        <xsd:complexContent>
            <xsd:extension base="ODX-CATEGORY">
                <xsd:sequence>
                    <xsd:element type="ECU-MEMS" name="ECU-MEMS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="ECU-MEM-CONNECTORS" name="ECU-MEM-CONNECTORS" minOccurs="0" maxOccurs="1"/>
```

```
                    <xsd:element type="ADDITIONAL-AUDIENCES" name="ADDITIONAL-AUDIENCES" minOccurs="0"
maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FLASH-CLASS">
        <!--Class: FLASH-CLASS-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType name="FLASH-CLASS-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="FLASH-CLASS-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FLASH-CLASSS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FLASH-CLASS" name="FLASH-CLASS"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="FLASHDATA">
        <!--Class: FLASHDATA-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="SIZE-LENGTH"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="ADDRESS-LENGTH"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="DATAFORMAT" name="DATAFORMAT"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ENCRYPT-COMPRESS-METHOD" name="ENCRYPT-COMPRESS-METHOD"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType name="FLASHDATAS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FLASHDATA" name="FLASHDATA"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCT-CLASS">
        <!--Class: FUNCT-CLASS-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType name="FUNCT-CLASS-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="FUNCT-CLASS-REF" type="ODXLINK"/>
        </xsd:sequence>
```

```
    </xsd:complexType>
    <xsd:complexType name="FUNCT-CLASSS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FUNCT-CLASS" name="FUNCT-CLASS"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCTIONAL-GROUP">
        <!--Class: FUNCTIONAL-GROUP-->
        <xsd:complexContent>
            <xsd:extension base="HIERARCHY-ELEMENT">
                <xsd:sequence>
                    <xsd:element type="DIAG-VARIABLES" name="DIAG-VARIABLES" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="VARIABLE-GROUPS" name="VARIABLE-GROUPS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="PARENT-REFS" name="PARENT-REFS"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCTIONAL-GROUP-REF">
        <!--Class: FUNCTIONAL-GROUP-REF-->
        <xsd:complexContent>
            <xsd:extension base="PARENT-REF"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="FUNCTIONAL-GROUPS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FUNCTIONAL-GROUP" name="FUNCTIONAL-GROUP"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCTION-DIAG-COMM-CONNECTOR">
        <!--Class: FUNCTION-DIAG-COMM-CONNECTOR-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" name="LOGICAL-LINK-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="1" name="DIAG-COMM-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FUNCTION-DIAG-COMM-CONNECTORS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FUNCTION-DIAG-COMM-CONNECTOR" name="FUNCTION-
DIAG-COMM-CONNECTOR"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCTION-DICTIONARY">
        <!--Class: FUNCTION-DICTIONARY-->
        <xsd:complexContent>
            <xsd:extension base="ODX-CATEGORY">
                <xsd:sequence>
                    <xsd:element type="FUNCTION-NODES" name="FUNCTION-NODES" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="FUNCTION-NODE-GROUPS" name="FUNCTION-NODE-GROUPS" minOccurs="0"
maxOccurs="1"/>
                    <xsd:element type="ADDITIONAL-AUDIENCES" name="ADDITIONAL-AUDIENCES" minOccurs="0"
maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
```

```
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCTION-IN-PARAM">
        <!--Class: FUNCTION-IN-PARAM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="UNIT-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="PHYSICAL-TYPE" name="PHYSICAL-TYPE"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="IN-PARAM-IF-SNREF" type="SNREF"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="FUNCTION-DIAG-COMM-CONNECTOR" name="FUNCTION-DIAG-COMM-
CONNECTOR"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FUNCTION-IN-PARAMS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FUNCTION-IN-PARAM" name="FUNCTION-IN-PARAM"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCTION-NODE">
        <!--Class: FUNCTION-NODE-->
        <xsd:complexContent>
            <xsd:extension base="BASE-FUNCTION-NODE">
                <xsd:sequence>
                    <xsd:element type="FUNCTION-NODES" name="FUNCTION-NODES" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FUNCTION-NODE-GROUP">
        <!--Class: FUNCTION-NODE-GROUP-->
        <xsd:complexContent>
            <xsd:extension base="BASE-FUNCTION-NODE">
                <xsd:sequence>
                    <xsd:element type="FUNCTION-NODE-REFS" name="FUNCTION-NODE-REFS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="FUNCTION-NODE-GROUPS" name="FUNCTION-NODE-GROUPS" minOccurs="0"
maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="FUNCTION-NODE-GROUPS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FUNCTION-NODE-GROUP" name="FUNCTION-NODE-GROUP"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FUNCTION-NODE-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="FUNCTION-NODE-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FUNCTION-NODES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FUNCTION-NODE" name="FUNCTION-NODE"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
    <xsd:complexType abstract="false" name="FUNCTION-OUT-PARAM">
        <!--Class: FUNCTION-OUT-PARAM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="UNIT-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="PHYSICAL-TYPE" name="PHYSICAL-TYPE"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="OUT-PARAM-IF-SNREF" type="SNREF"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="FUNCTION-DIAG-COMM-CONNECTOR" name="FUNCTION-DIAG-COMM-
CONNECTOR"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FUNCTION-OUT-PARAMS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="FUNCTION-OUT-PARAM" name="FUNCTION-OUT-PARAM"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FW-CHECKSUM">
        <!--Class: FW-CHECKSUM-->
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute use="required" type="SESSION-SUB-ELEM-TYPE" name="TYPE"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="FW-SIGNATURE">
        <!--Class: FW-SIGNATURE-->
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute use="required" type="SESSION-SUB-ELEM-TYPE" name="TYPE"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="GATEWAY-LOGICAL-LINK">
        <!--Class: GATEWAY-LOGICAL-LINK-->
        <xsd:complexContent>
            <xsd:extension base="LOGICAL-LINK">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="SEMANTIC"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="GATEWAY-LOGICAL-LINK-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="GATEWAY-LOGICAL-LINK-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="GLOBAL-NEG-RESPONSE">
        <!--Class: GLOBAL-NEG-RESPONSE-->
        <xsd:complexContent>
            <xsd:extension base="RESPONSE"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="GLOBAL-NEG-RESPONSES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
```

```
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="GLOBAL-NEG-RESPONSE" name="GLOBAL-NEG-RESPONSE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="GROUP-MEMBER">
    <!--Class: GROUP-MEMBER-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="BASE-VARIANT-REF" type="ODXLINK"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="FUNCT-RESOLUTION-LINK-REF" type="ODXLINK"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="PHYS-RESOLUTION-LINK-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GROUP-MEMBERS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="GROUP-MEMBER" name="GROUP-MEMBER"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="true" name="HIERARCHY-ELEMENT">
    <!--Class: HIERARCHY-ELEMENT-->
    <xsd:complexContent>
        <xsd:extension base="DIAG-LAYER">
            <xsd:sequence>
                <xsd:element type="COMPARAM-REFS" name="COMPARAM-REFS" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="IDENT-DESC">
    <!--Class: IDENT-DESC-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="DIAG-COMM-SNREF" type="SNREF"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="IDENT-IF-SNREF" type="SNREF"/>
        <xsd:choice maxOccurs="1" minOccurs="1">
            <xsd:element name="OUT-PARAM-IF-SNREF" type="SNREF"/>
            <xsd:element name="OUT-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="IDENT-DESCS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="IDENT-DESC" name="IDENT-DESC"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="IDENT-VALUE">
    <!--Class: IDENT-VALUE-->
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute use="required" type="IDENT-VALUE-TYPE" name="TYPE"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="IDENT-VALUES">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="IDENT-VALUE" name="IDENT-VALUE"/>
    </xsd:sequence>
</xsd:complexType>
```

```
<xsd:simpleType name="IDENT-VALUE-TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="A_UINT32"/>
        <xsd:enumeration value="A_BYTEFIELD"/>
        <xsd:enumeration value="A_ASCIISTRING"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="IMPORT-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="IMPORT-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="true" name="INFO-COMPONENT">
    <!--Class: INFO-COMPONENT-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element type="MATCHING-COMPONENTS" name="MATCHING-COMPONENTS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType name="INFO-COMPONENT-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="INFO-COMPONENT-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="INFO-COMPONENTS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="INFO-COMPONENT" name="INFO-COMPONENT"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="IN-PARAM-IF-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:choice maxOccurs="unbounded" minOccurs="1">
        <xsd:element name="IN-PARAM-IF-SNREF" type="SNREF"/>
        <xsd:element name="IN-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
    </xsd:choice>
</xsd:complexType>
<xsd:complexType abstract="false" name="INPUT-PARAM">
    <!--Class: INPUT-PARAM-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="PHYSICAL-DEFAULT-VALUE"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="DOP-BASE-REF" type="ODXLINK"/>
    </xsd:sequence>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    <xsd:attribute use="optional" type="xsd:string" name="SEMANTIC"/>
</xsd:complexType>
<xsd:complexType name="INPUT-PARAMS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="INPUT-PARAM" name="INPUT-PARAM"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="INTERNAL-CONSTR">
```

```
    <!--Class: INTERNAL-CONSTR-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" type="LIMIT" name="LOWER-LIMIT"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="LIMIT" name="UPPER-LIMIT"/>
        <xsd:element type="SCALE-CONSTRS" name="SCALE-CONSTRS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="INTERN-FLASHDATA">
    <!--Class: INTERN-FLASHDATA-->
    <xsd:complexContent>
        <xsd:extension base="FLASHDATA">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="DATA"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="INTERVAL-TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="OPEN"/>
        <xsd:enumeration value="CLOSED"/>
        <xsd:enumeration value="INFINITE"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="false" name="ITEM-VALUE">
    <!--Class: ITEM-VALUE-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="PHYS-CONSTANT-VALUE"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="TEXT" name="MEANING"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="KEY"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="RULE"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="TEXT" name="DESCRIPTION"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="AUDIENCE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ITEM-VALUES">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="ITEM-VALUE" name="ITEM-VALUE"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LAYER-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="LAYER-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="LEADING-LENGTH-INFO-TYPE">
    <!--Class: LEADING-LENGTH-INFO-TYPE-->
    <xsd:complexContent>
        <xsd:extension base="DIAG-CODED-TYPE">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="BIT-LENGTH">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:unsignedInt">
                            <xsd:minExclusive value="0"/>
                        </xsd:restriction>
```

```
                    </xsd:simpleType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="LENGTH-KEY">
    <!--Class: LENGTH-KEY-->
    <xsd:complexContent>
        <xsd:extension base="POSITIONABLE-PARAM">
            <xsd:choice maxOccurs="1" minOccurs="1">
                <xsd:element name="DOP-REF" type="ODXLINK"/>
                <xsd:element name="DOP-SNREF" type="SNREF"/>
            </xsd:choice>
            <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="LIBRARY">
    <!--Class: LIBRARY-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="CODE-FILE"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="ENCRYPTION"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="SYNTAX"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="REVISION"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="ENTRYPOINT"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType name="LIBRARY-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="LIBRARY-REF" type="ODXLINK"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LIBRARYS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="LIBRARY" name="LIBRARY"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="LIMIT">
    <!--Class: LIMIT-->
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute default="CLOSED" use="optional" type="INTERVAL-TYPE" name="INTERVAL-TYPE"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="LINK-COMPARAM-REF">
    <!--Class: LINK-COMPARAM-REF-->
    <xsd:sequence>
        <xsd:choice>
            <xsd:element maxOccurs="1" minOccurs="0" type="SIMPLE-VALUE" name="SIMPLE-VALUE"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="COMPLEX-VALUE" name="COMPLEX-VALUE"/>
        </xsd:choice>
```

```
            <xsd:element maxOccurs="1" minOccurs="0" type="DESCRIPTION" name="DESC"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="ODXLINK-ATTR"/>
    </xsd:complexType>
    <xsd:complexType name="LINK-COMPARAM-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="LINK-COMPARAM-REF" name="LINK-COMPARAM-REF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="LINKED-DTC-DOP">
        <!--Class: LINKED-DTC-DOP-->
        <xsd:sequence>
            <xsd:element type="NOT-INHERITED-DTC-SNREFS" name="NOT-INHERITED-DTC-SNREFS" minOccurs="0"
maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="1" name="DTC-DOP-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="LINKED-DTC-DOPS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="LINKED-DTC-DOP" name="LINKED-DTC-DOP"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="LOGICAL-LINK">
        <!--Class: LOGICAL-LINK-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element type="GATEWAY-LOGICAL-LINK-REFS" name="GATEWAY-LOGICAL-LINK-REFS" minOccurs="0"
maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="1" name="PHYSICAL-VEHICLE-LINK-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="PROTOCOL-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="FUNCTIONAL-GROUP-REF" type="ODXLINK"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="BASE-VARIANT-REF" type="ODXLINK"/>
            <xsd:element type="ECU-PROXY-REFS" name="ECU-PROXY-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="LINK-COMPARAM-REFS" name="LINK-COMPARAM-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="PROT-STACK-SNREF" type="SNREF"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType name="LOGICAL-LINKS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="LOGICAL-LINK" name="LOGICAL-LINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="LONG-NAME">
        <!-- Automatically generated wrapper element type -->
        <xsd:simpleContent>
            <xsd:restriction base="TEXT">
                <xsd:maxLength value="255"/>
            </xsd:restriction>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="MATCHING-BASE-VARIANT-PARAMETER">
        <!--Class: MATCHING-BASE-VARIANT-PARAMETER-->
        <xsd:sequence>
```

```
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="EXPECTED-VALUE"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:boolean" name="USE-PHYSICAL-ADDRESSING"/>
            <xsd:element maxOccurs="1" minOccurs="1" name="DIAG-COMM-SNREF" type="SNREF"/>
            <xsd:choice maxOccurs="1" minOccurs="1">
                <xsd:element name="OUT-PARAM-IF-SNREF" type="SNREF"/>
                <xsd:element name="OUT-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="MATCHING-BASE-VARIANT-PARAMETERS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="MATCHING-BASE-VARIANT-PARAMETER" name="MATCHING-
BASE-VARIANT-PARAMETER"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="MATCHING-COMPONENT">
        <!--Class: MATCHING-COMPONENT-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="EXPECTED-VALUE"/>
            <xsd:choice maxOccurs="1" minOccurs="1">
                <xsd:element name="OUT-PARAM-IF-SNREF" type="SNREF"/>
                <xsd:element name="OUT-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
            </xsd:choice>
            <xsd:choice>
                <xsd:element maxOccurs="1" minOccurs="1" name="MULTIPLE-ECU-JOB-REF" type="ODXLINK"/>
                <xsd:element maxOccurs="1" minOccurs="1" name="DIAG-COMM-REF" type="ODXLINK"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="MATCHING-COMPONENTS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="MATCHING-COMPONENT" name="MATCHING-COMPONENT"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="MATCHING-PARAMETER">
        <!--Class: MATCHING-PARAMETER-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="EXPECTED-VALUE"/>
            <xsd:element maxOccurs="1" minOccurs="1" name="DIAG-COMM-SNREF" type="SNREF"/>
            <xsd:choice maxOccurs="1" minOccurs="1">
                <xsd:element name="OUT-PARAM-IF-SNREF" type="SNREF"/>
                <xsd:element name="OUT-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="MATCHING-PARAMETERS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="MATCHING-PARAMETER" name="MATCHING-PARAMETER"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="MATCHING-REQUEST-PARAM">
        <!--Class: MATCHING-REQUEST-PARAM-->
        <xsd:complexContent>
            <xsd:extension base="POSITIONABLE-PARAM">
                <xsd:sequence>
```

```
                <xsd:element maxOccurs="1" minOccurs="1" type="xsd:int" name="REQUEST-BYTE-POS"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="BYTE-LENGTH"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="MEM">
    <!--Class: MEM-->
    <xsd:sequence>
        <xsd:element type="SESSIONS" name="SESSIONS" minOccurs="1" maxOccurs="1"/>
        <xsd:element type="DATABLOCKS" name="DATABLOCKS" minOccurs="1" maxOccurs="1"/>
        <xsd:element type="FLASHDATAS" name="FLASHDATAS" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="MEMBER-LOGICAL-LINK">
    <!--Class: MEMBER-LOGICAL-LINK-->
    <xsd:complexContent>
        <xsd:extension base="LOGICAL-LINK"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="MIN-MAX-LENGTH-TYPE">
    <!--Class: MIN-MAX-LENGTH-TYPE-->
    <xsd:complexContent>
        <xsd:extension base="DIAG-CODED-TYPE">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="MAX-LENGTH"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="MIN-LENGTH"/>
            </xsd:sequence>
            <xsd:attribute use="required" type="TERMINATION" name="TERMINATION"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="MODEL-YEAR">
    <!--Class: MODEL-YEAR-->
    <xsd:complexContent>
        <xsd:extension base="INFO-COMPONENT"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="MODIFICATION">
    <!--Class: MODIFICATION-->
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="CHANGE"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="REASON"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="MODIFICATIONS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="MODIFICATION" name="MODIFICATION"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="MULTIPLE-ECU-JOB">
    <!--Class: MULTIPLE-ECU-JOB-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
        <xsd:element type="FUNCT-CLASS-REFS" name="FUNCT-CLASS-REFS" minOccurs="0" maxOccurs="1"/>
```

```
            <xsd:element type="PROG-CODES" name="PROG-CODES" minOccurs="1" maxOccurs="1"/>
            <xsd:element type="INPUT-PARAMS" name="INPUT-PARAMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="OUTPUT-PARAMS" name="OUTPUT-PARAMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="NEG-OUTPUT-PARAMS" name="NEG-OUTPUT-PARAMS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="DIAG-LAYER-REFS" name="DIAG-LAYER-REFS" minOccurs="0" maxOccurs="1"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="AUDIENCE"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
        <xsd:attribute use="optional" type="xsd:string" name="SEMANTIC"/>
        <xsd:attribute default="true" use="optional" type="xsd:boolean" name="IS-EXECUTABLE"/>
    </xsd:complexType>
    <xsd:complexType name="MULTIPLE-ECU-JOB-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="MULTIPLE-ECU-JOB-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="MULTIPLE-ECU-JOBS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="MULTIPLE-ECU-JOB" name="MULTIPLE-ECU-JOB"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="MULTIPLE-ECU-JOB-SPEC">
        <!--Class: MULTIPLE-ECU-JOB-SPEC-->
        <xsd:complexContent>
            <xsd:extension base="ODX-CATEGORY">
                <xsd:sequence>
                    <xsd:element type="MULTIPLE-ECU-JOBS" name="MULTIPLE-ECU-JOBS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="DIAG-DATA-DICTIONARY-SPEC" name="DIAG-DATA-
DICTIONARY-SPEC"/>
                    <xsd:element type="FUNCT-CLASSS" name="FUNCT-CLASSS" minOccurs="0" maxOccurs="1"/>
                    <xsd:element type="ADDITIONAL-AUDIENCES" name="ADDITIONAL-AUDIENCES" minOccurs="0"
maxOccurs="1"/>
                    <xsd:element type="IMPORT-REFS" name="IMPORT-REFS" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="MUX">
        <!--Class: MUX-->
        <xsd:complexContent>
            <xsd:extension base="COMPLEX-DOP">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:unsignedInt" name="BYTE-POSITION"/>
                    <xsd:element maxOccurs="1" minOccurs="1" type="SWITCH-KEY" name="SWITCH-KEY"/>
                    <xsd:element maxOccurs="1" minOccurs="0" type="DEFAULT-CASE" name="DEFAULT-CASE"/>
                    <xsd:element type="CASES" name="CASES" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
                <xsd:attribute default="false" use="optional" type="xsd:boolean" name="IS-VISIBLE"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="MUXS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="MUX" name="MUX"/>
```

```
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NEG-OFFSET">
        <!--Class: NEG-OFFSET-->
        <xsd:complexContent>
            <xsd:extension base="TARGET-ADDR-OFFSET">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="NEGATIVE-OFFSET"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NEG-OUTPUT-PARAM">
        <!--Class: NEG-OUTPUT-PARAM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="1" name="DOP-BASE-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NEG-OUTPUT-PARAMS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="NEG-OUTPUT-PARAM" name="NEG-OUTPUT-PARAM"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NEG-RESPONSE">
        <!--Class: NEG-RESPONSE-->
        <xsd:complexContent>
            <xsd:extension base="RESPONSE"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="NEG-RESPONSE-REFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="NEG-RESPONSE-REF" type="ODXLINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NEG-RESPONSES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="NEG-RESPONSE" name="NEG-RESPONSE"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NOT-INHERITED-DIAG-COMM">
        <!--Class: NOT-INHERITED-DIAG-COMM-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="DIAG-COMM-SNREF" type="SNREF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NOT-INHERITED-DIAG-COMMS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="NOT-INHERITED-DIAG-COMM" name="NOT-INHERITED-
DIAG-COMM"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NOT-INHERITED-DOP">
        <!--Class: NOT-INHERITED-DOP-->
```

```
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="DOP-BASE-SNREF" type="SNREF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NOT-INHERITED-DOPS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="NOT-INHERITED-DOP" name="NOT-INHERITED-DOP"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NOT-INHERITED-DTC-SNREFS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="NOT-INHERITED-DTC-SNREF" type="SNREF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NOT-INHERITED-GLOBAL-NEG-RESPONSE">
        <!--Class: NOT-INHERITED-GLOBAL-NEG-RESPONSE-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="GLOBAL-NEG-RESPONSE-SNREF" type="SNREF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NOT-INHERITED-GLOBAL-NEG-RESPONSES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="NOT-INHERITED-GLOBAL-NEG-RESPONSE" name="NOT-
INHERITED-GLOBAL-NEG-RESPONSE"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NOT-INHERITED-TABLE">
        <!--Class: NOT-INHERITED-TABLE-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="TABLE-SNREF" type="SNREF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NOT-INHERITED-TABLES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="NOT-INHERITED-TABLE" name="NOT-INHERITED-TABLE"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NOT-INHERITED-VARIABLE">
        <!--Class: NOT-INHERITED-VARIABLE-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="DIAG-VARIABLE-SNREF" type="SNREF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NOT-INHERITED-VARIABLES">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="NOT-INHERITED-VARIABLE" name="NOT-INHERITED-
VARIABLE"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="NRC-CONST">
        <!--Class: NRC-CONST-->
        <xsd:complexContent>
            <xsd:extension base="POSITIONABLE-PARAM">
```

```
            <xsd:sequence>
                <xsd:element type="CODED-VALUES" name="CODED-VALUES" minOccurs="1" maxOccurs="1"/>
                <xsd:element maxOccurs="1" minOccurs="1" type="DIAG-CODED-TYPE" name="DIAG-CODED-TYPE"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="ODX">
    <!--Class: ODX-->
    <xsd:choice>
        <xsd:element maxOccurs="1" minOccurs="1" type="DIAG-LAYER-CONTAINER" name="DIAG-LAYER-CONTAINER"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="COMPARAM-SPEC" name="COMPARAM-SPEC"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="VEHICLE-INFO-SPEC" name="VEHICLE-INFO-SPEC"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="FLASH" name="FLASH"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="ECU-CONFIG" name="ECU-CONFIG"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="MULTIPLE-ECU-JOB-SPEC" name="MULTIPLE-ECU-JOB-SPEC"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="COMPARAM-SUBSET" name="COMPARAM-SUBSET"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="FUNCTION-DICTIONARY" name="FUNCTION-DICTIONARY"/>
    </xsd:choice>
    <xsd:attribute fixed="2.2.0" use="required" type="xsd:string" name="MODEL-VERSION"/>
</xsd:complexType>
<xsd:complexType abstract="true" name="ODX-CATEGORY">
    <!--Class: ODX-CATEGORY-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="0" type="ADMIN-DATA" name="ADMIN-DATA"/>
        <xsd:element type="COMPANY-DATAS" name="COMPANY-DATAS" minOccurs="0" maxOccurs="1"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType name="ODXLINK">
    <xsd:attributeGroup ref="ODXLINK-ATTR"/>
</xsd:complexType>
<xsd:complexType abstract="false" name="OEM">
    <!--Class: OEM-->
    <xsd:complexContent>
        <xsd:extension base="INFO-COMPONENT"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="OPTION-ITEM">
    <!--Class: OPTION-ITEM-->
    <xsd:complexContent>
        <xsd:extension base="CONFIG-ITEM">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="0" type="xsd:string" name="PHYSICAL-DEFAULT-VALUE"/>
                <xsd:element type="ITEM-VALUES" name="ITEM-VALUES" minOccurs="0" maxOccurs="1"/>
                <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="WRITE-AUDIENCE"/>
                <xsd:element maxOccurs="1" minOccurs="0" type="AUDIENCE" name="READ-AUDIENCE"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="OPTION-ITEMS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="OPTION-ITEM" name="OPTION-ITEM"/>
```

```
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="OUT-PARAM-IF-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:choice maxOccurs="unbounded" minOccurs="1">
        <xsd:element name="OUT-PARAM-IF-SNREF" type="SNREF"/>
        <xsd:element name="OUT-PARAM-IF-SNPATHREF" type="SNPATHREF"/>
    </xsd:choice>
</xsd:complexType>
<xsd:complexType abstract="false" name="OUTPUT-PARAM">
    <!--Class: OUTPUT-PARAM-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="DOP-BASE-REF" type="ODXLINK"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    <xsd:attribute use="optional" type="xsd:string" name="SEMANTIC"/>
</xsd:complexType>
<xsd:complexType name="OUTPUT-PARAMS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="OUTPUT-PARAM" name="OUTPUT-PARAM"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="false" name="OWN-IDENT">
    <!--Class: OWN-IDENT-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element maxOccurs="1" minOccurs="1" type="IDENT-VALUE" name="IDENT-VALUE"/>
    </xsd:sequence>
    <xsd:attribute use="required" type="xsd:ID" name="ID"/>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
</xsd:complexType>
<xsd:complexType name="OWN-IDENTS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="OWN-IDENT" name="OWN-IDENT"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType abstract="true" name="PARAM">
    <!--Class: PARAM-->
    <xsd:sequence>
        <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
        <xsd:element type="SDGS" name="SDGS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    <xsd:attribute use="optional" type="xsd:string" name="SEMANTIC"/>
</xsd:complexType>
<xsd:complexType abstract="false" name="PARAM-LENGTH-INFO-TYPE">
    <!--Class: PARAM-LENGTH-INFO-TYPE-->
    <xsd:complexContent>
        <xsd:extension base="DIAG-CODED-TYPE">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="LENGTH-KEY-REF" type="ODXLINK"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
```

```
    </xsd:complexType>
    <xsd:complexType name="PARAMS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="PARAM" name="PARAM"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="PARENT-REF">
        <!--Class: PARENT-REF-->
        <xsd:sequence>
            <xsd:element type="NOT-INHERITED-DIAG-COMMS" name="NOT-INHERITED-DIAG-COMMS" minOccurs="0"
maxOccurs="1"/>
            <xsd:element type="NOT-INHERITED-VARIABLES" name="NOT-INHERITED-VARIABLES" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="NOT-INHERITED-DOPS" name="NOT-INHERITED-DOPS" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="NOT-INHERITED-TABLES" name="NOT-INHERITED-TABLES" minOccurs="0" maxOccurs="1"/>
            <xsd:element type="NOT-INHERITED-GLOBAL-NEG-RESPONSES" name="NOT-INHERITED-GLOBAL-NEG-RESPONSES"
minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="ODXLINK-ATTR"/>
    </xsd:complexType>
    <xsd:complexType name="PARENT-REFS">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="PARENT-REF" name="PARENT-REF"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="PHYS-CONST">
        <!--Class: PHYS-CONST-->
        <xsd:complexContent>
            <xsd:extension base="POSITIONABLE-PARAM">
                <xsd:sequence>
                    <xsd:element maxOccurs="1" minOccurs="1" type="xsd:string" name="PHYS-CONSTANT-VALUE"/>
                    <xsd:choice maxOccurs="1" minOccurs="1">
                        <xsd:element name="DOP-REF" type="ODXLINK"/>
                        <xsd:element name="DOP-SNREF" type="SNREF"/>
                    </xsd:choice>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="PHYSICAL-DATA-TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="A_INT32"/>
            <xsd:enumeration value="A_UINT32"/>
            <xsd:enumeration value="A_FLOAT32"/>
            <xsd:enumeration value="A_FLOAT64"/>
            <xsd:enumeration value="A_UNICODE2STRING"/>
            <xsd:enumeration value="A_BYTEFIELD"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType abstract="false" name="PHYSICAL-DIMENSION">
        <!--Class: PHYSICAL-DIMENSION-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:int" name="LENGTH-EXP"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:int" name="MASS-EXP"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:int" name="TIME-EXP"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:int" name="CURRENT-EXP"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:int" name="TEMPERATURE-EXP"/>
```

```
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:int" name="MOLAR-AMOUNT-EXP"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:int" name="LUMINOUS-INTENSITY-EXP"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType name="PHYSICAL-DIMENSIONS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="PHYSICAL-DIMENSION" name="PHYSICAL-DIMENSION"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="PHYSICAL-TYPE">
        <!--Class: PHYSICAL-TYPE-->
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="PRECISION"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="PHYSICAL-DATA-TYPE" name="BASE-DATA-TYPE"/>
        <xsd:attribute use="optional" type="RADIX" name="DISPLAY-RADIX"/>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="PHYSICAL-VEHICLE-LINK">
        <!--Class: PHYSICAL-VEHICLE-LINK-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element type="VEHICLE-CONNECTOR-PIN-REFS" name="VEHICLE-CONNECTOR-PIN-REFS" minOccurs="1"
maxOccurs="1"/>
            <xsd:element type="LINK-COMPARAM-REFS" name="LINK-COMPARAM-REFS" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
        <xsd:attribute use="required" type="xsd:string" name="TYPE"/>
    </xsd:complexType>
    <xsd:complexType name="PHYSICAL-VEHICLE-LINKS">
        <!-- Automatically generated wrapper element type -->
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" type="PHYSICAL-VEHICLE-LINK" name="PHYSICAL-VEHICLE-
LINK"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="false" name="PHYS-MEM">
        <!--Class: PHYS-MEM-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element type="PHYS-SEGMENTS" name="PHYS-SEGMENTS" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="PHYS-SEGMENT">
        <!--Class: PHYS-SEGMENT-->
        <xsd:sequence>
            <xsd:group maxOccurs="1" minOccurs="1" ref="ELEMENT-ID"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:hexBinary" name="FILLBYTE"/>
            <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="BLOCK-SIZE"/>
            <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="START-ADDRESS"/>
        </xsd:sequence>
        <xsd:attribute use="required" type="xsd:ID" name="ID"/>
        <xsd:attribute use="optional" type="xsd:string" name="OID"/>
```

```
</xsd:complexType>
<xsd:complexType name="PHYS-SEGMENTS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" type="PHYS-SEGMENT" name="PHYS-SEGMENT"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="PIN-TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="HI"/>
        <xsd:enumeration value="LOW"/>
        <xsd:enumeration value="K"/>
        <xsd:enumeration value="L"/>
        <xsd:enumeration value="TX"/>
        <xsd:enumeration value="RX"/>
        <xsd:enumeration value="PLUS"/>
        <xsd:enumeration value="MINUS"/>
        <xsd:enumeration value="SINGLE"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType abstract="true" name="POSITIONABLE-PARAM">
    <!--Class: POSITIONABLE-PARAM-->
    <xsd:complexContent>
        <xsd:extension base="PARAM">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="0" type="xsd:unsignedInt" name="BYTE-POSITION"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="BIT-POSITION">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:unsignedInt">
                            <xsd:maxExclusive value="8"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="POS-OFFSET">
    <!--Class: POS-OFFSET-->
    <xsd:complexContent>
        <xsd:extension base="TARGET-ADDR-OFFSET">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" type="xsd:hexBinary" name="POSITIVE-OFFSET"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="false" name="POS-RESPONSE">
    <!--Class: POS-RESPONSE-->
    <xsd:complexContent>
        <xsd:extension base="RESPONSE"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="POS-RESPONSE-REFS">
    <!-- Automatically generated wrapper element type -->
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="POS-RESPONSE-REF" type="ODXLINK"/>
    </xsd:sequence>
```