

Road vehicles — Open diagnostic data exchange (ODX)

Part 1: Data model specification

ICS 43.180

National foreword

This British Standard is the UK implementation of ISO 22901-1:2008.

The UK participation in its preparation was entrusted to Technical Committee AUE/16, Electrical and electronic equipment.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 November 2008
© BSI 2008

ISBN 978 0 580 55358 5

Amendments/corrigenda issued since publication

Date	Comments

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)

AVA Documents
Document Sharing Hub
© ISO 2008

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO 2008

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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	v
Introduction.....	vi
1 Scope	1
2 Normative references	1
3 Abbreviated terms	2
4 ODX use cases.....	3
4.1 General	3
4.2 Use case 1: ODX process chain.....	3
4.3 Use case 2: Cross vehicle platform ECU diagnostic development.....	4
4.4 Use case 3: Franchise and aftermarket service dealership diagnostic tool support.....	5
4.5 Architecture of a Modular VCI compliant D-server	6
4.6 ODX benefit examples.....	6
5 Specification release version information	8
5.1 Specification release version location	8
5.2 Specification release version	8
6 Introduction to and use of Unified Modelling Language (UML).....	8
6.1 General aspects.....	8
6.2 Class diagrams	8
6.3 Mapping to XML.....	12
7 ODX data model	14
7.1 General modelling principles	14
7.2 ODX package	26
7.3 ODX data model for diagnostics.....	29
7.4 Usage scenarios (diagnostic).....	183
7.5 ODX data model for ECU memory programming.....	229
7.6 ECU programming usage scenarios (flash).....	253
7.7 ECU variant coding usage scenarios	265
7.8 ODX data model for ECU configuration	266
7.9 Function dictionary	276
8 Data model implementation in XML.....	287
8.1 Classifier.....	287
8.2 Relationships	295
9 Packaged ODX data (PDX).....	304
9.1 Overview.....	304
9.2 Structure of PDX package	305
9.3 Usage scenarios	308
Annex A (normative) Enumerations and pre-defined values	315
Annex B (normative) ODX checker rules.....	326
Annex C (normative) XML schema.....	345
Annex D (informative) User-defined formats for flashdata.....	420
Annex E (informative) Coherent examples for diagnostic services	424
Annex F (informative) ECU-MEM example.....	464
Annex G (informative) Session security example.....	472

Bibliography	485
---------------------------	------------

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.

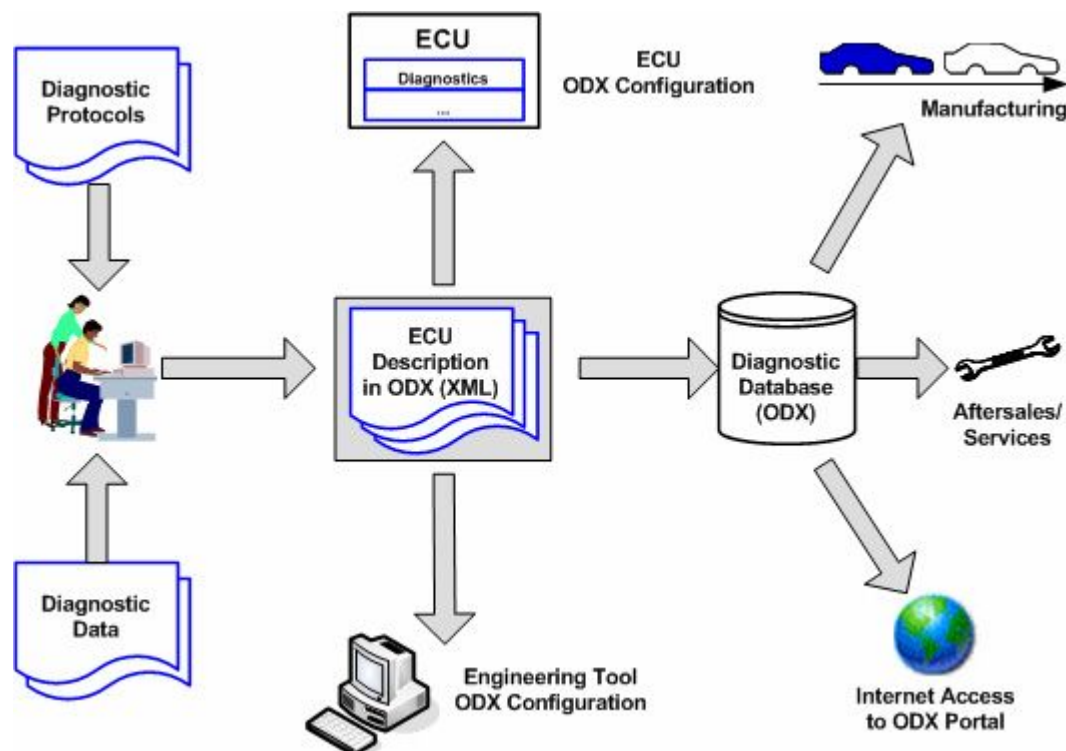


Figure 1 — Usage of ODX data in the ECU life cycle

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.

- 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.
- 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.
- 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.

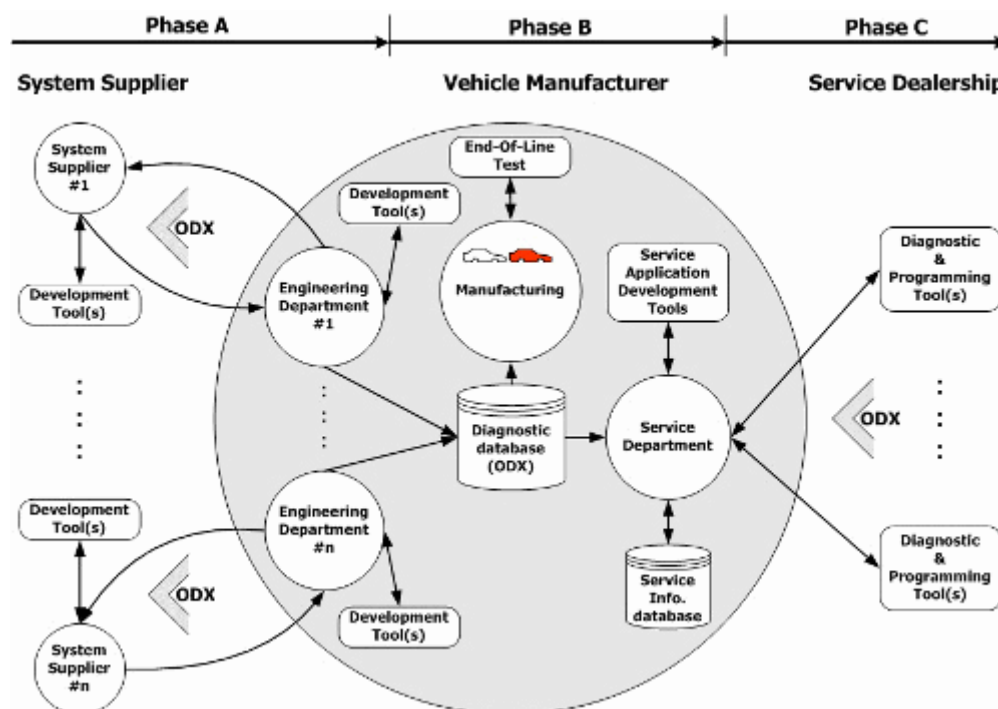


Figure 2 — Example of ODX process chain

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.

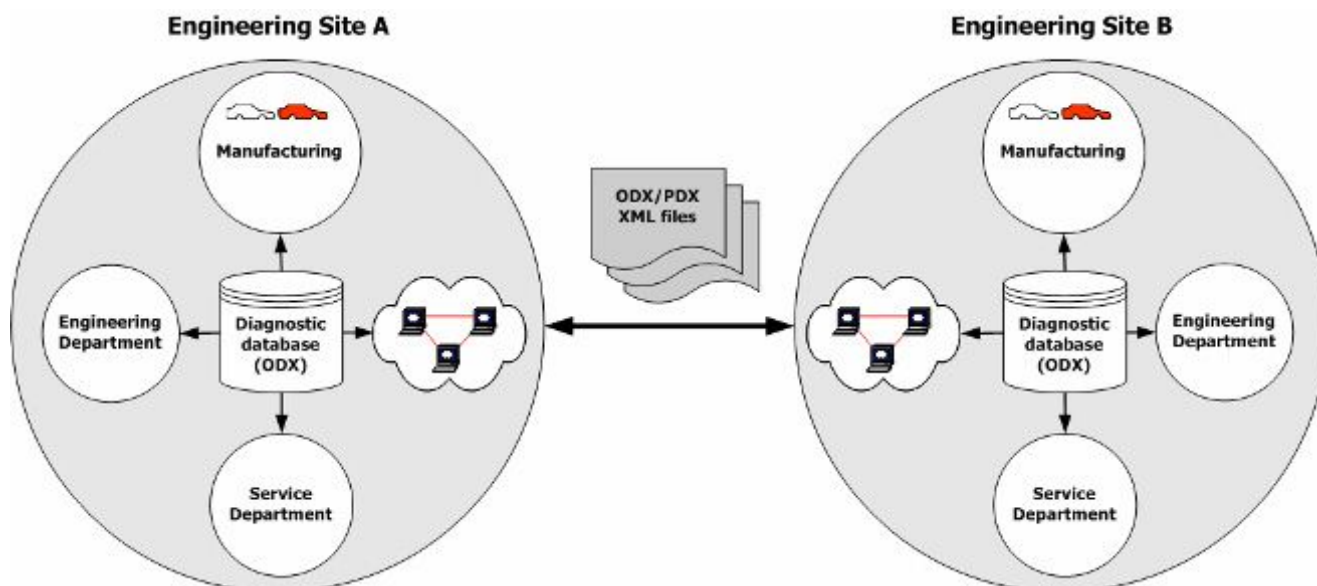


Figure 3 — Example of cross vehicle platform ECU diagnostic development

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.

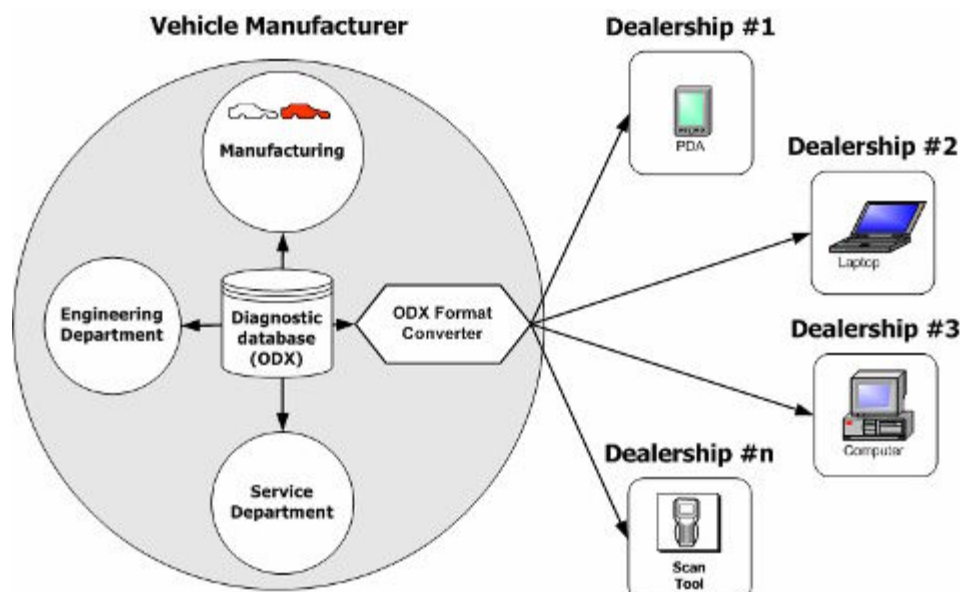


Figure 4 — Example of automotive dealership diagnostic tool support

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.

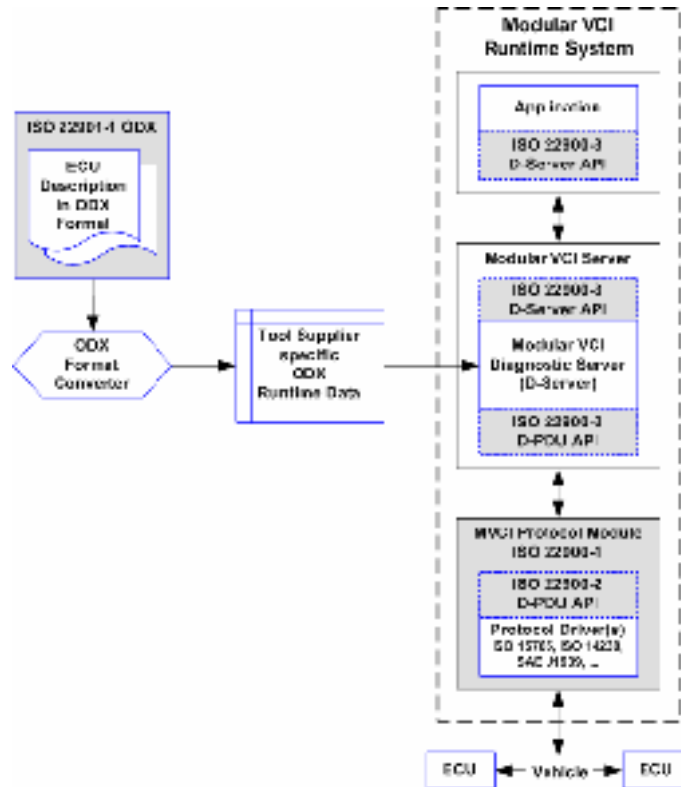


Figure 5 — Architecture of a Modular VCI compliant D-server

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.

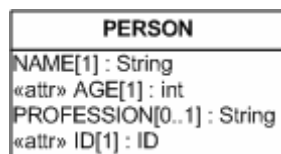


Figure 6 — UML representation of class

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 implemented 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.

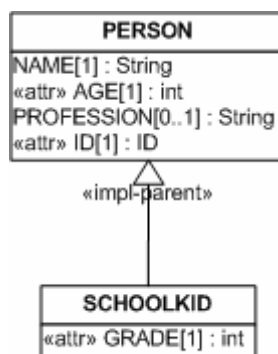


Figure 7 — UML representation of inheritance relationship

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 deleted¹⁾.

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 used. 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).

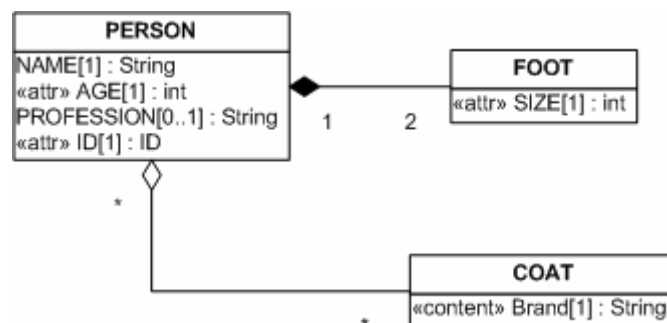


Figure 8 — UML representation of composition and aggregation relationships

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.

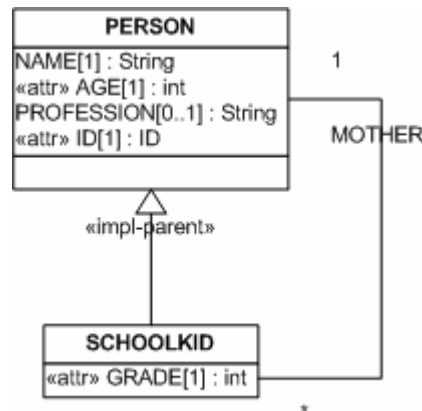


Figure 9 — UML representation of association relationship

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.

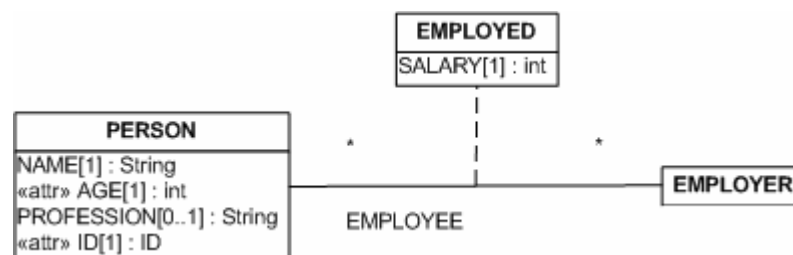


Figure 10 — UML representation of association class

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: EMPLOYER 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.

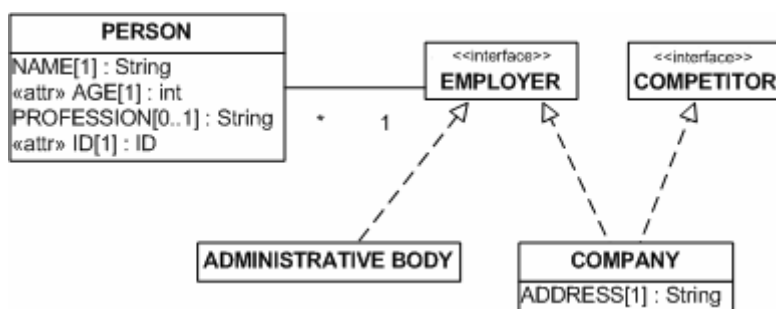


Figure 11 — UML representation of interfaces

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.

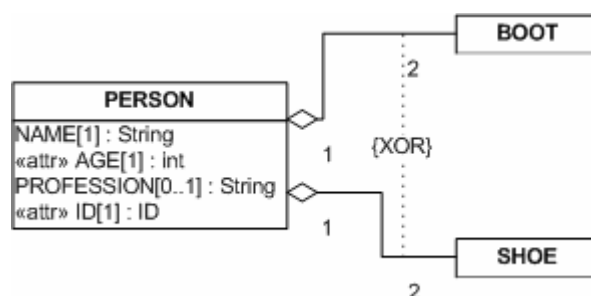


Figure 12 — UML representation of constraint

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.

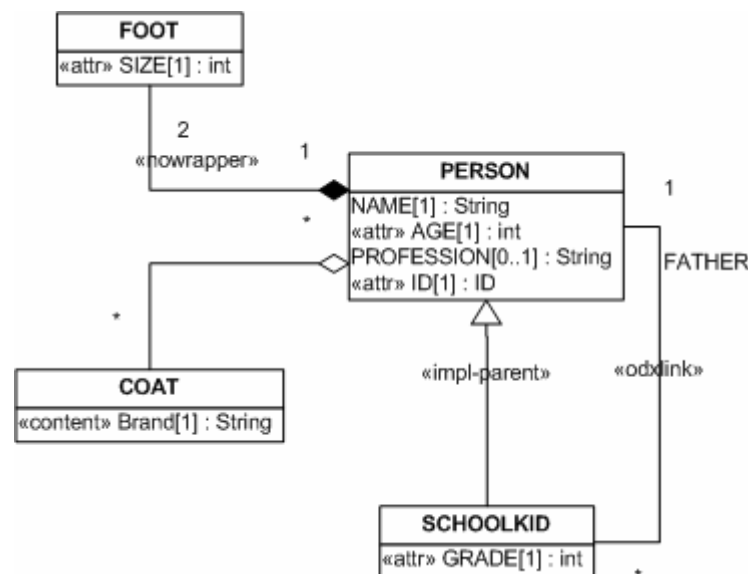


Figure 13 — UML representation of mapping example

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>, , <u>, <sub>, <sup>, , , .

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.

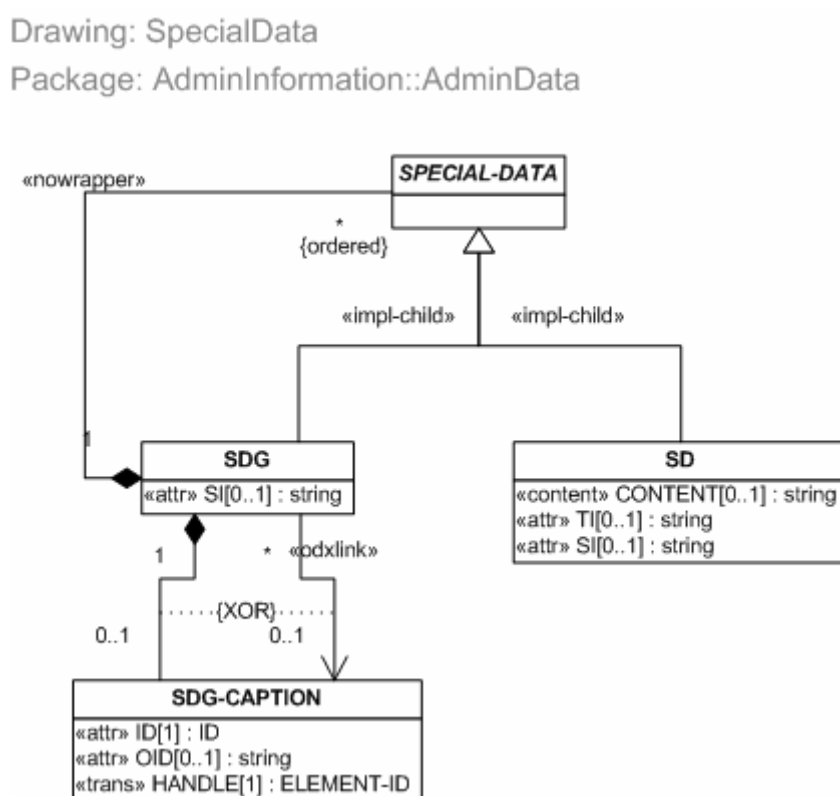


Figure 14 — UML representation of special data group (SDG)

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.

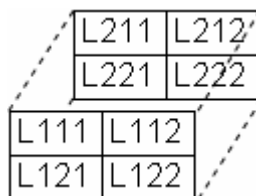


Figure 15 — Special data group

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="ilrow1">
          <SHORT-NAME>firstRow</SHORT-NAME>
        </SDG-CAPTION>
        <SD SI="example">L111</SD>
        <SD SI="example">L112</SD>
      </SDG>
      <SDG>
        <SDG-CAPTION ID="ilrow2">
          <SHORT-NAME>secondRow</SHORT-NAME>
        </SDG-CAPTION>
        <SD SI="example">L121</SD>
        <SD SI="example">L122</SD>
      </SDG>
    </SDG>
  </SDG>
</SDGS>
```

```

        <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>
</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.

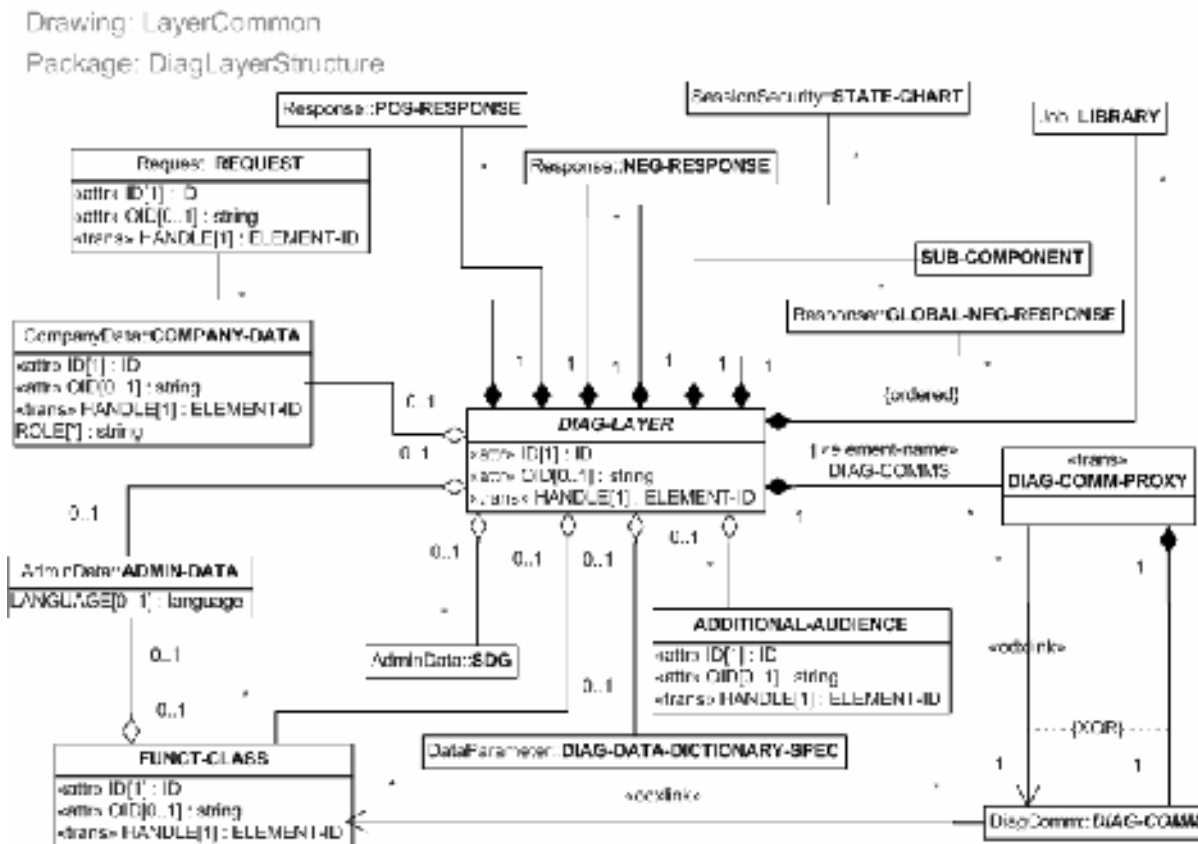


Figure 16 — UML representation of layer common – additional audience

EXAMPLE ADDITONAL-AUDIENCES:

```
<ADDITIONAL-AUDIENCES>
  <ADDITIONAL-AUDIENCE ID="IS-SUPPLIER.ID">
    <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 ADDITIONAL-AUDIENCE for the detailed modelling in UML.

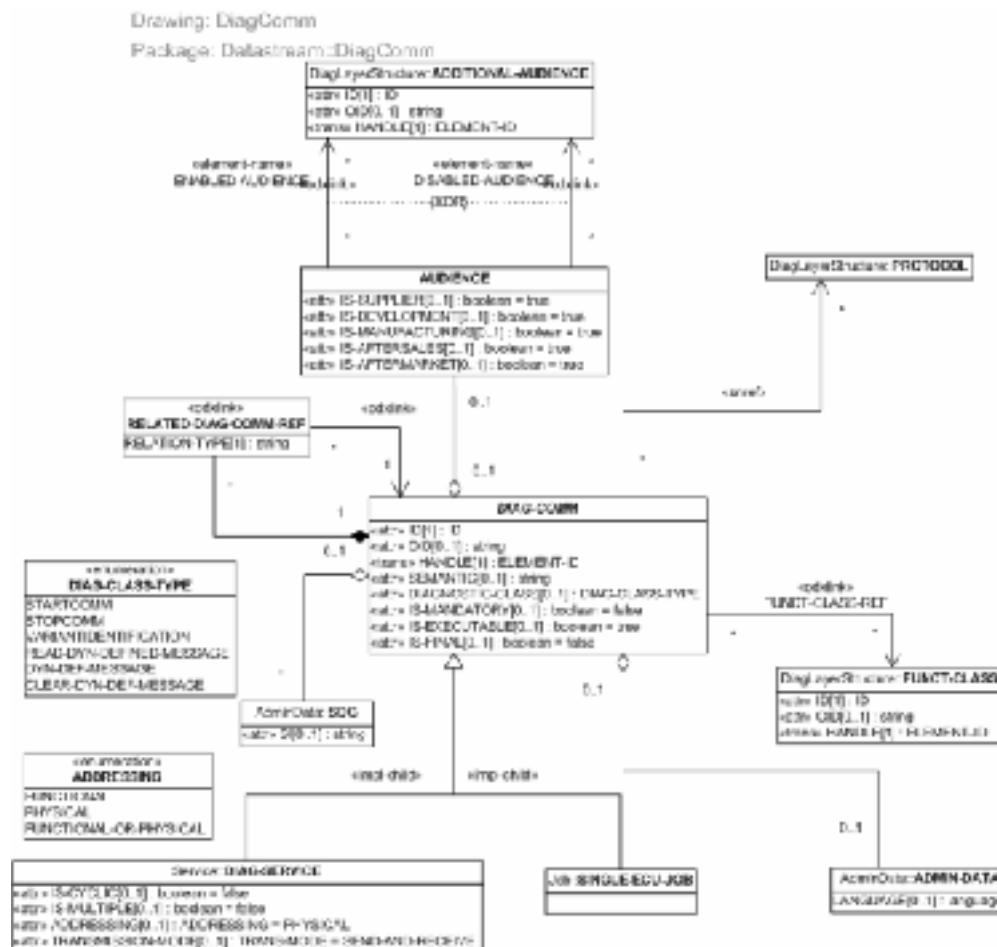


Figure 17 — UML representation of DIAG-COMM and ADDITIONAL-AUDIENCE

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.

Drawing: AdminData

Package: AdminInformation::AdminData

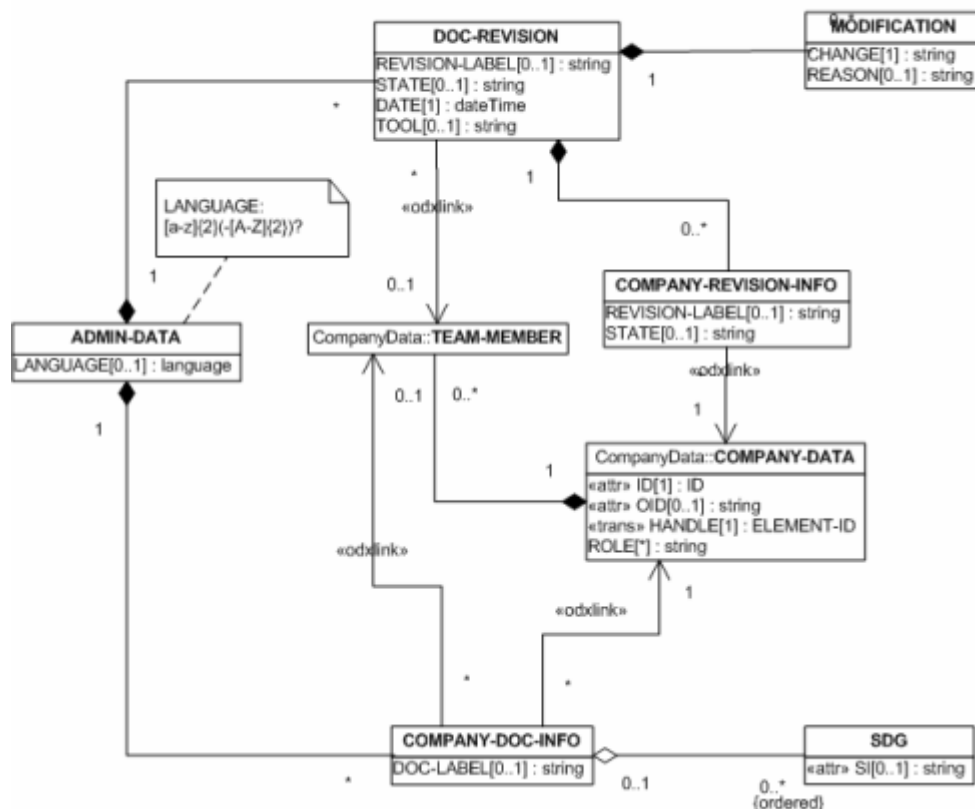


Figure 18 — UML representation of admin data

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.

Drawing: AdminInfo

Package: AdminInformation

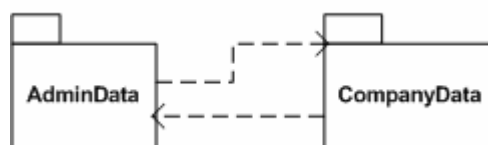


Figure 19 — Admin info

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.

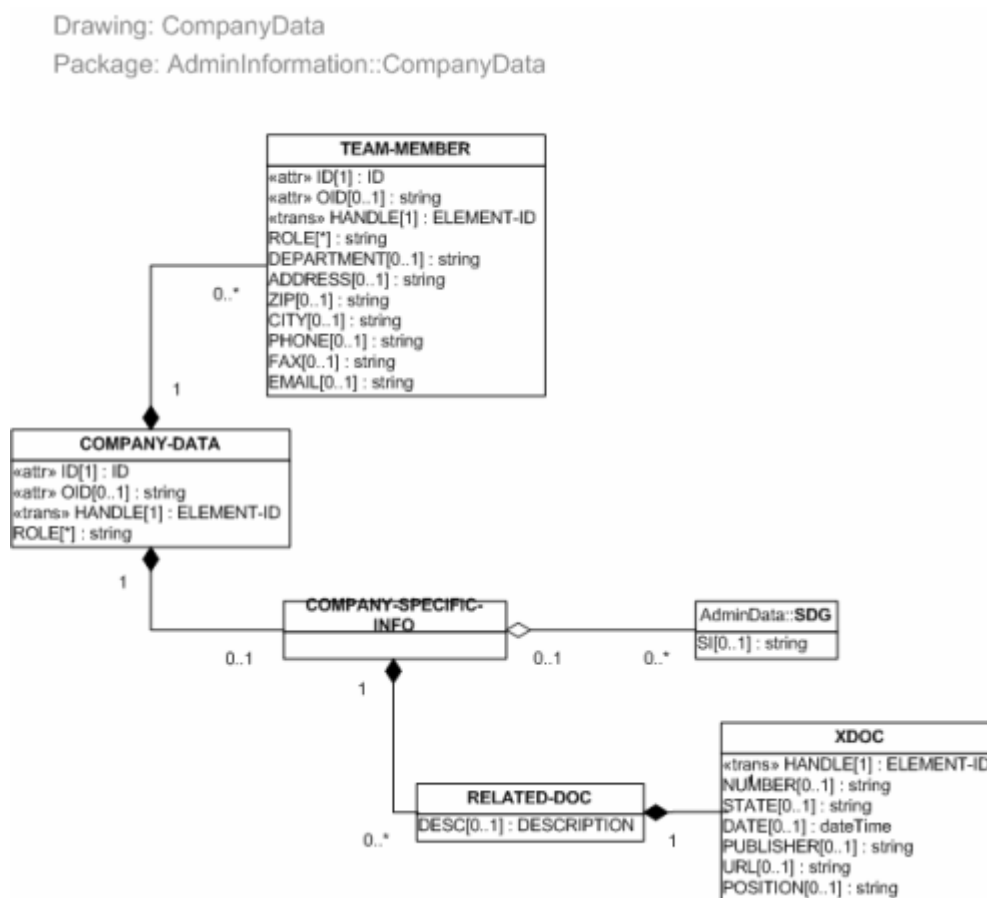


Figure 20 — UML representation of company data

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="JD0"/>
      <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-REVISIONS>

```

```
<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.

Drawing: PackageOverview
Package: Top



Figure 21 — ODX package overview

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

- **DiagLayerStructure, CompParam, 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.

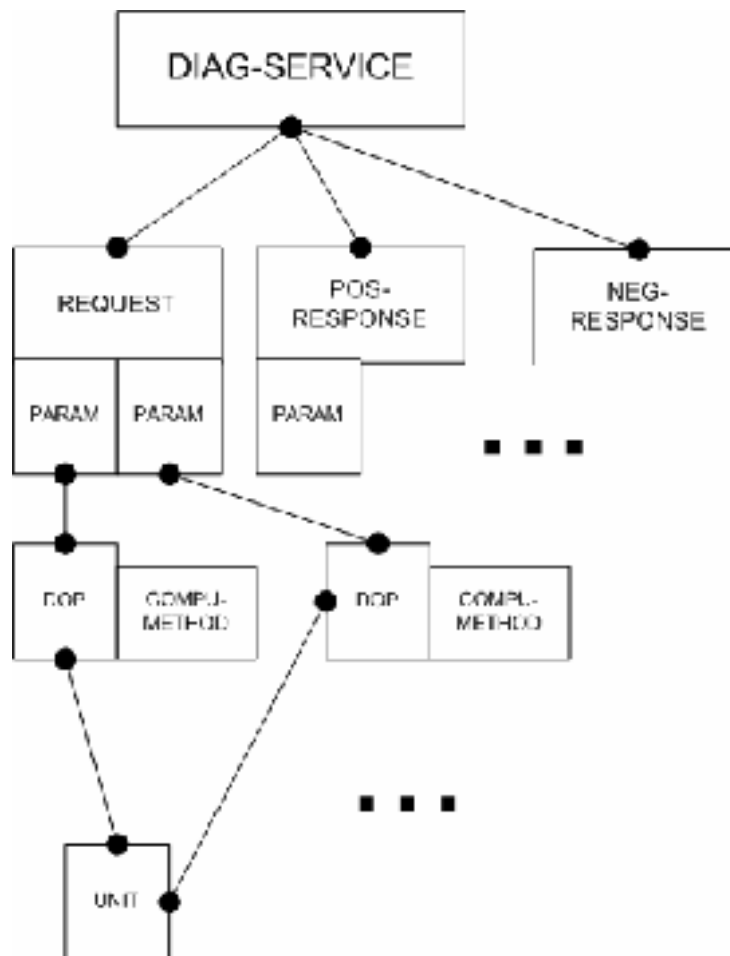


Figure 23 — Object chain in ODX DiagLayerStructure

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).

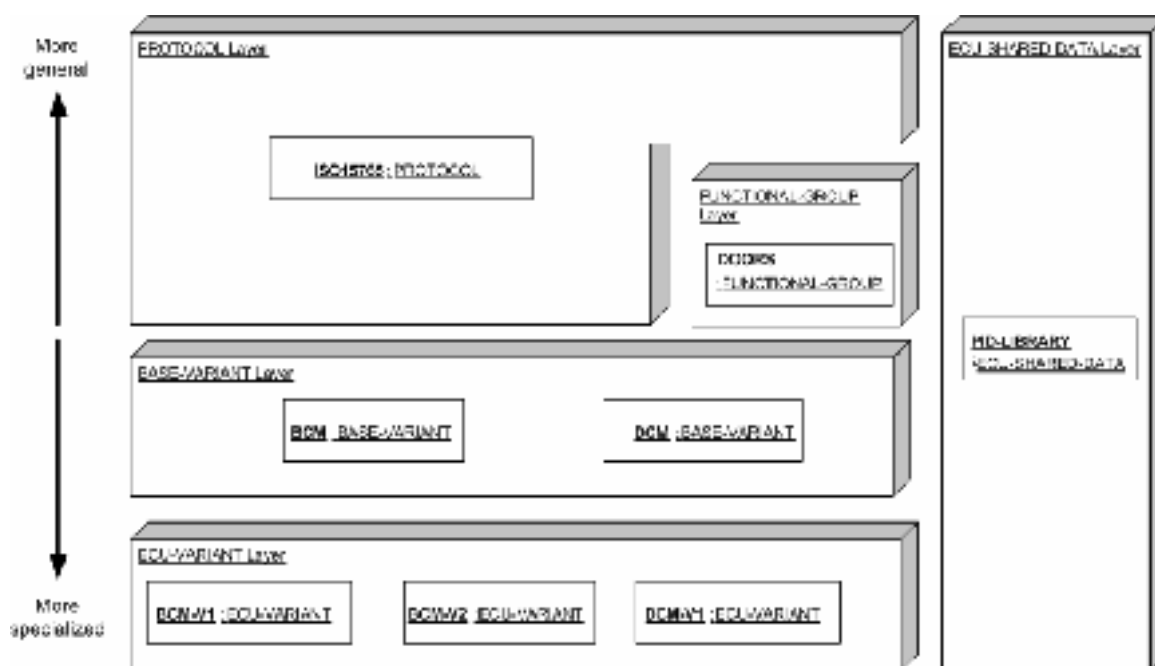


Figure 24 — Diagnostic layers overview example

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.

Drawing: DiagLayers

Package: DiagLayerStructure

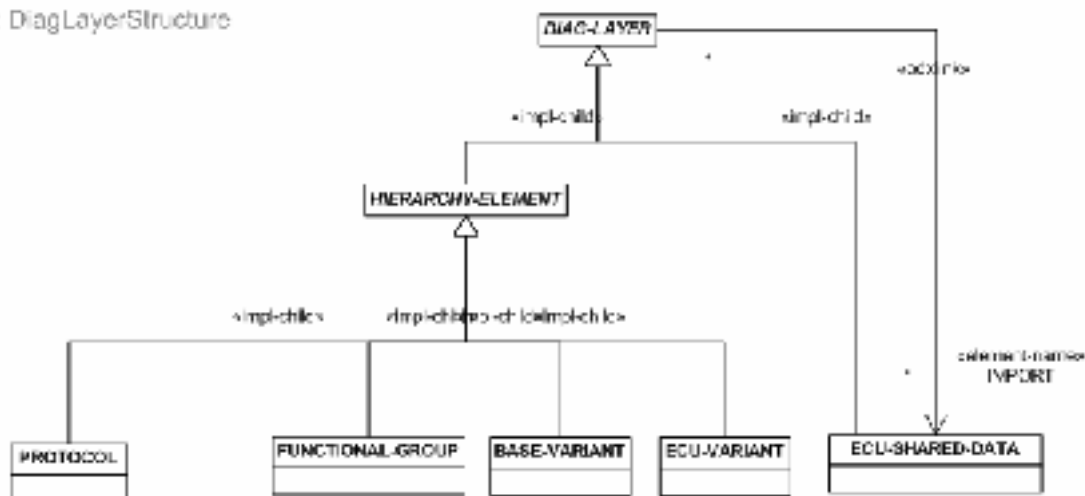


Figure 25 — UML representation of diagnostic layers

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.

Drawing: LayerCommon

Package: DiagLayerStructure

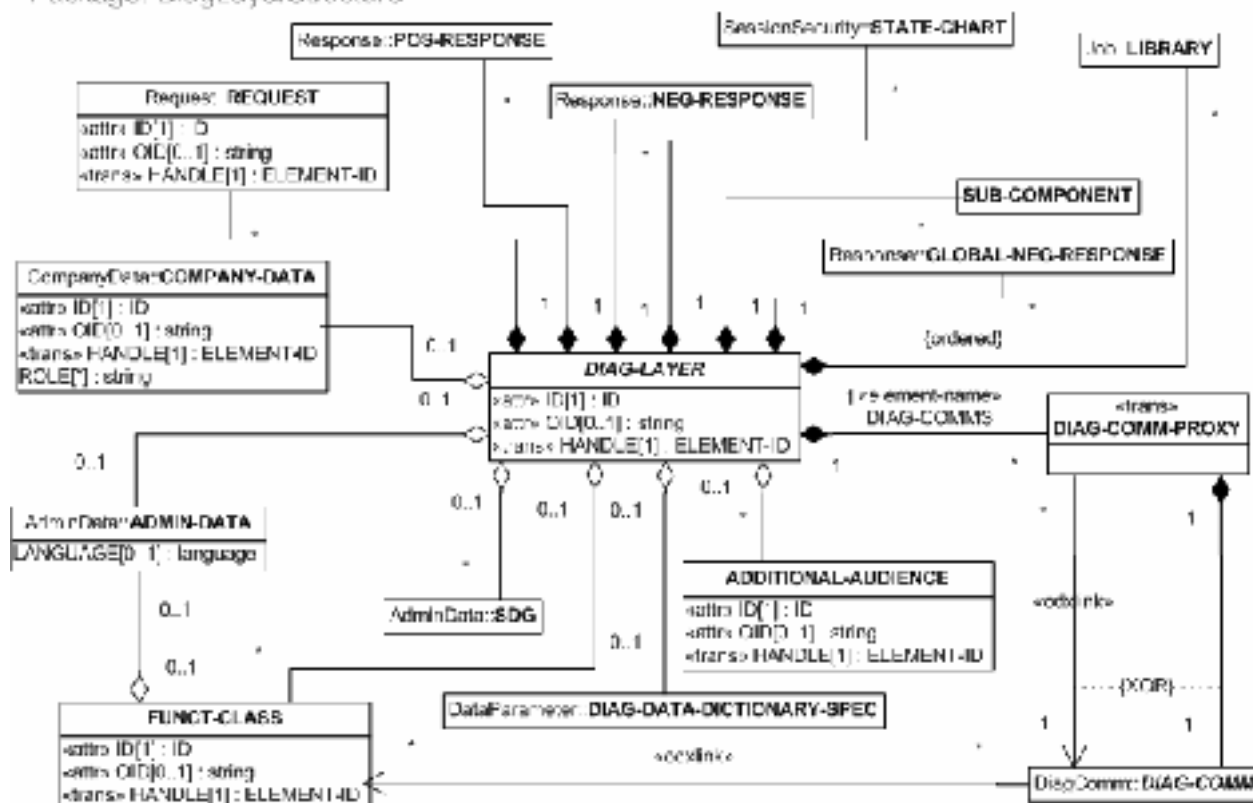


Figure 26 — UML representation of diagnostic layer commonalities

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.

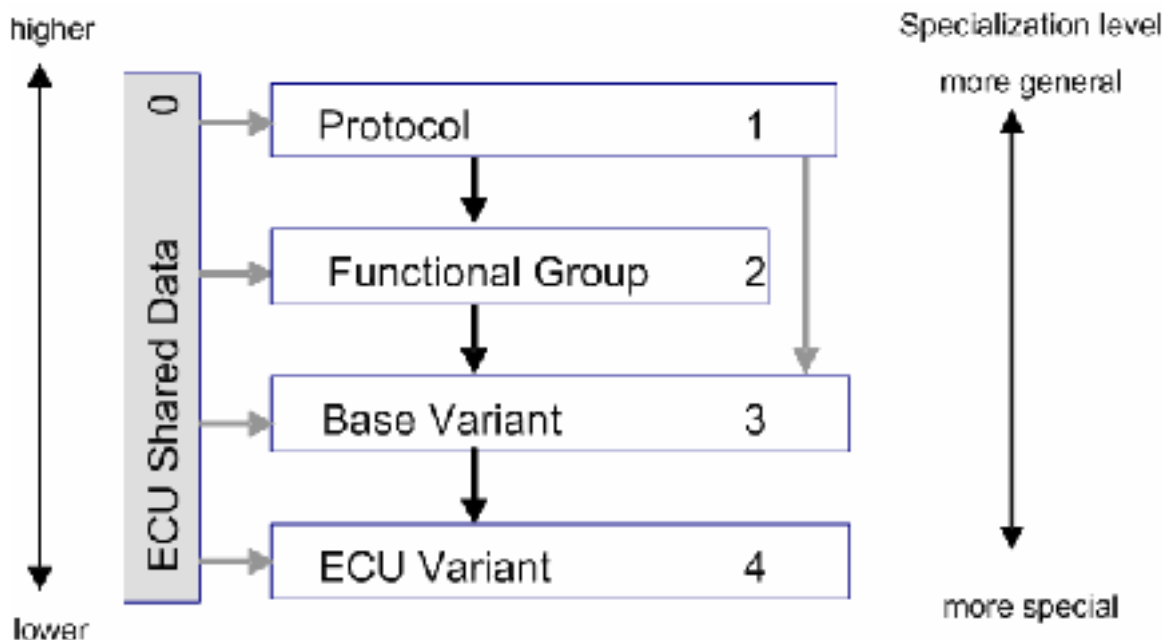


Figure 27 — Diagnostic layer hierarchy

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.

Drawing: LayerHierarchy
Package: DiagLayerStructure

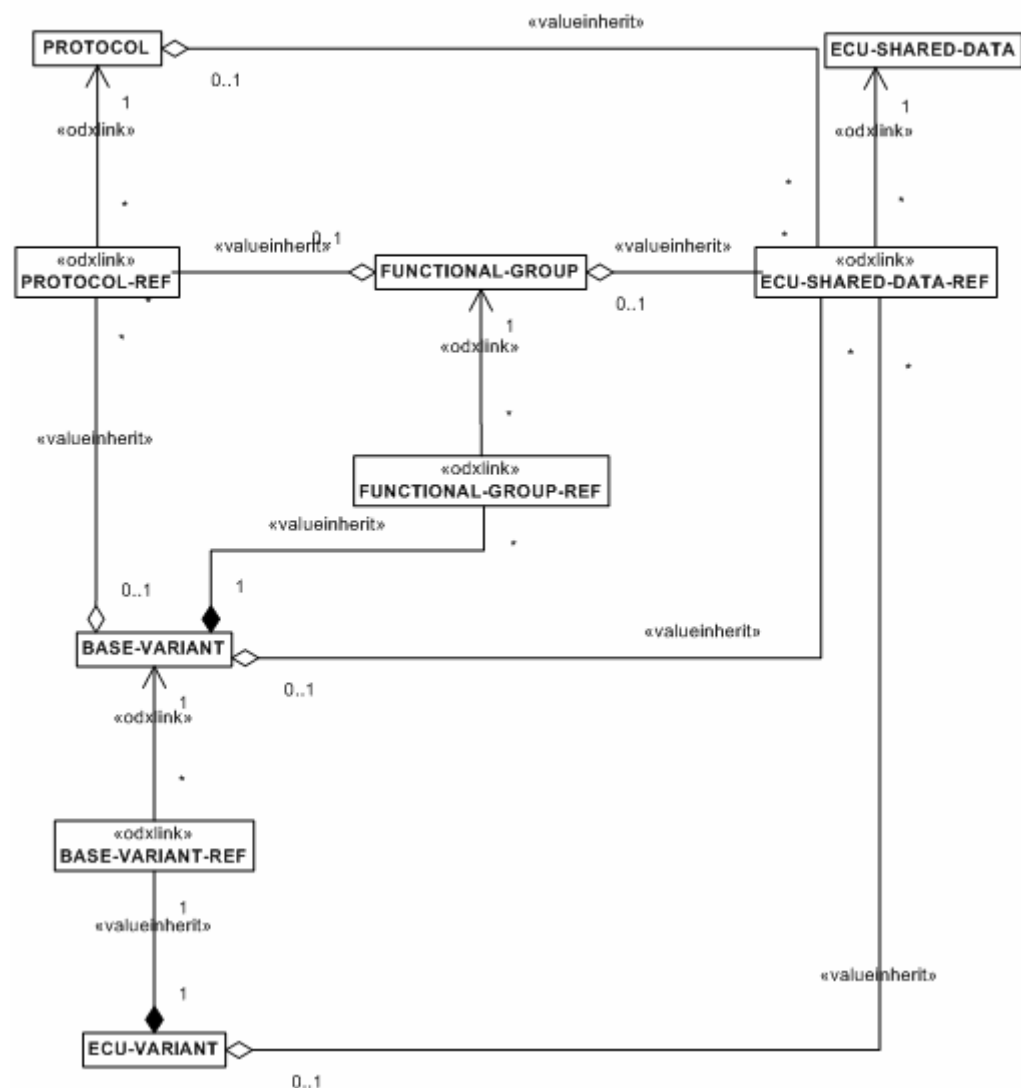


Figure 28 — UML representation of the diagnostic layer hierarchy

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.

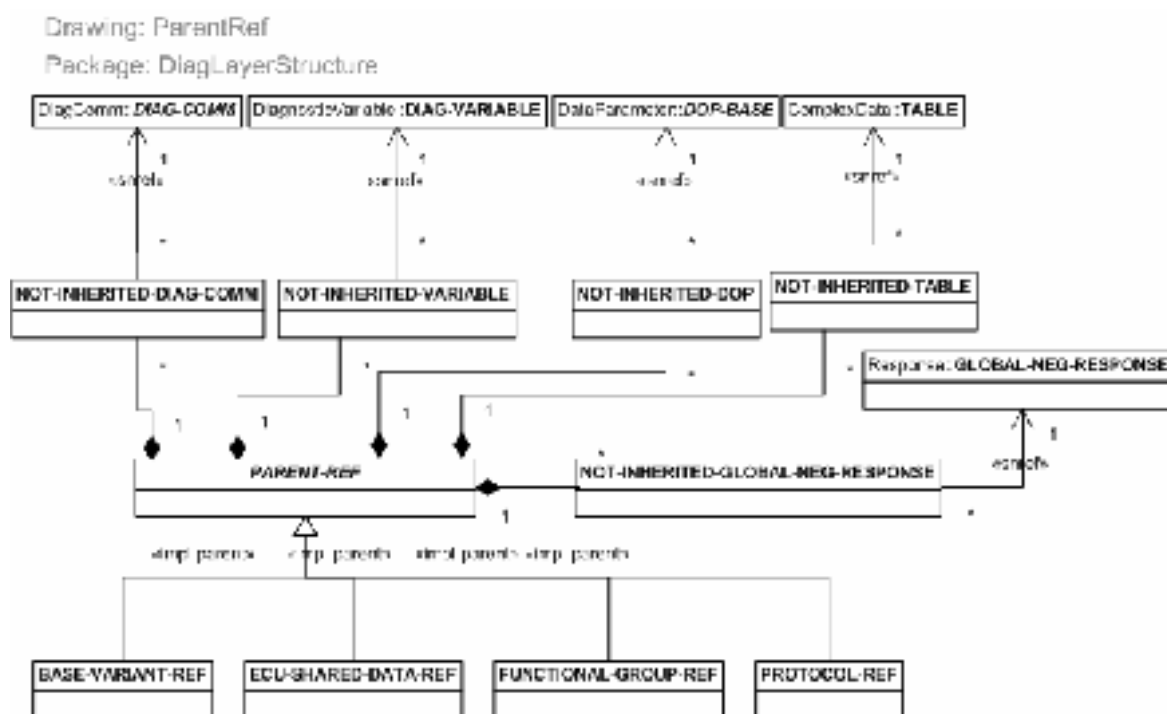


Figure 29 — UML representation of parent reference

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 mandatory²⁾.
- 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.

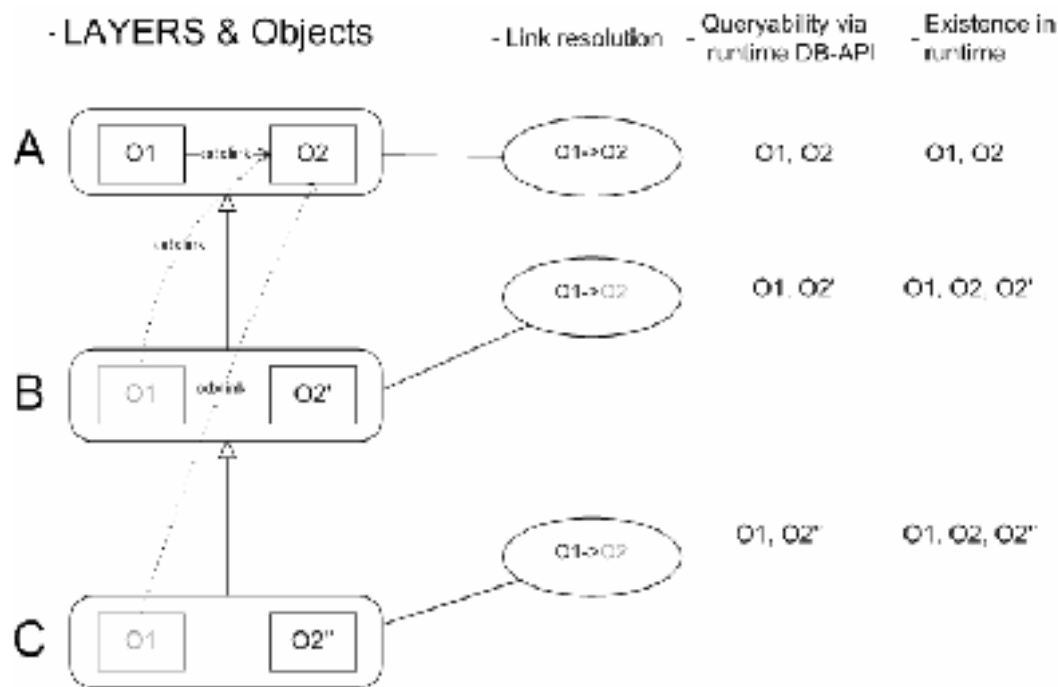


Figure 31 — Inheritance and odxlink referencing

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).

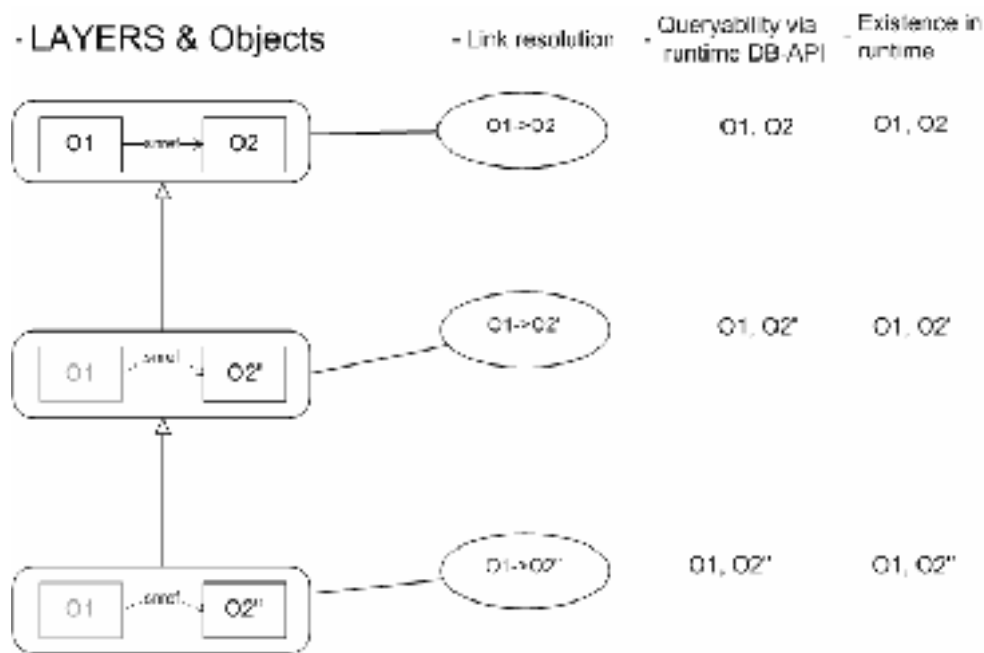


Figure 32 — Inheritance and SNREFs

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.

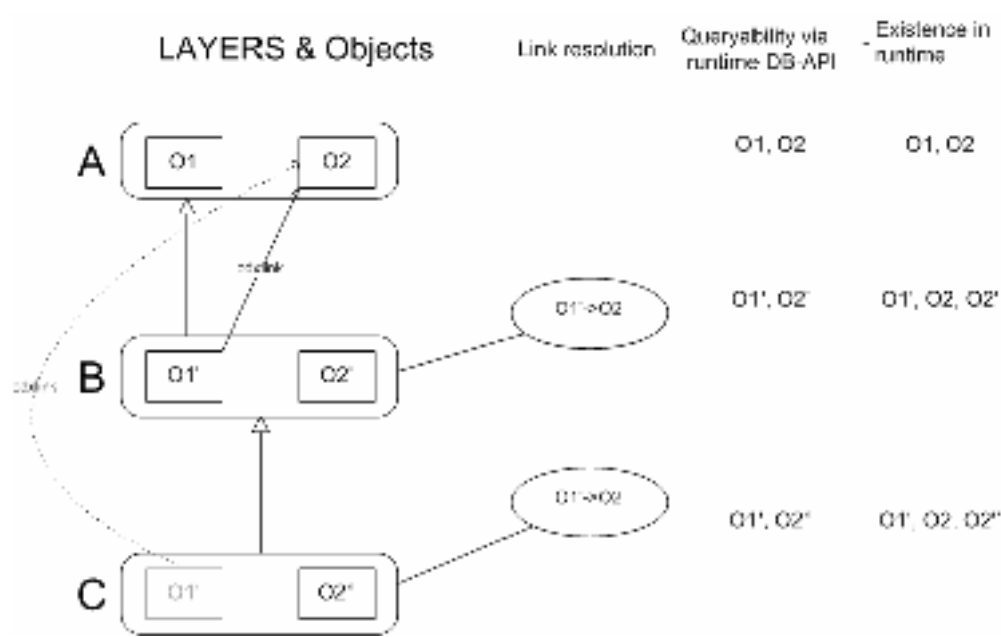


Figure 33 — Visibility of overridden objects

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.

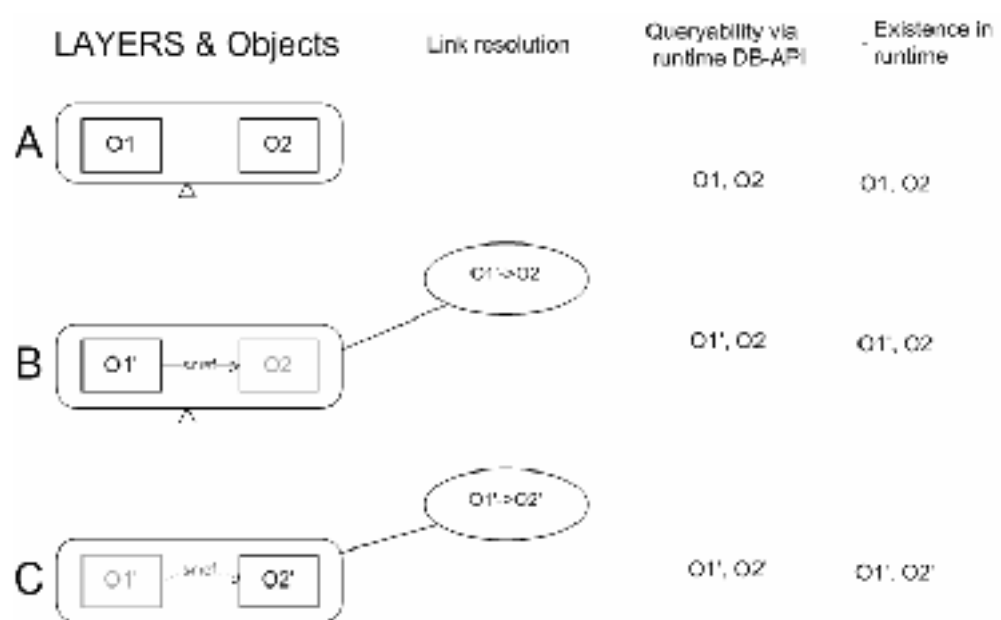


Figure 34 — Overriding objects referenced by SNREFs