INTERNATIONAL STANDARD

ISO 22900-3

Second edition
2012-12-01

Road vehicles — Modular vehicle communication interface (MVCI) —

Part 3: Diagnostic server application programming interface (D-Server API)

Véhicules routiers — Interface de communication modulaire du véhicule (MVCI) —

Partie 3: Interface pour la programmation des applications du serveur de diagnostic (D-Server API)

Reference number
ISO 22900-3:2012(E)

Contents

Page

Foreword

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

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

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

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

ISO 22900-3 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

This second edition cancels and replaces the first edition (ISO 22900-3:2009), which has been technically revised.

ISO 22900 consists of the following parts, under the general title *Road vehicles — Modular vehicle communication interface (MVCI)*:

— *Part 1: Hardware design requirements*

— *Part 2: Diagnostic protocol data unit application programming interface (D-PDU API)*

— *Part 3: Diagnostic server application programming interface (D-Server API)*

Introduction

0.1 Overview

This part of ISO 22900 has been established in order to define a universal application programmer interface of a vehicle communication server application. Today's situation in the automotive market requires different vehicle communication interfaces for different vehicle OEMs supporting multiple communication protocols. However, until today, many vehicle communication interfaces are incompatible with regard to interoperability with multiple communication applications and vehicle communication protocols.

Implementation of the MVCI diagnostic server concept supports overall cost reduction to the end user because, for example, a single diagnostic or programming application will support many vehicle communication interfaces supporting different communication protocols and different vehicle communication modules of different vendors at one time.

A vehicle communication application compliant with this part of ISO 22900 supports a protocol independent D-PDU API (Protocol Data Unit Application Programming Interface) as specified in ISO 22900-2. The server application will need to be configured with vehicle- and ECU-specific information. This is accomplished by supporting the ODX data format (Open Diagnostic Exchange format) as specified in ISO 22901-1.

A server compliant with this part of ISO 22900 supports the function block Diagnostics (D). A compliant server also supports Job-Language (Java) and may support optional features like ECU (re)programming. The defined object-oriented API provides for a simple, time saving and efficient interchangeability of different servers.

The client application and the communication server do not necessarily need to run on the same computer. A remote use via an interface may also be envisaged and is supported by the design of the server API. This interface is provided for ASAM GDI, COM/DCOM [10] [Technology Reference COM-IDL], for C++ [11] [Technology Reference C++] and for Java [12] [Technology Reference Java].

0.2 ASAM e.V. implementation reference documents

This part of ISO 22900 references several ASAM e.V. documents which contain the Technology Reference Mapping Rules for COM-IDL, C++ and Java.

The following ASAM documents are relevant for the implementation of this part of ISO 22900:

— ASAM Technology Reference COM-IDL, *COM-IDL Technology Reference Mapping Rules* [10]:
this document describes the platform, programming language and linking mechanisms for the implementation of the generic object model in COM-IDL.

— ASAM Technology Reference C++, *C++ Technology Reference Mapping Rules* [11]:
this document describes the platform, programming language and linking mechanisms for the implementation of the generic object model in C++.

— ASAM Technology Reference Java, *Java Technology Reference Mapping Rules* [12]:
this document describes the platform, programming language and linking mechanisms for the implementation of the generic object model in Java.

Road vehicles — Modular vehicle communication interface (MVCI) —

Part 3: Diagnostic server application programming interface (D-Server API)

1 Scope

This part of ISO 22900 focuses on the description of an object-oriented programming interface. The objective is the ability to implement server applications, used during the design, production and maintenance phase of a vehicle communication system, compatible to each other and thus exchangeable. From a user's perspective, access and integration of on-board control units is provided by a corresponding application, the communication server and a VCI module for diagnostics. The user is granted access for the handling of control units (ECUs) in vehicles for the diagnostic services.

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 14229 (all parts), *Road vehicles — Unified diagnostic services (UDS)*

ISO 14230-3, *Road vehicles — Diagnostic systems — Keyword protocol 2000*

ISO 15765 (all parts), *Road vehicles — Diagnostic communication over Controller Area Network (DoCAN)*

ISO 22901-1, *Road vehicles — Open diagnostic data exchange (ODX) — Part 1: Data model specification*

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

3 Terms, definitions, symbols and abbreviated terms

3.1 Terms and definitions

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

3.1.1
AccessKey
path identifier through the inheritance hierarchy as defined in ISO 22901-1 ODX to a diagnostic data element

3.1.2
ancestor object
parent object
located above in the object hierarchy with respect to a given object

3.1.3
descendant object
child object
object, located below in the object hierarchy with respect to a given object

3.1.4
FlashJob
new class derived from MCDJob which is used to start FlashSessions within the MVCI diagnostic server

NOTE This information is provided by the databases. At the runtime object it is possible to set the FlashSession that has to be flashed by this service. Only one session can be set for one job. The application can access the priority defined in the database for every FlashSession and sort the sessions according to this priority.

The job interface of flash jobs (MCDFlashJob) extends the job interface of normal diagnostic jobs (MCDSingleECUJob) by a session object, i.e. its method prototype is extended as follows:

```
JobName(...,MCDDbFlashSession session)
```

3.1.5
FlashKey
unique identification for a flash session

3.1.6
FlashSessionClass
user-defined collection of FlashSessions, which can be used to separate FlashSessions for different tasks (e.g. sessions for data, sessions for boot, or sessions for code and data)

3.1.7
FlashSession
smallest unit that can be flashed separately by the MVCI diagnostic server, and which may consist of several data blocks

3.1.8
functional class
set of diagnostic services

3.1.9
function dictionary
hierarchical function catalog to organize external test equipment user interfaces (available at MCDDbProject):

— references to one or several ECUs and their diagnostic data content relevant for that function;

— references to services/jobs to make functions “executable”;

— definition of function input and output parameters with optional references to parameters of related services

3.1.10
interface connector
connector at the vehicle’s end of the interface cable between the vehicle and the communication device

3.1.11
job
sequence of diagnostic services and other jobs with a control flow

3.1.12
location
set of diagnostic data valid on a given hierarchical level of inheritance according to ISO 22901-1 ODX

NOTE The following locations exist:

— Multiple ECU Job,

— Protocol,

— Functional Group,

— ECU Base Variant,

— ECU Variant.

3.1.13
Logical Link
set of data, identifying the physical line, the interface and protocol used for an ECU

3.1.14
physical interface link
physical connection between the VCI connector of a VCI and the interface connector

3.1.15
physical link
physical vehicle link connected to a physical interface link, so it is the connection from the interface of the diagnostic server to the ECU in the vehicle

3.1.16
physical vehicle link
unique bus system in a vehicle, so it is the connection between the vehicle connector and the ECU

3.1.17
priority
term used by test systems to decide in which order the sessions have to be flashed

3.1.18
project
pool of diagnostic data

NOTE References between such data are resolvable inside this same project.

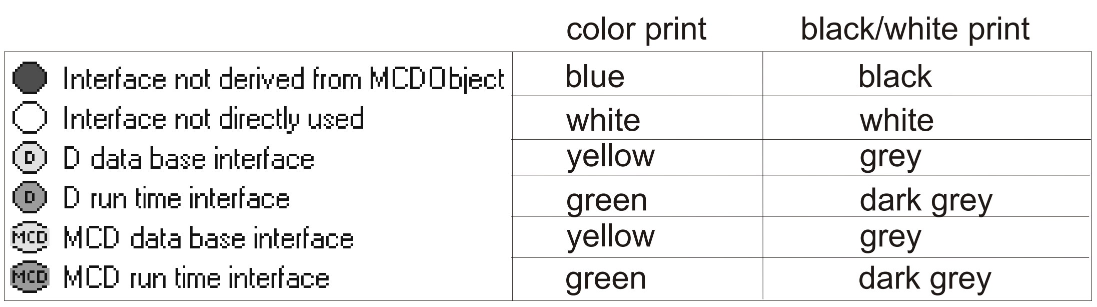
3.1.19
sub component
ECU sub functionality or components

EXAMPLE LIN-slaves (available at `MCDDbLocation`).

3.1.20
vehicle connector
connector on a vehicle providing access to the bus systems in the vehicle

3.2 Symbols

Figure 1 shows the legend of hierarchical models.



	color print	black/white print
● Interface not derived from MCDObject	blue	black
○ Interface not directly used	white	white
Ⓓ D data base interface	yellow	grey
Ⓓ D run time interface	green	dark grey
MCD MCD data base interface	yellow	grey
MCD MCD run time interface	green	dark grey

Figure 1 — Legend of hierarchical models

3.3 Abbreviated terms

API	Application Programmers Interface
ASAM	Association for Standardisation of Automation and Measuring Systems
ASCII	American Standard for Character Information Interchange
AUSY	AUtomation SYstem
CAN	Controller Area Network
COM/DCOM	Distributed Component Object Model
CORBA	Common Object Request Broker Architecture
CRC	Cyclic Redundancy Check
D	Diagnostics
Diag	Diagnostic
DLL	Dynamic Link Library
DoCAN	Diagnostic communication over CAN
DOP	diagnostic Data Object Property
DoIP	Diagnostic Over Internet Protocol
DTC	Diagnostic Trouble Code
DTD	Document Type Definition
DynID	Dynamically Defined Identifiers
ECU	Electronic Control Unit
ECU MEM	Electronic Control Unit MEMory
ERD	Entity Relationship Diagram
IDL	Interface Description Language

JAVA RMI	JAVA Remote Method Invocation
KWP	KeyWord Protocol
LIN	Local Interconnect Network
MCD	Measurement Calibration Diagnostic
MDF	Module Description File
MVCI	Modular Vehicle Communication Interface
ODX	Open Diagnostic data eXchange
OEM	Original Equipment Manufacturer
PC	Personal Computer
PDU	Protocol Data Unit
SDG	Special Data Groups
SI	Système International d'unités
UDS	Unified Diagnostic Services
UTC	Coordinated Universal Time
VI	Variant Identification
VIS	Variant Identification and Selection
VIT	VehicleInformationTable
XML	eXtended Markup Language

4 Conventions

4.1 General

This part of ISO 22900 is based on the conventions discussed in the OSI Service Conventions (ISO/IEC 10731:1994) as they apply for diagnostic services.

4.2 Typographical conventions and mnemonics

Normal text of the specification is presented like this.

Source code and technical artifacts within the text are presented like `this`.

Diagrams that denote interaction sequences, relationships or dependencies between interfaces are presented using the Unified Modeling Language's (UML) convention.

The name of each interface and each class defined by this part of ISO 22900 shall use the prefix of the stereotype, e.g. "D".

The leading letter of each method and each parameter is small.

The leading word of each method shall be a verb.

The letter "_" is not allowed in interface names, method names and parameter names, but it is allowed for constants.

The leading letter of each constant is "e" and behind this the name is written in capital letters.

ODX element names are written in upper cases, e.g. SHORT-NAME. MVCI diagnostic server Names are written in mixed fixed, e.g. MCDDbProject.

4.3 Sequence diagrams

With the help of Sequence Diagrams the interactive use of the API and the sequences for certain general cases are presented in chronological order.

The sequence diagrams are oriented according to the presentation in UML and are structured as follows. The chronological sequence arises while reading from top downwards. The commentary column, in which single activities are commented, is placed at the left margin. Within the sequence diagram the Client application is shown on the left; if necessary for the respective case, the EventHandler is shown there as well. The API objects necessary for the respective case are located to the right of the Client (with or without EventHandler). If necessary, the MVCI diagnostic server is presented at the right, outside.

Not all API objects possible for the respective instant of time are shown; instead, only those of relevance for the respective case are shown. The thin line leading down vertically from the objects represents the life line, the wider sections on it represent activities of the object.

The black horizontal arrows between the single objects, Client and MVCI diagnostic server represent the actions necessary for the respective case. The object to which the arrow points at will execute the action. The grey horizontal arrows represent the return of objects.

4.4 Stereotypes

Stereotypes are abbreviation characters which are used in MVCI diagnostic servers to mark the affiliation of statements, interfaces and methods to one of the possible parts.

Table 1 defines the stereotypes which are used in MVCI diagnostic servers.

Table 1 — Stereotypes

Stereotype	Usage of method and class is in following Function Blocks allowed
<<MCD>>	Measurement, Calibration and Diagnostic
<<D>>	Diagnosis
<<JD>>	Methods with this stereotype can only be used inside of Diagnostic Job. These methods are not available for use at the API.

5 Specification release version information

The specification release version of this part of ISO 22900 is: 3.0.0.

6 Structure of a MVCI diagnostic server

Each server is divided into the functional block "D" (Diagnostic) and the database.

Figure 2 shows the architecture of an MVCI diagnostic server.

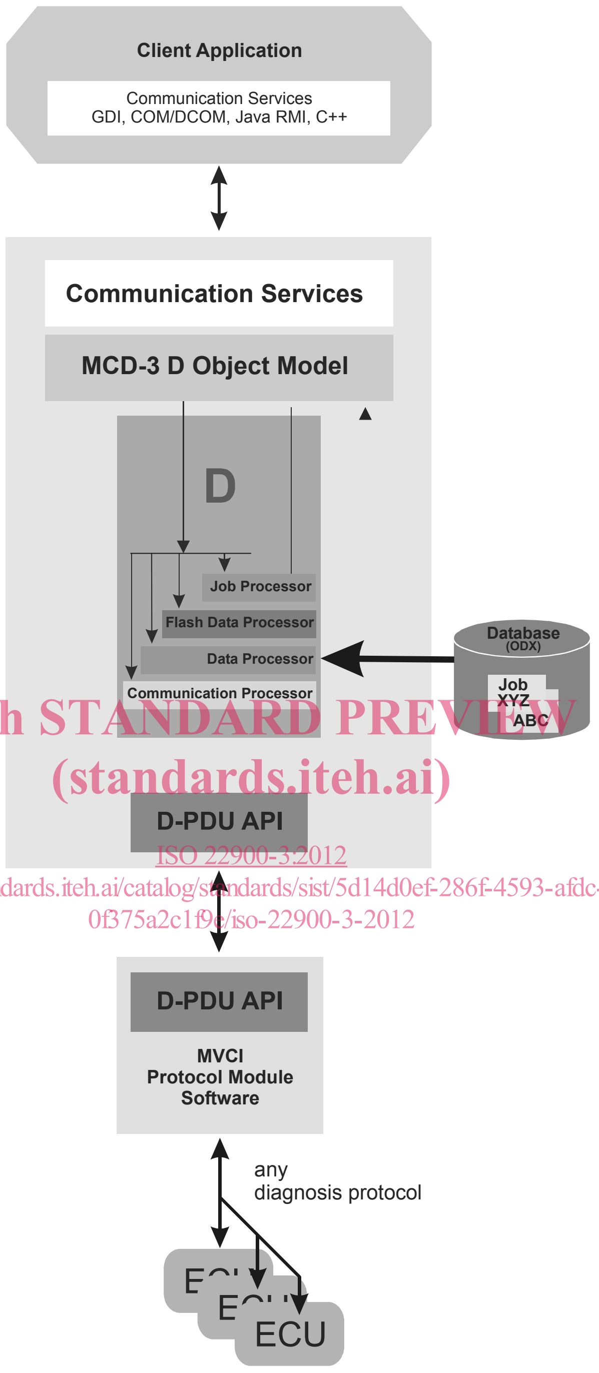


Figure 2 — Architecture of an MVCI diagnostic server

With the help of a server the control units are optimally adapted to the relevant requirements for their use in vehicles. This procedure is often referred to as “Applying”.

The following features (interfaces and methods) are optional:

— MCDDbProjectConfiguration,

— ECU Configuration,

— ECU Reprogramming (Flash),

— DynID,

— Monitoring,

— System properties,

— Function dictionary,

— SubComponents,

— Audiences,

— ECU state,

— Multiple ECU Jobs,

— PDU Time stamps,

— Library concept,

— DoIP,

— PDU API support,

— the concept of System generated Vehicle Information table.

Optional means that the runtime, as well as the database part of the model, do not have to be implemented by a diagnostic server that is omitting the feature in question. When a client application calls a method that is part of an optional feature, the diagnostic server should return an empty collection if the return type of the method is inheriting from MCDCollection. Otherwise, such a method call should throw an `MCDSystemException` of type `eSYSTEM_METHOD_NOT_SUPPORTED`. In the case of support of optional features these have to be implemented completely. An overview of methods which belong to optional functionalities can be found in Annex C.

The number of control units applied in vehicles is continuously increasing. The capabilities of the single control unit concerning diagnosis become available for the server by means of control unit description files (Data Description Interface). The control unit description files represent a manufacturer independent data exchange format, which means that any server may handle the data out of a control unit description file. All configuration data of the diagnostic server, the internal data of ECUs or ECU nets and the communication methods for the ECU access are stored in the ODX database. This database is server and operating system independent and therefore allows data to be exchanged between vehicle manufacturers and ECU suppliers.

An application can read out all data from the database that is necessary to drive the MVCI diagnostic server; this means only the MVCI diagnostic server can access the information of the separate control unit description files comprised within one database. With this, at the same time the consistency of the information between AUSY and MVCI diagnostic server is guaranteed.

Also, a decoupling from the used data description exchange format (XML) takes place.

The MVCI diagnostic server has to manage the database and to provide the required and necessary information for the single MVCI diagnostic server objects. The database does not belong to one specific MVCI diagnostic server object, but is available within the whole diagnostic server. The organization of the object or reference allocation is solved implementation specifically by the diagnostic server.

The object model supports Single Client Systems, to provide for a simple use for this most typical and most frequently occurring application case. This means that no client references are included within the single objects. The administration of the client references is done by the diagnostic server and has to be solved implementation specifically.

The object model has been designed in a technology and programming language independent way. It may be used remotely as well as locally.

The object model of MVCI diagnostic server enables the linking of MVCI diagnostic servers to automation systems. The objective of this linking is the remote control of the MVCI diagnostic server in test stands. By means of the object model, the functionality, which means the interfaces with accompanying methods, are standardized. The communication has to be realized via the particular implementation of the object model for the used platform, programming language and linking mechanisms.

Among others the following are realized:

— ASAM GDI,

— JAVA,

— COM/DCOM and

— C++.

The necessary specifications for this will be described and published in separate documents. For this process design patterns and mapping rules are defined and published.

All other specifications will be set up and realized implementation specifically by the respective system provider.

Within the function block diagnostic a breaking down into characteristic sub tasks will take place, which are shown in the following:

The Communication Processor is responsible for generating and analysing request and response telegrams to ECUs. This processor handles all protocol-specific tasks like timings, creation of protocol headers and checksums, etc. Diagnostic protocol specification is (at the moment) not a task of ASAM, because this is covered by ISO activities according to ISO 14230-3 KWP 2000, ISO 14229 UDS and ISO 15765 DoCAN. Nevertheless, the communication processor shall be parameterized via Communication Parameters. The Communication Processor is an interface (the only one) to the ECU.

The Data Processor is responsible for the supply of parameters and results on a physical level. By means of the Data Processor all necessary information is fetched from the database. Additionally, the Data Processor converts ECU answers from hexadecimal representation into a physical or any text representation and vice versa. The Data Processor is an interface (the only one) to the ODX database and offers an ODX library to the Job Processor. The Data processor also handles Jobs, as they are stored in the ODX database.

The Flash Data Processor is responsible for the loading of programs and data in ECUs. The flash data is part of the database. The Flash Data Processor provides access to the ASAM MCD2-ECU-MEM which contains all information about physical/logical data-/code-layout and possible combinations of data and code segments and more. The Flash Data Processor is an interface (the only one) to the flash data and offers a flash library (flash object) to the Job Processor.