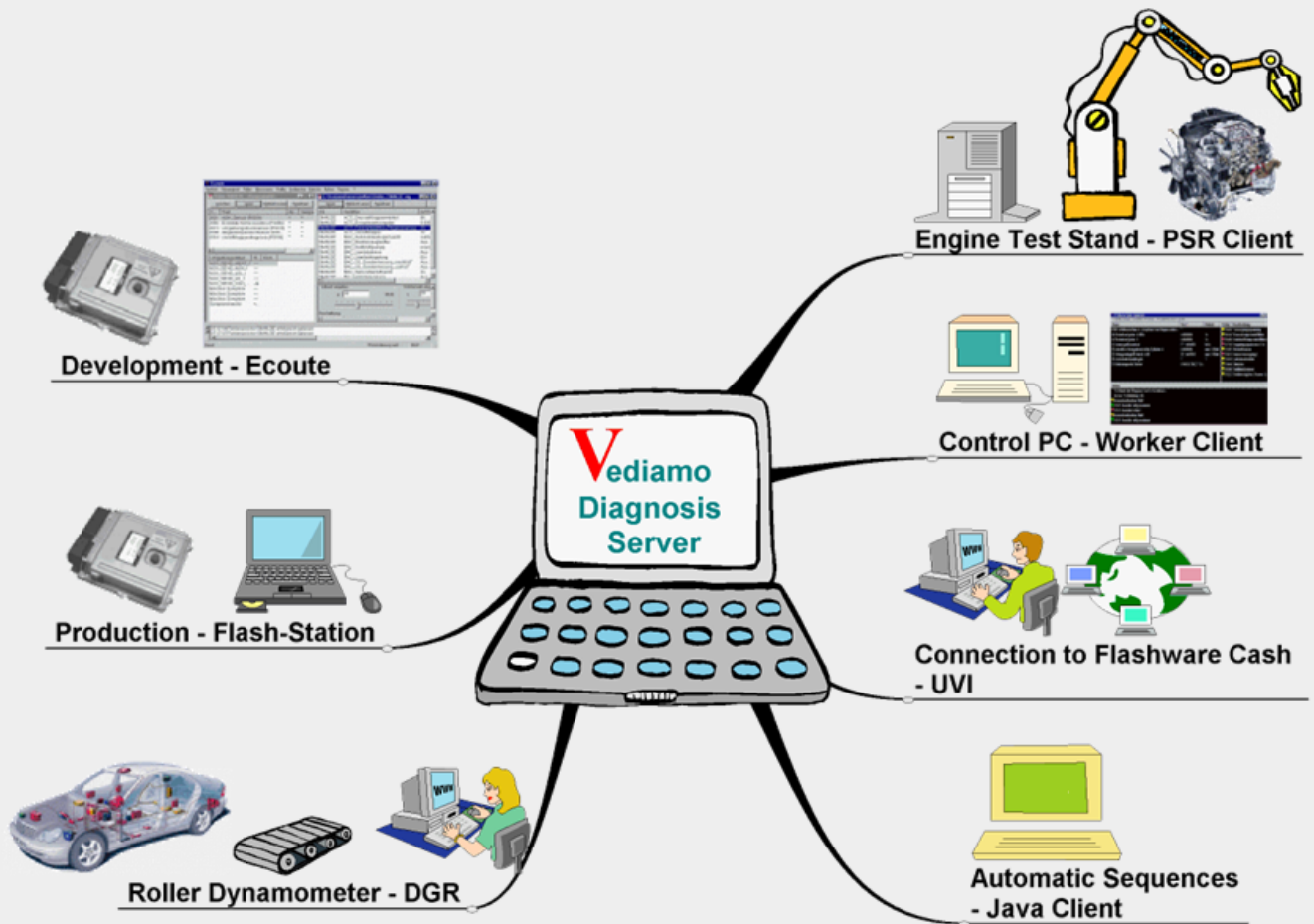


# Vediamo Manual



Version	05.00
Creation Date	10 / 2013
Copyright ©	Mercedes-Benz, PWT/VEP
	Werk 010, HPC H152
Homepage	<a href="http://diagnose.intra.daimlerchrysler.com/vediamo/">http://diagnose.intra.daimlerchrysler.com/vediamo/</a>
FAX	+49 -(0)711 17 - 7908 1949

# VEDIAMO

version 05.00.05

Application licences:

ECOUTE	Licence available
DIMELO	Licence available
FlashStation	Licence available
WERKER	Licence available

## New or improved in this version:

Support for smr-x (ODX-x) Diagnosis data  
Editor and interpreter for OTX Sequences (Version 1.0)  
SOAP based API  
Diagnosis over IP

**For further informations and downloading diagnostic data  
visit our homepage**

<http://diagnose.intra.daimlerchrysler.com/vediamo/>

built: 19.05.2014

Diag kernel version: (MVC1)

version date: 19.05.2014

INI-file: C:\Documents and Settings\All Users\Application Data\Vediamo\Config\Vediamo.ini

Licence: 16.05.2014 18:26:08

Disclaimer

# Table of Contents

1	Introduction.....	7
1.1	Application Areas, Focus .....	7
1.1.1	Development .....	8
1.1.2	Production .....	8
1.1.3	Java Routines.....	8
1.2	First Steps.....	8
1.2.1	Basic Terms.....	8
1.2.2	Operating Instructions for The Hasty.....	10
1.3	Development Cycles .....	13
1.4	Association with Hardware and Other Software .....	13
1.4.1	CAESAR.....	13
1.4.2	LUCA.....	15
1.4.3	PSR.....	16
1.4.4	UNIPAS .....	16
1.4.5	Java Runtime.....	16
1.5	Perspectives .....	17
1.6	Ordering, License, Support .....	17
1.7	Installation.....	18
1.7.1	System Requirements.....	18
1.7.2	Earlier Versions .....	18
1.7.3	Installation Procedure .....	19
1.7.4	Installation Options / Parameters .....	20
1.7.5	Installing and Checking the eCom Hardware (Part P) .....	21
1.7.6	Other Required Hardware/Software .....	22
1.8	Design and Operating Mode .....	22
1.8.1	C/S Architecture.....	22
1.8.2	Layer Model.....	23
1.8.3	DCOM .....	23
1.8.4	WCF and .Net 4.0 .....	24
1.8.5	Multitasking (Multi-ECU) .....	24
1.8.6	Logs.....	24
1.8.7	Vediamo Configuration .....	26
2	The Vediamo Modules .....	27
2.1	DiagServer .....	28
2.1.1	Introduction.....	29
2.1.2	Configuration (INI Parameter) .....	29
2.1.3	Diagnostic Parameterization .....	29
2.1.4	Examples: How can I... .....	30
2.2	StartCenter .....	32
2.2.1	Structure.....	32
2.2.2	Functions of the StartCenter .....	32
2.3	System Configuration.....	35
2.3.1	Introduction.....	35
2.3.2	Structure.....	36
2.3.3	The Functions.....	41

2.3.4	Working with the System Configuration .....	54
2.3.5	Special Features .....	60
2.3.6	Configuration (INI Parameters) .....	61
2.4	<b>Ecoute</b> .....	<b>61</b>
2.4.1	Introduction.....	62
2.4.2	GUI Structure.....	63
2.4.3	The Ecoute Files.....	68
2.4.4	The Ecoute Functions .....	68
2.4.5	The Ecoute Menus.....	155
2.4.6	Keyboard Operation.....	163
2.5	<b>Java Handler Functions</b> .....	<b>166</b>
2.5.1	Vediamo Java Interface .....	166
2.6	<b>Java Programs (Java Routines)</b> .....	<b>167</b>
2.6.1	Introduction.....	167
2.6.2	Executing a Java routine from Ecoute or Another Client .....	168
2.6.3	Java Program as Standalone Client.....	168
2.6.4	Example: Program, Compile and Execute a Simple Routine.....	169
2.6.5	Particulars.....	170
2.6.6	Configuration (INI Parameters) .....	171
2.7	<b>BlackBox</b> .....	<b>171</b>
2.7.1	Introduction.....	171
2.7.2	Structure and Function.....	171
2.7.3	BlackBox Functions .....	172
2.7.4	BlackBoxViewer: Log Display at Runtime .....	172
2.7.5	Linking to Other Applications .....	174
2.7.6	Configuration (INI Parameters) .....	174
2.8	<b>PSR Adapter</b> .....	<b>175</b>
2.8.1	Introduction.....	175
2.8.2	Communication between Vediamo and PSR .....	176
2.8.3	The Functions of the PSR Clients .....	178
2.8.4	Configuration .....	180
2.8.5	The Engine Table .....	181
2.8.6	Examples - how can I.....	182
2.9	<b>2.9 Worker Client</b> .....	<b>187</b>
2.9.1	Introduction.....	187
2.9.2	Structure.....	188
2.9.3	Function Description .....	189
2.9.4	Examples: How can I... .....	190
2.9.5	Command Line Parameters .....	191
2.9.6	Configuration (INI Parameter).....	191
2.10	<b>Other Clients</b> .....	<b>192</b>
2.10.1	Flash Station.....	192
2.10.2	DiMeLo .....	193
2.10.3	UVI .....	193
2.10.4	More Clients And Utilities.....	193
2.11	<b>INI Editor</b> .....	<b>193</b>
2.11.1	Menu .....	194
2.11.2	User Interface Areas.....	195
2.11.3	Input Elements.....	195
2.11.4	All INI Parameters.....	196

3	How Can I.....	197
3.1	Connect a Vehicle.....	197
3.2	Connect an ECU without a Vehicle.....	198
3.3	Flash an ECU.....	199
3.4	Restart the Server.....	200
3.5	Read Measurements from an ECU.....	201
3.5.1	Read individual measurements.....	201
3.5.2	Read multiple measurements simultaneously or read measurements cyclically.....	201
3.6	Read an ECU ID Block.....	201
3.7	Read and Clear an ECUs Error Memory.....	201
3.8	Execute a Quicktest.....	202
3.9	Perform Variant Coding.....	202
3.10	Execute a Java Routine (Java Program).....	202
3.11	Change the Connection Between K-Line and CAN.....	203
3.12	Open Ecoute in the Same State I Closed It.....	204
4	Glossary.....	205
5	INI Parameters.....	209
6	PSR Messages.....	226
7	Example: Java Routine.....	237

This page intentionally left blank

# 1 Introduction

Vediamo - the distributed diagnostic application for engines (**V**erteilte **D**iagnose **A**nwendung für **M**otoren) - is a software system for electronic control unit (ECU) diagnostics which is integrated in the CAESAR/DIOGENES process chain. It allows diagnostics on any ECU over K-line as well as CAN, and encompasses all established protocols from RTMD+, MBISO, KWFB, KW2000 to UDS. Despite its name, Vediamo is no longer limited only to engine ECUs.

Vediamo is based on the Mercedes-Benz DCDI standard communication platform CAESAR and hence supports all communication hardware such as, e.g., Part C, Part A, Part Y, etc. DIOGENES data, used by all diagnostics applications in the company and which can be drawn from the central TAMINO data base, can be used as diagnostics data.

Future versions of Vediamo will be set up on the MVCI (ASAM standard) developed by GSP/ODE and will also support the ODX diagnostics data format.

The implemented client server architecture allows, among other things:

- distributed diagnostics with multiple, simultaneous special client applications, e.g., test run control over PSR adapter while simultaneously verifying with Ecoute
- Expansion of the system by addition clients,
- Diagnostics, coding, or flashing of multiple ECUs simultaneously

## 1.1 *Application Areas, Focus*

Vediamo originally had two user groups in mind: Development and Production.

The system configuration is used to adapt the available ECUs to the user's specific requirements, e.g., by selecting the standard connector, integration of automatically executed initialization services, selection of selected services from a longer list, and much more.

The architecture of the system allows it to be expanded easily by additional functionalities and even complete applications. This is done primarily using Java programs, for which Vediamo possesses a simple yet comprehensive connection, encompassing all ECU functions. Even users with limited programming skills in the standard language Java can develop their own applications. Generally applicable functions can subsequently be integrated in the clients by the Vediamo developers (e.g., graphic measurement display or the quick test) or be developed as new DCOM clients.

### 1.1.1 Development

The Ecoute client allows interactive access to all available functionalities of one or multiple ECUs using numerous specialized tools. E.g., access to all available services is possible, windows can be customized with a selection of different types of services which can be executed once or periodically; all communication parameters can be modified, any request messages can be sent to the ECU, and all data streams, procedures and events can be logged in many specialized formats.

### 1.1.2 Production

The test bench controller (PSR) adapter as well as the UVI (Unipas-Vediamo Interface) allows automatic testing without requiring manual changes in the software. The complete test run is controlled by the test stand controller, which the Vediamo diagnostic server provides with all the data necessary to control the test runs and evaluate their results. In order to relieve the test software of diagnostic know-how, the Vediamo clients' TS adapters and UVI have very simple interfaces, tailored to the needs of the test stand, which allow effective access to the diagnostic data over a serial line or network connection.

The flash station allows automated flashing of ECUs.

### 1.1.3 Java Routines

The Vediamo-Java interface allows customized applications to be generated in the standard programming language Java without requiring knowledge of diagnostic protocols, CAESAR-DIOGENES interfaces, or client-server architecture. These Java programs can serve to either support the clients Ecoute (development) and TS adapter (engine test facility), or can be used separately as independent applications.

## 1.2 *First Steps*

### 1.2.1 Basic Terms

Communication hardware	Hardware components (CAESAR, DCDI), which are connected between the PC and the ECU. The computer needs CAESAR software (CAESAR-slave library c32s.dll) to operate this hardware, and possibly hardware drivers.
ECU	Electronic Control Unit. Electronic unit in vehicle which controls the function of the engine or other systems and which can communicate for diagnostic purposes with external equipment (tester, PC with communication hardware) either directly or by way of another ECU (gateway).



System	<p>This term refers to a set of ECUs which are diagnosed at the same time. The set can consist of either a single or multiple ECUs. Vediamo allows simultaneous communication with multiple ECUs, depending on ECU type (K-line or CAN) and the configuration of diagnostic hardware (number and type of connections, number of hardware components, ...). Particularly during the quick test, the complete vehicle is considered as one system and the communication takes place with all installed ECUs.</p> <p>A system file (VSB - Vediamo System Binary) is generated for each system. It is only valid in connection with the parameterization of the included ECUs (CBF files).</p> <p>The term <i>system</i> corresponds approximately to the term project in ASAM systems.</p>
Diagnostic Service	<p>A function of an ECU that causes a specific action or which sends information from the ECU to the tester (PC). Typical services are, e.g., read measurement results (such as engine temperature), set controller (throttle valve, etc.), and many more.</p> <p>Generally for a service, a message is sent from the tester to the ECU and a reply is received and evaluated. However, the actual number of sent messages can be greater.</p>
Resource	<p>Hardware connection. For K-line communication (serial connection), every ECU needs a separate line which can be switched by the multiplexer or manually with banana plugs. CAN communication always uses the same CAN bus, but the communication hardware requires an internal channel per ECU. The number of channels depends on the CAESAR hardware and software used.</p>
Contact	<p>State in which data exchange with the ECU is possible. The ECU must be initialized (activated) to establish contact. This state must be actively maintained by the diagnostic hardware. Only an ECU in contact state replies to messages from the tester.</p>
Variant coding	<p>Special diagnostic service which stores information on the variant (e.g., on a vehicle's options) in the ECU.</p>
Flashing	<p>Most ECUs can be programmed by Vediamo with different software (firmware). This process is called flashing.</p>
DiagServer	<p>Primary module of the Vediamo system. Controls the diagnostic data, ECU states and coordinates service execution on behalf of clients.</p>
Client	<p>Application which executes certain tasks and communicates with ECUs by way of the DiagServer.</p>

## 1.2.2 Operating Instructions for The Hasty

### Hints for the Admin

After Vediamo has been installed (which of course requires Admin privileges), it can be run without Admin privileges. Since Vediamo accesses numerous files during operation and also writes in parts of these, certain access privileges may have to be granted.

The Vediamo files are installed into the ALLUSERSPROFILE folder. This folder is defined in environment variables and depends on the version of the operation system.

In Windows XP Professional it may be:

```
C:\Documents and Settings\All Users\Application Data\Vediamo
```

In Windows 7 the folder normally is:

```
C:\ProgramData\Vediamo
```

In this folder four subdirectories are created, in which any user has full access rights:

- VediamoData - for Diagnosis data (CBF, VSB etc.)
- VediamoShorttestData - for Shorttest data files
- Config - for INI files, license files and other configuration files
- Log - for Log files created during runtime

The first two folders can be changed during installation.

If an older version of Vediamo was installed, the setup uses the former folders instead of creating new ones.

### If You Have No Diagnostic Hardware and/or License

You can start Vediamo in simulation mode and test its possibilities. This makes it easier to decide whether Vediamo has the tools you require.

- [Install](#) Vediamo with standard settings
- Load the data for selected ECUs from the [VEDIAMO intranet homepage](#), copy these into the data directory (usually ... \VediamoData)
- Open the Vediamo [StartCenter](#)
- From the list under *Ecoute* select "Simulation" and start [Ecoute](#) with a click on the button.
- Use *System / Select* to open a system (it contains one or more ECUs)
- Try out all possible functions you find in the menu. These functions offer practically all the possibilities which CAESAR offers.

- Some functions are not possible in simulation mode. These include, e.g., flashing, manual command input and monitoring. All others can be executed, but have less than fascinating results - e.g., measurement services always return the result "?". The service *Read Errors* returns the result "no Errors". However, it is possible to obtain other results in simulation mode if you have a [simulation file](#) for the appropriate ECU.
- You can uninstall Vediamo completely. Go to *System Control / Software / Vediamo Diagnostic System* and select *Remove*. You can subsequently delete the path `Vediamo` in `ALLUSERSPROFILE` as well as `C:\Program Files\Vediamo` along with all the contents.

### **If You Have Diagnostic Hardware but No License for Vediamo**

You can install Vediamo anyway and begin testing in simulation mode (see above). If you like to test Vediamo in actual operation before you purchase a license, then proceed as follows:

- First install the CAESAR hardware and, if necessary, the appropriate drivers. Details can be found on the [VEDIAMO intranet homepage](#), the procedure is independent of the type of hardware used.
- Select the function "update Vediamo Server / CAESAR" in StartCenter
- Make certain that the [CAESAR parts](#) you are using are selected. If uncertainty exists, multiple entries can be selected. The software recognizes automatically whether the specified parts are actually connected and addressable.
- Start the "update" function. After a certain amount of time information about the detected hardware will be displayed.
- Make a note of the displayed serial number(s) of your hardware.
- [Order](#) a test license, stating your hardware serial number. One number is sufficient in case of multiple hardware components.
- As soon as possible (e.g., by email), you will receive a customized `VLicence.inf` file. Copy this file into the `...\Vediamo\Configdirectory` (usually `C:\Documents and Settings\All Users\Application Data\Vediamo\Config`)
- Open the Vediamo StartCenter again and start *Ecoute*. If no message is displayed that you do not have a valid license or that no hardware could be found, your Vediamo is ready for operation.
- Connect the ECU. Exact directions are given [below](#).
- If necessary, download the data for your ECU from the [VEDIAMO intranet homepage](#) and copy these into the directory `...\VediamoDaten`.

### **If You Have both Hardware and A License**

After verifying as in the previous steps that Vediamo is installed, the data for your ECU is copied, the ECU is connected correctly, and the license for your CAESAR hardware is correctly installed, it is best to begin your work with the StartCenter. You have direct access there to all installed Vediamo components and their settings.

The program used most is [Ecouté](#), the interactive client that provides all diagnostic functionalities. This client, as with all others, only functions when the [DiagServer](#) is running. This is started automatically via [DCOM](#) as soon as a client (in this case Ecouté) wants to access it. Since only one CAESAR instance can be active per computer, the server is a "singleton". Dependent on the [client/server architecture](#), multiple Ecouté clients (or other clients which can be installed with Vediamo or programmed yourself, e.g., with [Java](#)) can be used at the same time.

Use the [system configuration](#) to generate or modify the system files (it is possible to work without them, but they make the work significantly easier). If you have modified a system file, the DiagServer has to be started again, if it is running. The easiest way of doing this is by ending the Ecouté client and starting it again.

In addition, you will occasionally use the [INI editor](#). It starts when you click on the *Options* symbol in the StartCenter.

In case of problems with the software such as crashes, unforeseen events, contact problems and much more, the [BlackBox](#) should be activated by the parameter [BLACKBOX]run=1 in the file Vediamo.ini. The BlackBox is started then by the server and runs in the background to log what's "happening". If necessary, the logs from the BlackBox can be sent by email to the Vediamo team for analysis.

After all INI parameters have been set in accordance to your needs and you wish to work only with Ecouté, you can start Ecouté directly without the StartCenter. Select *Start / Programs / Vediamo / Ecouté* from the Windows menu, or place a shortcut to Ecouté.exe on the desktop.

Starting Ecouté (or any given Vediamo client) automatically starts the DiagServer. Ending the last client automatically also ends the DiagServer.




Important:

Stopping the server can take up to 20 seconds. If you start a Vediamo client before the server is completely ended, this can lead to unstable program results such as crashes and unrecognized hardware, or similar. In this case, use the task manager to make sure the process DiagServer is ended (or terminate it yourself), remove any possible "leftovers" from Ecouté or other clients and then start the client anew. It is not necessary to restart the computer.

### **In Case of Problems...**

...you should start the BlackBox first. This program creates logs. Then check the Vediamo file directory. To be able to trace errors, the Vediamo team needs exact information on the software version used, the settings ([Vediamo.ini](#), can be found in the program directory ... \Vediamo\Config), specification of your DCDI hardware (type,

serial number), as well as all log files and the exact details on the CAESAR data used by you. If you have installed Vediamo with the standard configuration, you will find:

- The CAESAR data as well as system files in the directory `... \VediamoDaten`
- Files which have been generated for a Vediamo system, such as measurement result and controller groups, in the subdirectory of `... \VediamoDaten` with the same name as the Vediamo system.
- Program logs and traces of ECU communication in the directory `... \Vediamo\Log` and its subdirectories.
- Information on program versions can be found either via file properties / version, or by starting the program and calling up *Info on...* This menu entry can be found in the main menu "?" in Ecoute. For programs without a user interface (DiagServer, BlackBox, PSRClient) in the taskbar menu: click on the appropriate icon ,  or  with the right mouse key and select the item *Info on...* This info window will also tell you which program is actually running and which INI file is being used.  
If you have multiple versions of Vediamo installed at the same time, this information can help you avoid problems.

## 1.3 ***Development Cycles***

There is at least one new Vediamo version a year which integrates the most current CAESAR hardware and software and supports all previous and newly introduced protocols.

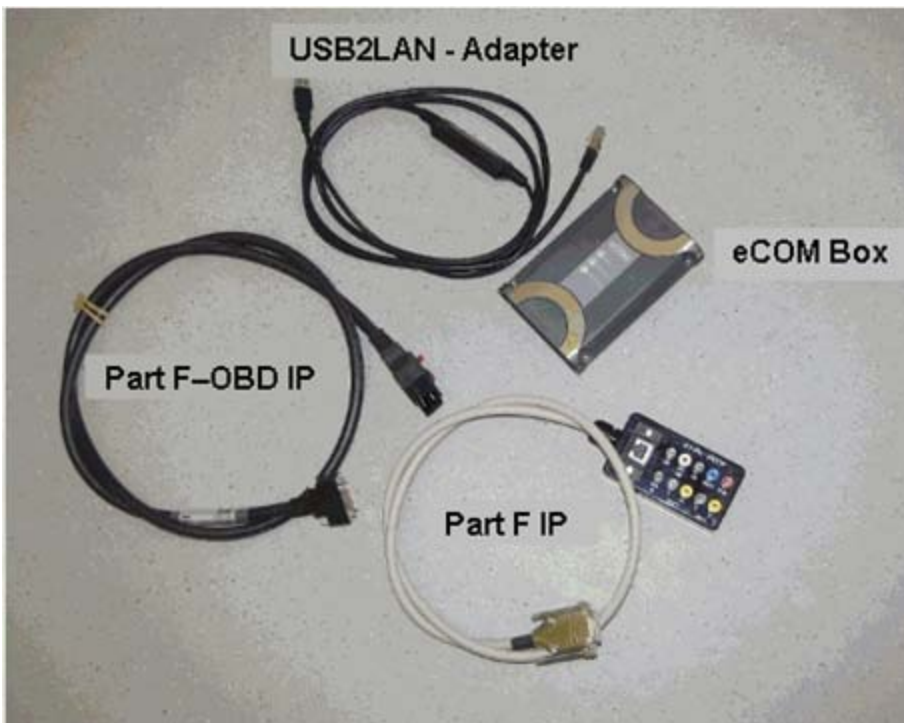
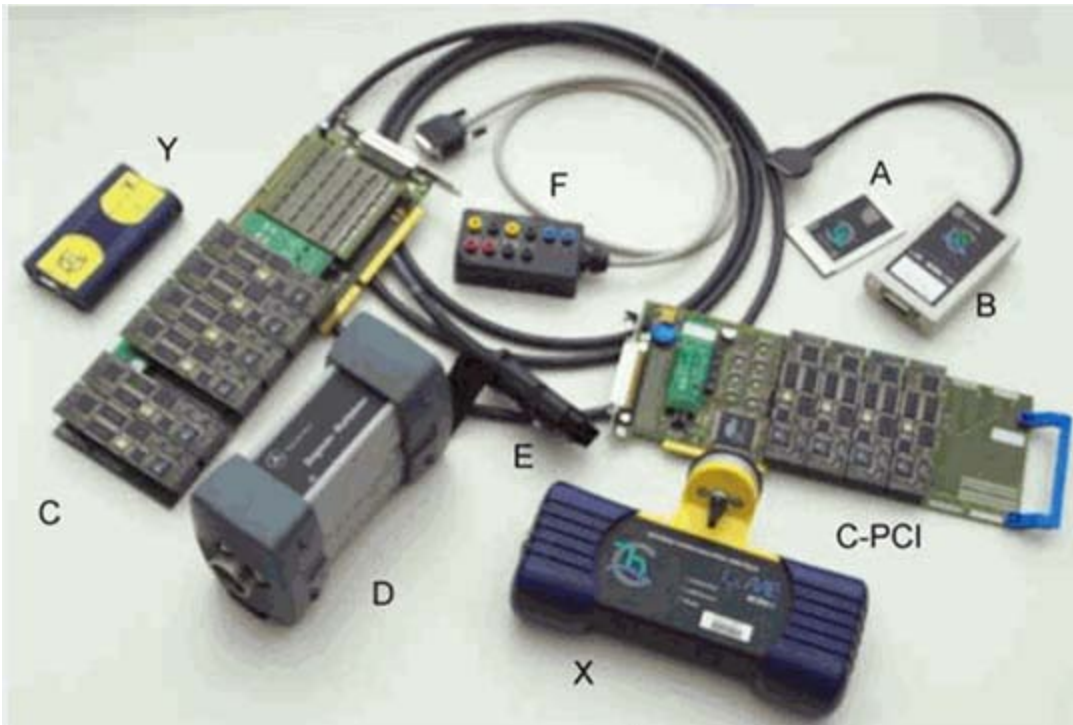
Additional releases are distributed as soon as critical errors, urgent modifications or new CAESAR versions make this necessary. In the latter case, the Vediamo release follows the newest CAESAR version by the few weeks required to perform tests and quality control.

## 1.4 ***Association with Hardware and Other Software***

### 1.4.1 **CAESAR**

The connection to the diagnostic hardware (CAESAR, DCDI) by the appropriate drivers and libraries forms the base of the Vediamo diagnostic server.

CAESAR hardware and software suppliers can be found [here](#).



The following hardware components are supported:

- Part A: PCMCIA card, also requires Part B as well as drivers
- Part C: ISA or PCI plug-in card with a variable number of piggy backs (processors for parallel diagnostics) and integrated multiplexer
- Part D: "First generation" diagnostic hardware, with serial connector
- Part E: OBD connector cable with integrated multiplexer, is connected to Part A+B, Y, or similar
- Part F: Connector box for lab set up, can be used in place of Part E but does not have a multiplexer
- Part J: Pass-through device
- Part P - eCom: also known as MVCI-Device, PDUAPI
- Part X: Diagnostic part with wireless PC connector
- Part Y: Diagnostic part with USB or serial PC connector
- Part W: SDConnect...

Independent of the hardware component used (it is also possible to simultaneously use several different ones), various CAESAR files (Dynamic Link Library, Release.cmf, INI files, protocol definitions, etc.), as well as the DIOGENES parameterization in the form of a CBF file for every ECU to be diagnosed, and possibly other files (CCF coding files, CFF flashware files, etc.) are required.

Especially the driver software delivered with the hardware must be installed before use.

In case of Part P - eCom, the configuration file `slave.ini` must contain a proper *RootPath*-Entry, depending on the folder the driver software has been installed in.

For further configuration please consult the documentation of the user hardware part(s).

Important:

The Caesar Hardware: Part A / Part B / Part C / Part Y are not supported in Windows 7.

## 1.4.2 LUCA

Langner Universal Communication API (LUCA) from [Langner Communications AG](#) in Hamburg provides a standardized programming interface for different communication protocols and allows effective protocol enhancements without intervention in the program flow. The communication between PSR and PSR adapter occurs over serial lines (3964R protocol) or over LAN (HDLC, TCP/IP) using the LUCA.DLL. The library LUCA.DLL is delivered together with Vediamo and can be installed (with a PSR adapter) when required.

It is therefore recommended to implement the same communication software when developing test bench software. If required, the Vediamo development team delivers code examples for easier integration of LUCA.

### 1.4.3 PSR

Simplicity, robustness, clarity, effectiveness, reliability and error tolerance are all of great importance in the interaction with the test bench. At the same time, the diversity of the test programs, different test bench manufacturers, and continually changing ECUs and their data must be accommodated as well as the requirement that testing time be optimized on account of the high quantities and the integration in larger processes.

This diversity results in the high elasticity of the PSR adapter as well as the necessity to design the test bench software just as flexible in accordance with the Vediamo PSR specification. In particular, the test bench controller cannot assume that the complete process is static in the sense that the time between certain actions or the sequence of messages always remains the same. It is possible, e.g., that loading new ECU data can significantly change the timing of the test run. Even closing a visualization or log window can result in an appreciable acceleration of reactions.

### 1.4.4 UNIPAS

This software for controlling test benches is used by Vediamo to communicate using the special client UVI commands from UNIPAS. Answers from Vediamo are transmitted in XML format via DCOM (via LAN). UVI has a relatively limited functionality.

### 1.4.5 Java Runtime

The Java runtime environment from [Sun Microsystems](#) is required for running Java routines. In theory, every standardized Java Runtime should execute the programs correctly. However, it is recommended to use the included and automatically installed version. The Vediamo-Java interface is tested with the included version. In addition, all Java relevant parameters are automatically entered correctly in the `vediamo.ini` when Vediamo is installed.

#### Integrated External Software

VEDIAMO is installed with components developed by other companies. By installing the user accepts the license agreements of these modules.

The following modules are used:

Apache Cocoon 2.1.11	Apache License 2.0
Apache XMLSecurity 1.4.5	Apache License 2.0
Apache POI 2.5.1	Apache License 2.0
JDOM 1.1	Apache License 2.0
Xerces-C++ XML Parser Version 3.1.1	Apache License 2.0
Xqilla	Apache License 2.0
Java JDK 1.7.0_25	GNU General Public <i>License</i> (GPL)



Fat Jar Eclipse Plug-In  
TinyXml  
CodeSynthesis XSD Data Binding compiler V. 3.3.0  
GPL  
zlib License  
GPL V. 2

## 1.5 ***Perspectives***

Vediamo will continue to be integrated in the diagnostic process in the future, in the same timely manner in which Vediamo has always supported all new diagnostic hardware and been based on the newest available CAESAR software. Vediamo will support the standardized ASAM server software platform when it is introduced and will continue to provide the familiar user interface and connection to test benches.

Vediamo-specific data such as system files (VSB), measurement and controller groups (MWG, STG) can either be assumed from older versions, or Vediamo converts existing files into the new format.

## 1.6 ***Ordering, License, Support***

The current Vediamo version as well as data for ECUs can be found on the [VEDIAMO intranet homepage](#).

A Vediamo license is valid for specific communication hardware, independent of the computer. The hardware ID numbers are read by the diagnostic server after CAESAR has been initialized and compared with entries in the license file. If multiple different hardware components are connected to the computer, it is sufficient that one of them is licensed - Vediamo can still address all the parts. A list of all found hardware components with their ID number can be seen in the status window after Ecouste has been started.

If Vediamo is licensed, any desired number of clients can be used separately or simultaneously. The clients DGR and FlashStation, which require a special license, are the exception.

The license is valid for a specific version, but only the two first number pairs are relevant, i.e., the 05.00.00 license applies to all subsequent updates 05.00.xx, but not to 04.02.xx or 05.01.xx.

No license is required for operation in simulation mode. There is no communication with ECUs in this mode, the data exchange with ECUs is simulated on the server. This mode is for demonstration purposes, for training, and for testing DIOGENES parameterization.

A functional unlimited but temporary license can be requested for evaluation purposes.

For single or multiple license orders as well as error reporting please contact:

Mercedes-Benz,  
Dept PWT/VEP,  
Werk 010 HPC H152,  
Fax +49 -(0)711 17 - 7908 1949

Order forms, current information or help with problems/questions can be found online on the [VEDIAMO intranet homepage](#) under *Contact*.

## **1.7 Installation**

### **1.7.1 System Requirements**

- Pentium-III-PC with 1 GHz (1.5 GHz recommended)
- Windows 2000 or Windows XP or Windows 7
- 1024 MB RAM memory
- 250 MB available hard disk space for Vediamo in addition to hard disk space for diagnostic data (5 GB recommended)
- Depending on the communication hardware to be used, either a free PCI or ISA connector, a USB interface, COM port or PCMCIA slot is required.
- A network connection must exist for distributed application (server and clients on separate computers).
- A network connection or serial interface (COM port) is used for communication with the test bench controller.

The diagnostic hardware and if necessary the drivers, have to be installed separately either before or after installing Vediamo. The appropriate entries have to be made in the `Vediamo.ini` settings file after installation to allow the hardware to function with Vediamo.

Lower PC performance characteristics or a loading from other processes (open applications, SMS services, virus scanners, etc.) can reduce Vediamo's performance.

### **1.7.2 Earlier Versions**

Dependent on the client-server architecture, no two Vediamo versions can be installed at the same time. If an earlier version is installed, it must be uninstalled first before the new version can be installed. However, a copy of the settings( `vediamo.ini`) as well as files generated by the user (system files BSB, measurement and controller groups, etc.) are not deleted.

If a new version is installed in a directory an earlier Vediamo version was installed, the `Vediamo.ini` settings file found there is taken over so that the new Vediamo generally works exactly like the old one.

Possible changes in the file formats for the new Vediamo version are automatically applied to the files if required.

### 1.7.3 Installation Procedure

Vediamo is always delivered as a separate setup program. All system components are attuned to one another and may not under any circumstances be mixed with components from other versions (not even release and debug versions with the same version number). To install, you need admin privileges. If you work with Windows 7, the setup should be started "As Admin".

During installation, setup checks whether `*.ini` or `*_ini.old` settings files are in the specified path. If this is the case, all settings which are still valid for the new version are taken over and only new ones are entered. All setting can be modified subsequently using the [INI-Editor](#).

To uninstall, follow the directions given by *Startmenu / Settings / System Control / Software → Uninstall Vediamo*.

#### 1.7.3.1 Alternative Default Settings for Data Paths

The default paths for diagnostic data and for short test data can be overridden by optional INI files. These files must be stored separately for CAESAR and MVCI usage and they can contain the following items:

```
[ SERVER ]
SystemPfad=[path to diagnostic data
(VSX, SMR-x etc) ]

[ ECOUTE ]
ShortTestDataDir=[path to special set
of diagnostic data (VSX, SMR-x etc) for
short test]
```

To overwrite the mentioned paths, start the Vediamo Setup with command line parameters as follows:

```
VediamoSetup050100.exe /v"OPT_INI_PATH_C=<alternative INI file for CAESAR>" /v"OPT_INI_PATH_M=<alternative INI file for MVCI>"
```

"C" and "M" mean Caesar and MVCI.

It is also possible to pass only one file as parameter.

### **Important**

If you change the default or legacy data path, a new directory will be created, but your data files will not be moved there automatically. This must be done separately.

## **1.7.4 Installation Options / Parameters**

All the user's inputs during installation can be logged. The resulting answer file can be used to perform silent installations, i.e., Vediamo can be installed on other computers automatically without user interaction. The selections are then made on the basis of the log file.

To log the user inputs in the file `vediamo.iss`, start setup with the following parameter:

```
VediamoSetup.exe /r /f1"c:\temp\vediamo.iss"
```

This answer file can be used on other computers to execute setup automatically without user interaction:

```
VediamoSetup.exe /s /f1"c:\temp\vediamo.iss"
```

It is not necessary to enter the filename. In this case, the file "Setup.iss" in the Windows system directory is created/read.

**Important:**

If a filename is entered, the complete path must be included even if the file is in the current directory. No path or relative paths lead to unforeseen results.

For a complete pre-configuration, it is recommended to combine the silent installation with a pre-installed `vediamo.ini`, i.e., generate a batch file which:

1. Generates the installation path with a subdirectory BIN.
2. Copies a prepared `vediamo.ini` into the BIN subdirectory.
3. Starts setup with the parameters listed above.

### **Tips on Silent Installation**

- Delete/rename any previous answer files when logging a new answer file.

- Do not use answer files from previous Vediamo installations.
- The setup process varies depending on whether an INI file is available or not. An answer file with INI should therefore only be use with an INI.
- An InstallShield icon is displayed in the Windows taskbar during silent installation. If this is missing or disappears much too quickly, the installation is not running.

### 1.7.5 Installing and Checking the eCom Hardware (Part P)

- To use the eCom Hardware with "USB2LAN Adapter" a driver must be installed. You find the driver on the Vediamo Homepage "DCDI Treiber". Alternatively you can get the driver directly [from ASIX \(http://www.asix.com.tw/products.php?op=pItemdetail&PItemID=97;71;101&PLine=71\)](http://www.asix.com.tw/products.php?op=pItemdetail&PItemID=97;71;101&PLine=71) .
- Interface configuration:
  - Switch on the power supply for the eCom Hardware.
  - In Network find the correct connection ("ASIX AX88772A USB2.0 to Fast Ethernet Adapter" or so) and edit the properties. In TCP/IP properties enter the following **fixed** IP-Address:
    - IP-Address 169.254.255.40
    - Subnet Mask. 255.255.0.0
- Firewall Settings:
  - depending on the used firewall, check if the following ports are opened. More info is available on the Diagnose Homepage under Tools|Diagnose Hardware|Driver & Installation eCOM

Usage	Protocol	Port
eCom	UDP	1024
	UDP	4011
	UDP	4012
DoIP	TCP	13400
	UDP	13400
	UDP	13401
	UDP	13402
	UDP	13403

- Function check:
  - Switch on the power supply for the eCom Hardware.( the green LED on eCom Hardware must burn)

- Plug in the USB to LAN adapter
- in ...\\Program Files\\Vediamo\\Caesar\\driver\\eCom\\I+ME Actia GmbH\\XS D PDU API execute the program IME\_D\_PDU\_API\_Tester.exe. At the top select for "D-PDU API DLL" the PDUAPI\_I+ME\_ACTIA\_XS.dll, press the START button at the bottom. If everything works fine, all check boxes in the list should be marked after some time..
- The eCom should be recognized by the Vediamo Start-Center now (otherwise check if eCOM Part P is selected)

## 1.7.6 Other Required Hardware/Software

### DCDI Hardware

Forms and ordering process are included on the [VEDIAMO intranet homepage](#) under *Ordering Information / CAESAR Hardware*.

Further information on the diagnostic hardware can be found on the [Diagnostics Portal](#).

### Driver (DCDI)

Can be found on the [VEDIAMO intranet homepage](#) under *Downloads / DCDI Driver*

### Diagnostic Data (VSX, SMR-x & Co.)

Are on the [VEDIAMO intranet homepage](#) under *Downloads / Diagnostic Data*.

### Java Development Environment

If you like to program your own [Java routines](#), you will need a development environment. You can get one free of charge from [Eclipse](#) and from [Oracle](#).

## 1.8 Design and Operating Mode

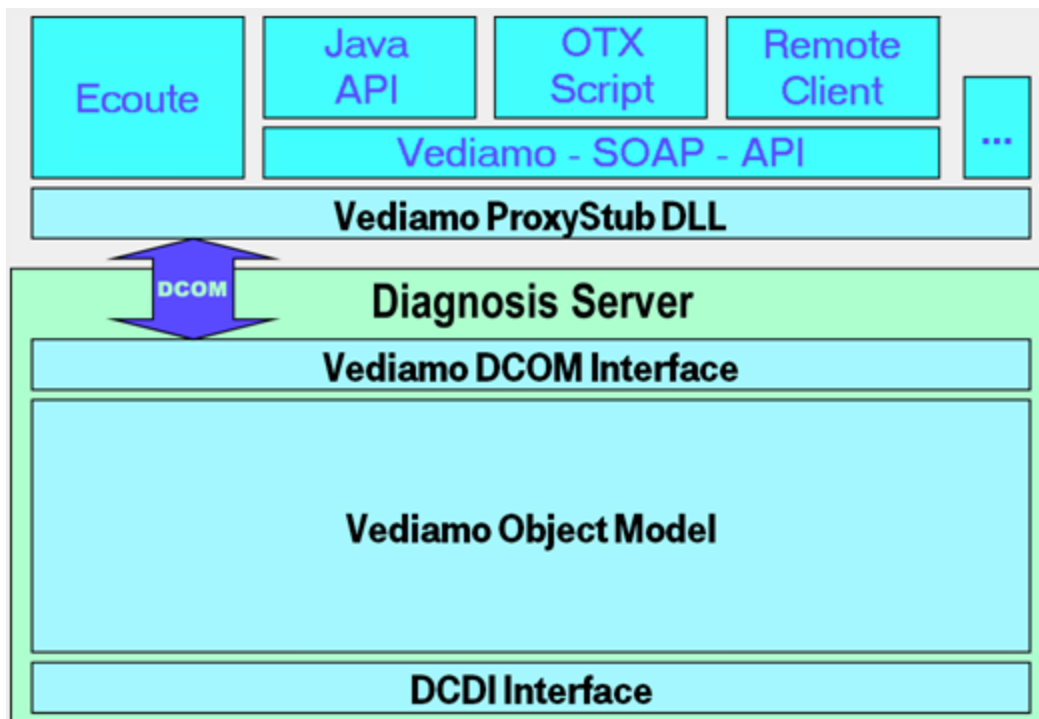
### 1.8.1 C/S Architecture

Separating the Vediamo diagnostic system into a server and multiple client components allow among other things:

- Diagnostics with multiple special client applications at the same time, e.g., test load control using the PSR adapter while simultaneously checking the communication on byte level with Ecoute.
- Expanding the system by additional clients such as, e.g., Java programs which automate certain processes or evaluate data and transmit it to other applications

- Distributed installation: Checking the test processes on multiple computers at the same time: at the test bench (worker) and in the office (shift manager)
- Simultaneous operation of multiple ECUs, e.g., one ECU can be calibrated while at the same time another is being flashed

## 1.8.2 Layer Model



The architecture of the Vediamo system resulted from a requirement for simplest reusability. The DCOM API makes the effective development of special clients possible. More than one client at one time can be used (multi client capability).

The SOAP API is a standardized API designed to run OTX sequences and to supply an easy, platform independent access for remote systems.

## 1.8.3 DCOM

DCOM (Distributed Component Object Model) is a Microsoft technology for realizing distributed client/server applications. It is part of Windows operating systems as of Version 98 and NT 4.0. Vediamo uses this technology for the client communication with the diagnostic server.

After installation, Vediamo is immediately ready for operation. No computer reboot nor system configuration if necessary to run DiagServer and clients.

The Vediamo DCOM architecture allows the usage of slim, specialized clients simultaneously with the standard client (Ecouste). There is no need for universal clients overloaded with lots of functions used only by few. This makes also the development cycle shorter and the system more extensible.

#### 1.8.4 WCF and .Net 4.0

Collaboration with engine test stands and standardized test routines needs a platform independent, well-documented interface. This is offered as a OTX-SOAP-API, based on HTTP, XML and WSDL. This API can be accessed by other systems via network calls from every system. It can be a Windows PC, but also any other machine with a Java Runtime.

The SOAP API is constructed using Windows Communication Foundation (WCF) based on .NET 4.0. If your machine does not have .NET 4.0, it will be installed by the Vediamo Setup.

#### 1.8.5 Multitasking (Multi-ECU)

Thanks to multitasking, the Vediamo DiagServer can operate multiple ECUs simultaneously, up to the full application of all communication channels (CAESAR resources) and the CPU capability. Since most of the time in ECU communication is spent waiting for a reply from the ECU, the performance capability of the computer is optimally applied. It has only become possible with Vediamo multitasking to simultaneously flash and calibrate two ECUs with only one computer and one CAESAR hardware.

The application, i.e., the concrete task to be completed, is the focus of the Vediamo philosophy and not the setup of the hardware. This means, e.g., that measurement results from different ECUs can be shown at the same time in one output window, while a number of actuators are adjusted in a different window.

#### 1.8.6 Logs

The log files recorded during runtime are not required for regular operation, but are necessary for trouble shooting. There are different log types which can be turned on and off with different INI parameters (see [Vediamo Configuration](#)) as required.



The log files are stored in several directories, depending on settings in the INI file. It is however not necessary to search them - in the Start Center the button "Logfile administration" opens a list of all log files, where you can pack them into a ZIP archive or delete them.



## Program Logging (BlackBox)

Obviously no software is complete error-free. It is therefore possible that Vediamo could experience a program error. A detailed log of all foregoing activities is necessary in order to determine the position as well as the exact situation. This is done by a type of software trip recorder - the [BlackBox](#).

If you have DiagServer or a client crash or other problems with the system functions, please activate the BlackBox in the INI editor.

These logs are an exception; they are not constantly written to the hard drive but are recorded in a ring buffer in RAM. If a serious error occurs (client or server crashes) or if the user requests it, the last 5000 entries (this number can be changed by INI parameter) are written to a file. These logs contain entries from all Vediamo applications running at that time. In case a problem occurs, the situation does not have to be recreated with logging turned on (which does not always guarantee another occurrence of the problem). Rather, the log is stored after the problem and it does not additionally load the system during normal operation.

## MVCI and Hardware Driver Logs

There are several files logging internal processes of the diagnosis hardware and driver software of different levels.

You find these files in the following directories:

- MVCI layer: [AllUsersProfile]\Vediamo\DTS\Traces
- PDU-API layer: [AllUsersProfile]\Vediamo\PDU\IME\PDU\logs

The logging depth and paths can be configured in diverse IN and XML configuration files.

## ECU Logs

##### überarbeiten AFEWORK

The communication with an ECU takes place using a communication channel. Channel specific logs are generated to analyze the communication with individual ECUs:

- The queries and replies from the ECU can be monitored in the Ecoute trace window either in data block form, or in detail as individual bytes with time stamps. This can be configured via *Ecoute Options / Protocol*.
- Channel-specific CAESAR output as well as the ECU communication can be logged in files on the server side. A file is generated for each communication channel with a name consisting of the ECU ID and the ending `kanal.log`. The

configuration can be done via *Ecoute Options / Protocol* or directly by INI parameters.

## **Status.log**

This log contains all texts which are displayed in the Ecoute status window.

## **System Configuration Logs**

The system configuration generates logs of various processes (e.g., update or consistency check of a system description). Since the system configuration uses CAESAR, the outputs generated by CAESAR to log the internal processes also result when working with the system configuration. These outputs can be monitored in the log window at runtime, but they are also stored in the file `VediamoSysConfLog.txt` for subsequent analysis.

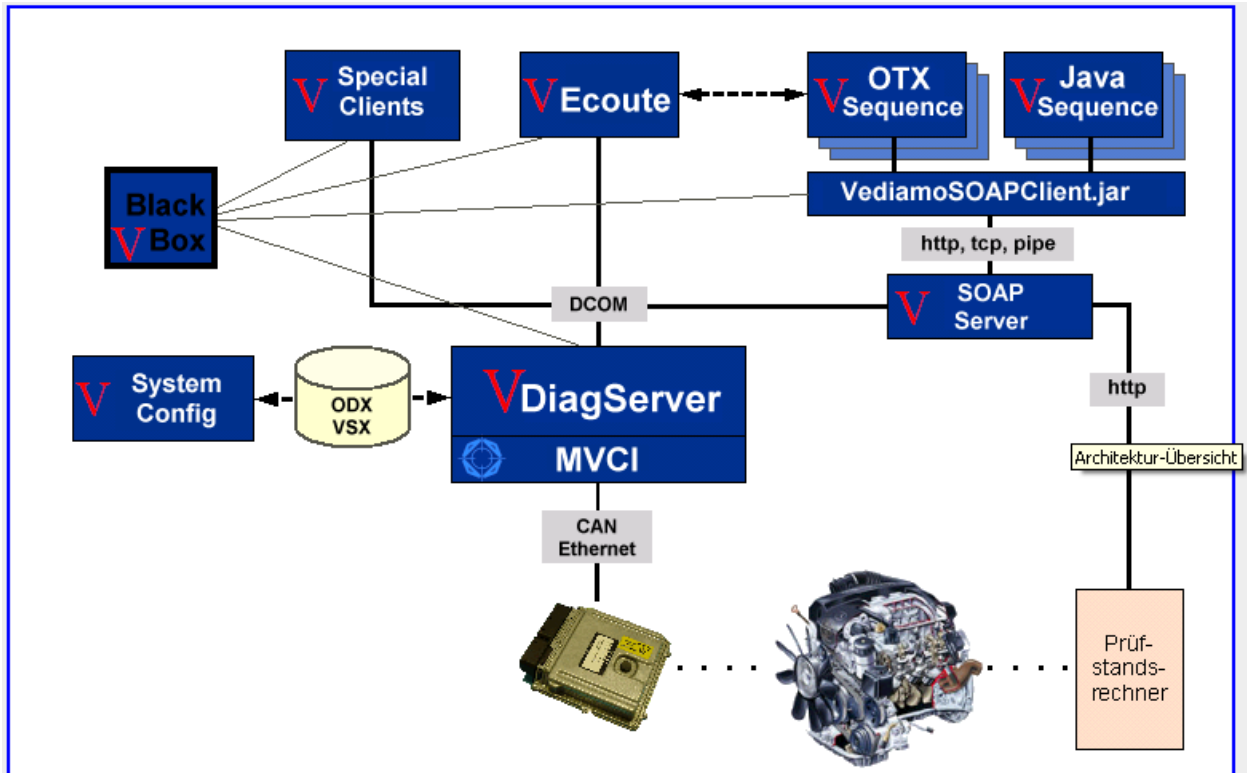
### **1.8.7 Vediamo Configuration**

The Vediamo system is configured using the `vediamo.ini` file. The file is separated into sections, each assigned to a system module (DiagServer, CAESAR, clients, Java connection, or "common" for common parameters). During a new installation the parameters are set to values which generally allow optimal operation. Some values are changed during runtime by the applications (e.g., window positions). It is recommended to only make changes where results are understood and desired. The [INI-Editor](#), which also contains a complete description of all parameters, is used for configuring.

The configuration possibilities for the modules are presented in detail in the following sections on the Vediamo system modules.

## 2 The Vediamo Modules

Vediamo consists of several applications (colored blue in the figure):



### DiagServer:

This application encapsulates the diagnostic hardware and software, makes their functionality available to the client applications (local as well as distributed over LAN), and coordinates client access to the system. The DiagServer manages the ECU data, establishes communication with the ECUs, executes services, and delivers results or received information to the client(s). A sophisticated client administration allows each client to have practically unlimited access to all necessary functions: the server coordinates the access so that

- requested information reaches only those clients expecting it
- clients do not block one another, but in the worst case just slow each other down (when using the same communication channels)
- the same information for multiple clients is only acquired once and distributed to the clients

### Clients:

These are the programs which the user sees and operates: [Ecoute](#), [Worker-Client](#), [DGR](#) etc. They are designed for specific tasks - it is their job to exchange data

with the ECUs over the DiagServer and to process or forward it. At the same time, the client itself can be a server for a further application, e.g., the PSR adapter which serves 2 clients: the test bench controller and the worker-client.

**System Configuration:**

Independent program for generating and editing system descriptions.

**BlackBox:**

This program is a type of watchdog. It stores log outputs from all Vediamo modules (servers as well as clients) in RAM, in order to write them into a log file in case of a crash or other serious problem. In this manner, all of the final events prior to the occurrence of a problem are stored. This continuous operation impedes neither the runtime nor the hard drive space.

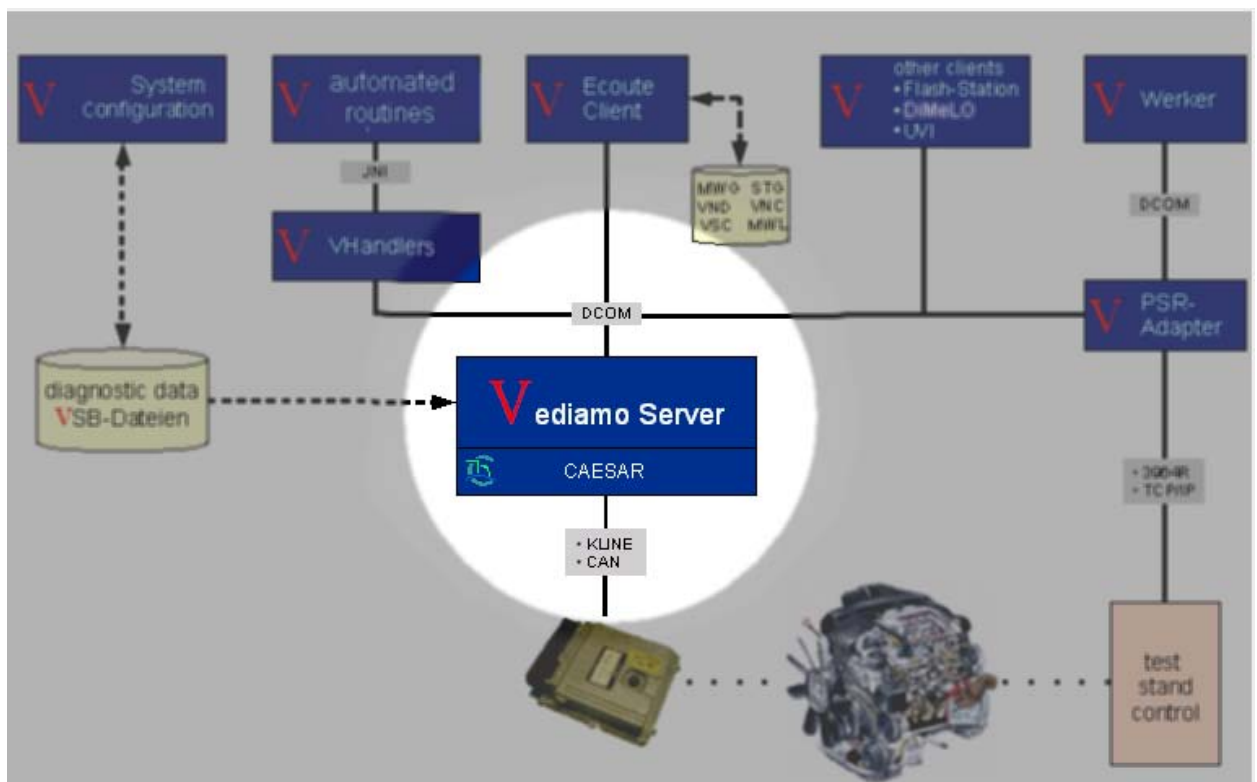
**StartCenter:**

A user interface for compact presentation of all important Vediamo elements.

**INI Editor:**


A program for configuring the Vediamo settings. It contains detailed descriptions of all parameters.

## 2.1 DiagServer



## 2.1.1 Introduction

The server is the central module of the entire system. Its job is to control the communication with the ECUs using the available DCDI hardware, to administer the data, to process and to coordinate the data exchange with the clients.

The DiagServer has no user interface of its own, other than a menu and an info window. This menu can be accessed via the icon  (right mouse key) in the taskbar and contains the items *Info on...* and *Exit*.

The DiagServer has a DCOM interface which allows the clients to access its functions. Multiple clients can be active in parallel - the coordination of their data exchange is handled and optimized by the server.

## 2.1.2 Configuration (INI Parameter)

The Vediamo DiagServer can be configured with the [INI-Editor](#) . The following sections are included in the `Vediamo.ini` for the DiagServer:

COMMON:

Settings applicable to all Vediamo modules, e.g., the language to use, are found here.

SERVER:

These are the server-specific settings, e.g., the path for the system descriptions.

CAESAR:

These are for configuring the performance of CAESAR

## 2.1.3 Diagnostic Parameterization

DiagServer requires a number of files for the data exchange with ECUs:

- CAESAR files, consisting of GBF, CMF, FRM and INI files. They are installed along with Vediamo in the subdirectory `Caesar` and must match the CAESAR software used.
- ECU files: `CxF` (x represents different characters). They define the ECU communication parameters and diagnostic services.
- System files `VSB`. They define systems with one or more ECUs, assign connection numbers to the ECUs, define lists of services to be displayed to the user, as well as a number of additional specifications. More on this under [System Configuration](#).
- Simulation files `SIM`: These files can be used to try out the Vediamo components without contacting an ECU.
- Java routines `CLASS, JAR`: These are Java programs which can be integrated into the system descriptions `VSB` to expand the diagnostic functionality.

Note for JAR files:


When starting a JAR sequence the Java commandline must specify the main class. The server takes this information from the file manifest.mf, which must be contained in the JAR.

## 2.1.4 Examples: How can I...

### 2.1.4.1 End Server ("Kill")

The server ends automatically as soon as the last client ends or is terminated by other means (crash, ended by task manager). This process can take up to 20 seconds if a client crashes.

For some problems, however, it can be necessary to end the server process itself. There are two possibilities of doing this:

- As long as the server is not completely out of control, open the server menu (right mouse click on the server icon  in the taskbar) and select "Exit".
- As a last-ditch effort, even if a crashed server has not released all PC resources, you can end the process "DiagServer" in the task manager ("Processes" tab).

### 2.1.4.2 Exchange CAESAR Hardware

Vediamo supports the following CAESAR hardware, which can be activated by subsequent INI entries:

- Part A (USE\_SIPCMCIADriver)
- Part C (USE\_SIPartCDriver)
- Part D (USE\_SISerialDriver)
- Part E (USE\_SIPartEDriver)
- Part J (USE\_SIPartJDriver)
- Part P - eCOM Box (USE\_SIPartPDriver)
- Part X (USE\_SIPartXDriver)
- Part Y (USE\_SIPartYDriver)
- Part W (USE\_SIPartWDriver)

To exchange or install CAESAR hardware, please proceed as follows:

- Determine which requirements for operation must be met from the manufacturer's hardware specifications (correct connector, system requirements, required hardware driver).
- If the manufacturer makes a (software) tool available for checking the functionality of the hardware, this should be used to verify the functionality.
- Activate the appropriate entry in the "vediamo.ini" file under "CAESAR" using the [INI-Editor](#).

- Deactivate all hardware entries which you will not use in the foreseeable future. It is possible to keep all hardware entries activated, but at the least this increases the amount of time that CAESAR needs at Vediamo start-up to determine which hardware is actually connected.
- Start Vediamo anew. The recognized CAESAR hardware is displayed in the Ecoute application's status window. A function test of connected hardware can be made using the function "update Vediamo Server / CAESAR" in the StartCenter.

### 2.1.4.3 Flash CAESAR Hardware with other Firmware

CAESAR software consists of several parts which are structured in different ways. For example, there is a difference between the CAESAR "master" and CAESAR "slave".

"Master" refers to that part of the CAESAR software which runs directly on the computer being used and which for example handles the implementation of the available DIOGENES data or actually interprets the replies from the ECU.

"Slave" is the CAESAR hardware/software which is directly responsible for the communication with the ECU. This "slave" requires "firmware" for operation, i.e., software which generally is stored directly in the non-volatile memory of the CAESAR hardware (hence the name "firmware").

CAESAR makes different types of firmware available, dependent on the application. Vediamo supports the operation of the following CAESAR firmware:

CaesarGo:

This firmware usually runs directly in the non-volatile memory of your CAESAR hardware, e.g., in Part A or Part Y. Supports older communication protocols as well, e.g., KWFB. The new UDS protocol is not supported.

TLSlave:

This firmware runs on the computer used. The faster the computer, the quicker the different communication tasks can be completed (e.g., flash processes). Also allows the simultaneous opening of up to 50 CAN channels. This firmware supports "newer" protocols as well, e.g., UDS.

BusSim:

This is special firmware which is used to support CAESAR's CAN bus simulation function.

The setting to determine which firmware to use can be made either in the Startcenter or with the [INI editor](#) (under CAESAR). The diagnostic server must be restarted for a change to become effective.

### 2.1.4.4 Try Out DiagServer without Hardware or License (Simulation Mode)

A [simulation mode](#) is available in Vediamo in which the communication with an ECU is simulated on an abstract level without an actual ECU present. The simulation mode can

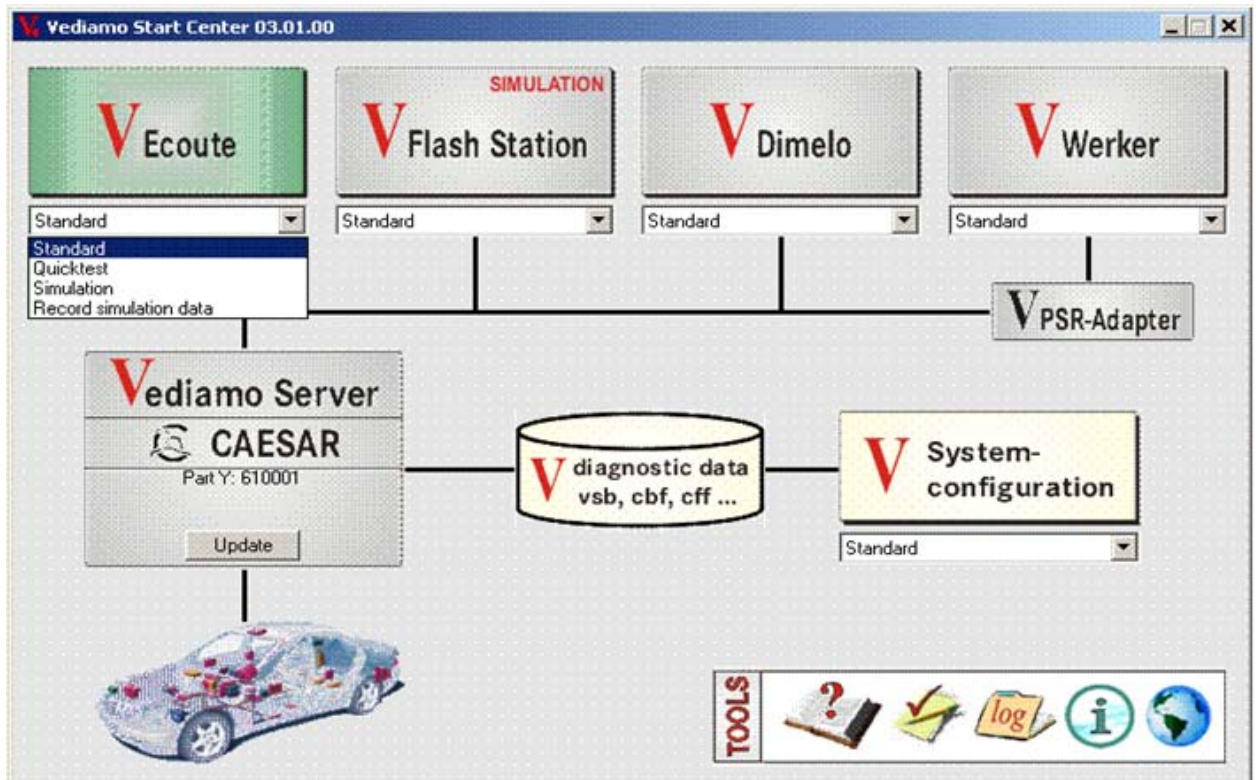
be permanently activated by making an INI entry with the [INI editor](#) in the "SERVER" section.

If no valid hardware or license is available only simulation mode is supported. In this case Ecoute indicates upon starting that the license is unavailable and asks whether Vediamo should be operated in simulation mode.

## 2.2 StartCenter

The Vediamo StartCenter is a simple application where all Vediamo programs can be started and managed from. The program interface contains a graphic display with the symbols of the separate Vediamo modules and additional tools.

### 2.2.1 Structure



### 2.2.2 Functions of the StartCenter

#### Start Vediamo

A program can be started clicking on the respective symbol or button. For programs which can be started with command line parameters an option exists to select predefined starting parameters below the program symbol.

The Vediamo modules can be licensed separately. If a module is not licensed, then the



label “simulation” is displayed on the respective program symbol, indicating the fact that this module can be operated only in simulation mode. For modules currently running the respective program symbol is displayed with green background.

## Module Start Options

For modules which can be started with command line parameters (e.g., Ecoule - quick test), a list of possible start options is displayed below the module symbol.

The behavior of the StartCenter is controlled by means of a text file (StartProfile.txt) which contains all the information on the modules to be started and the command line parameters. The text file can be adapted / expanded with a text editor.

If not at least the "Standard" entry is available for a certain module, the related button in Startcenter appears disabled.

The file has the following structure (for example):

Example:

```
;Vediamo StartProfile file
;Filename can be given as command line parameter of StartCenter.exe
; e.g., "StartCenter.exe MyProfiles.txt"
;
; File format:
; Ecoule||Profile description||Ecoule.exe||Command line parameter
; System Configuration||Profile description||SystemConfiguration.exe||Command line
parameter
; Worker||Profile description||Werker_Client.exe||Command line parameter
```

```
Ecoule||Standard||Ecoule.exe
Ecoule||Quick test||Ecoule.exe||-K
Ecoule||Simulation||Ecoule.exe||/vi "[Server] Simulation 1"
Ecoule||Record simulation data||Ecoule.exe||/vi "[Server] Simulation 2"
System Configuration||Standard||SystemConfiguration.exe
System Configuration||With data import/export in text
format||SystemConfiguration.exe||/VI "[SYSTEMCONFIGURATION]
EnableImportExport 1"
Worker||Standard||Worker_Client.exe
```

Lines in the file beginning with semicolons are interpreted as comment lines.

## Configure Vediamo Programs or Modules

The [INI-Editor](#) for managing and editing the INI parameters can be called up using the symbol "Options" in the StartCenter.

After editing and accepting Vediamo options, a notice is displayed that the changes become effective only after a Vediamo restart.

## **Update Diagnostic Hardware**

With the function "update Caesar hardware" it can be examined whether the selected hardware is operational. The status of all detected Caesar parts activated in the Vediamo.ini and, if possible, their serial number is determined. The numbers of the accessible parts are displayed on the left side in the field "Vediamo server/Caesar" in a listbox.

A dialog is showed in which the activation of the used caesar hardware (Part ...) can be made.

In addition the following entries can be activated:

- Use Part E

- Use Mux mode (Pinmapping)

This controls whether a CAESAR Part E is used during diagnosis and whether pinmapping is activated. Pinmapping is automatically activated when Part E is used.

- Use TL-Slave firmware or CaesarGo firmware

This controls which firmware the connected CAESAR slave uses. Further information on this can be found in the CAESAR documentation.

At the bottom of the dialog, available information about the currently detected Caesar hardware, such as cable ID and security level is displayed.

Changes in these settings immediately take effect for the diagnostic server. Upon exiting the dialog by pressing the "Update" button, the diagnostic server will be restarted using the changed settings.

## **Edit Diagnostic Data**

The button "diagnostic data" opens an explorer window in the currently set diag. data path.

## **Logfile Administration**

By selection of the symbol "logfile administration" down right the dialogue "Trace Analysis" is displayed with a list of all presently relevant Vediamo log files. Individual log files can be selected in the dialogue at will. Over a context menu (right mousebutton) then the following functions can be implemented for the selected files:

- Open file(s) using the standard editor

- Delete file(s)

- Copy the path(s) of the selected file(s) to the clipboard

- Open the path(s) of the selected file(s) in an explorer window

- Copy selected file(s) into a .zip archive

## **Open Vediamo Help**

The button *User Handbook* opens and displays the user documentation.

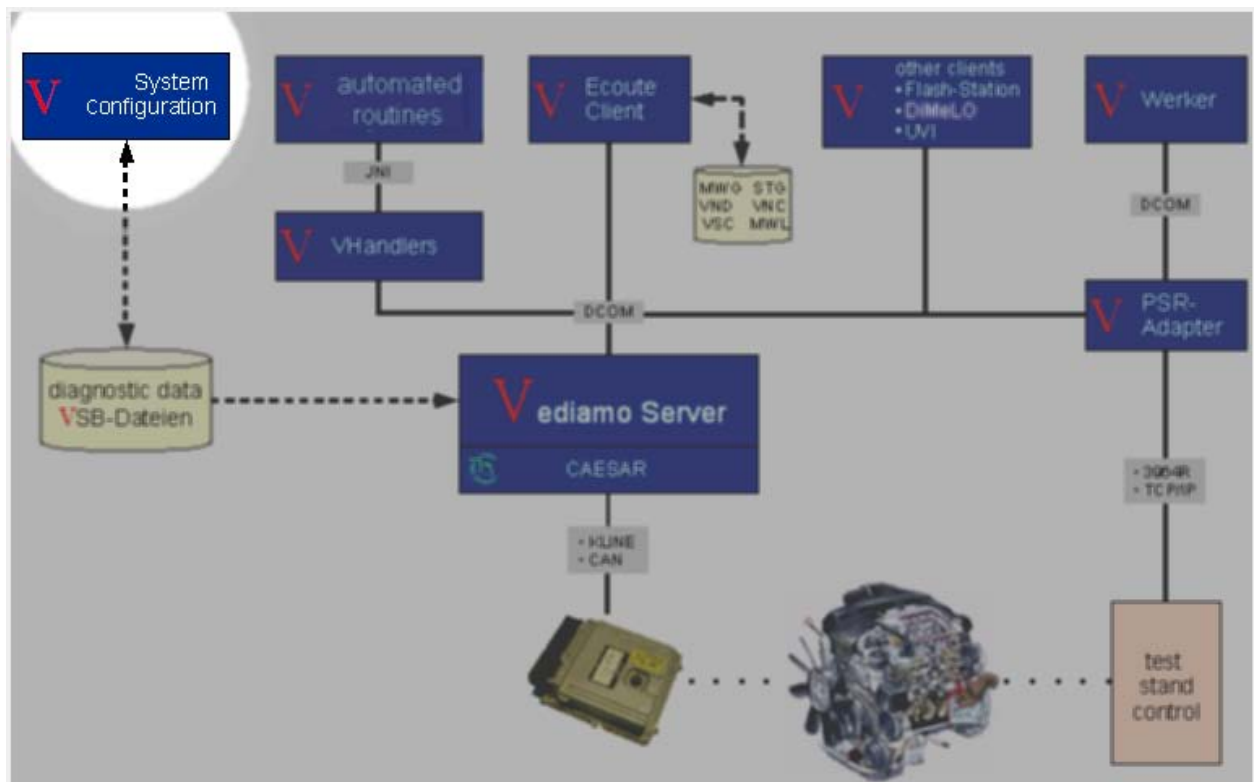
## Version Information

Clicking on the button *Version / Info* determines and displays the current versions of Vediamo and CAESAR.

## Vediamo Homepage

Clicking on the button *Vediamo Homepage* opens the Vediamo Homepage in your standard internet browser.





## 2.3 System Configuration



### 2.3.1 Introduction

The Vediamo system configuration is used to configure an ECU system. With the Vediamo system configuration, it is possible to:

- Generate system descriptions ([VSB files](#))

- Filter out measurements, functions, actuators, etc., which may not be used in the engine test facility. Filter out errors and error environment data which result due to a missing complete vehicle system. The filter status of an entry is displayed through the following symbols:
  -  For positive filtered entries. If the particular entry contains further subentries of which not all are filtered positive, the symbol will be displayed in gray..
  -  For negative filtered entries.
  -  For erroneous entries.
  -  For new, unfiltered entries.
- Reference [Java routines](#)
- Create [standard objects](#)
- [Search](#) for entries

Beside these basic functions, additional functions are available to ensure data consistency at runtime for an ECU system:

- Updating of the CAESAR data files and Java routine files used in a system description.
- Consistency check of the data contained in a system description.

The system configuration user interface supports multiple languages. The language is defined by the entry *Language* in the [\[COMMON\]](#) section of the configuration data `vediamo.ini`. The following condition must be met in order to work with the system configuration:

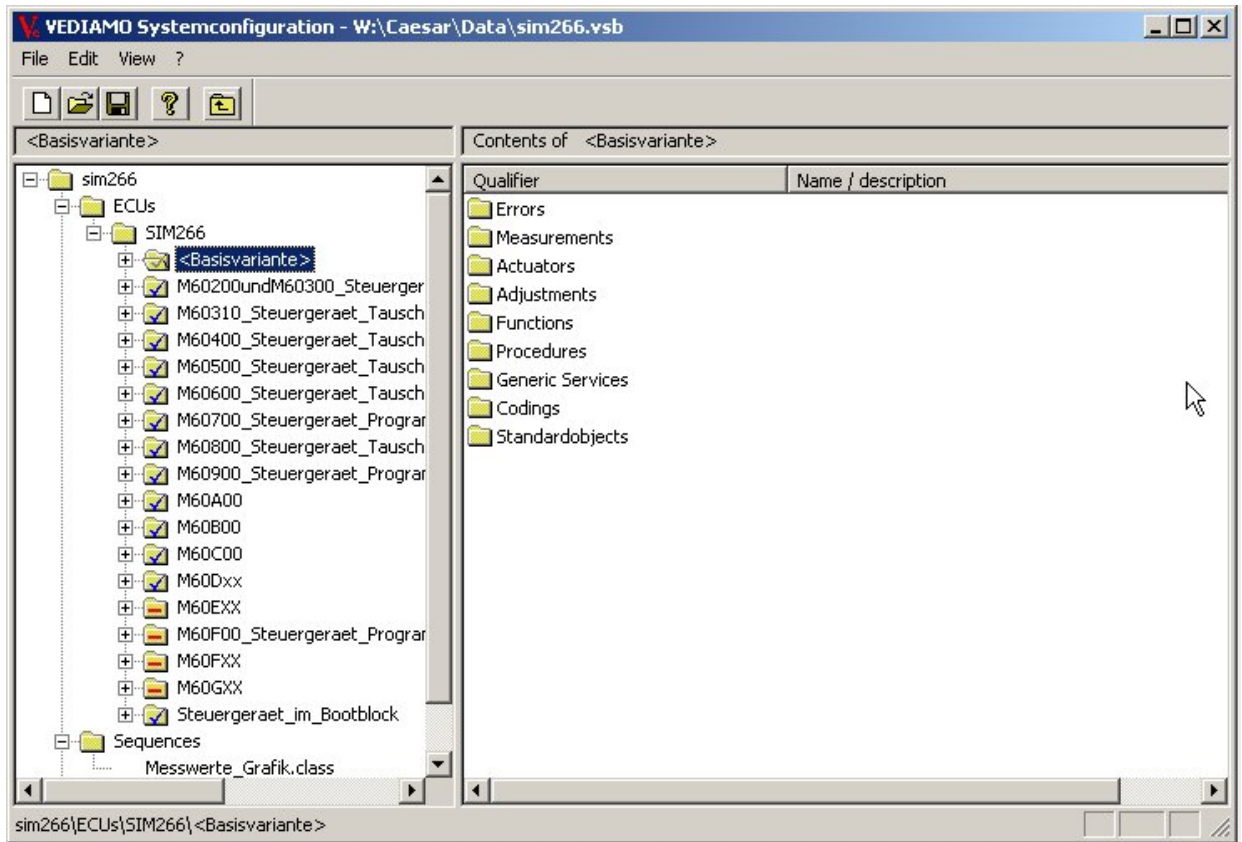
- Correct [installation](#) of the Vediamo system configuration portion (especially the CAESAR data files).

CAESAR Hardware is not required.

## 2.3.2 Structure

The system configuration is an independent application. Neither CAESAR hardware nor the Vediamo diagnostics server are required for its operation.

The system configuration user interface is shown in the following figure:



The name of the current system description being edited is depicted in the title bar of the main page. The [menu bar](#) for selecting the various program functions is below the name. Below the menu bar is a [symbol bar](#). Frequently used functions, some which are also accessible using the menu bar, can be executed by clicking on the symbol with the cursor. The main page is split vertical into halves. The setup resembles the Windows Explorer application. In the left half, the system setup is depicted in the form of a tree structure, while in the right half, the contents of the currently selected elements are shown in form of a list, and if available, a description of the respective entry is included. Clicking the column headings in the right half causes an assortment by the contents of the current column, alternating ascending/descending.

A status line in which various information is displayed, depending on the context, is located at the bottom of the main page

## The Menu bar

The following menu functions are available:

The File menu

- [New](#)
- [Open](#)
- [Save](#)
- [Save as](#)
- [Names of last opened files](#)
- [Close](#)
- [Import](#)
- [Export](#)

The Edit menu

- [Settings](#)
- [Refresh](#)
- [Check consistency](#)
- [Filter all elements positive](#)
- [Filter all elements negative](#)
- [Reverse filter settings](#)
- [Automatically change error environment data](#)

The View menu

- [Symbol bar](#)
- [Status bar](#)
- [Log window](#)





The ? menu

- [Help topics](#)
- [System information](#)

Every entry in the tree structure includes a context menu, a listing of the related functions accessible through that entry. These menus are made displayed by placing the cursor over the entry and right-clicking the mouse.

## The Symbol bar

The Symbol bar allows the following functions:

- |   |   |
|---|---|
|  | <a href="#">New file</a>                                |
|  | <a href="#">Open file</a>                               |
|  | <a href="#">Save file</a>                               |
|  | <a href="#">Information on the system configuration</a> |



Back: Displays the contents of the immediate superior entry in the list on the right side of the main window.

## The Tree structure

The following entries are possible in the tree structure of the system description:

### System

- ECU
  - ECU with
    - ECU-variants with filter information on the following services:
      - Errors
        - Environment data
      - Measurements
      - Actuators
      - Adjustments
      - Functions
      - Procedures
      - General Services
      - Coding
      - Standard objects
        - Standard object class coding
          - Standard object of the class coding 1 - n
        - Standard object class flashing
          - Standard objects of the classification flashing 1 - n
        - Standard object class services
          - Standard objects of the class services 1 - n
        - Standard object user defined class 1
          - Standard objects 1 - n of the user defined class 1
        - ....
        - Standard object user defined class m
          - Standard objects 1 - n of the user defined class m
- routines
  - Java routine 1-n
- Standard objects

- Standard object class Coding
  - Standard object of the class coding 1 - n
- Standard object class flashing
  - Standard objects of the class flashing 1 - n
- Standard object class services
  - Standard objects of the class services 1 - n
- Standard object user defined class 1
  - Standard objects 1 - n of the user defined class 1
- ....
- Standard object user defined class m
  - Standard objects 1 - n of the user defined class m

### **Classification of the Diagnostic Services**

The base variant and the variants of the ECUs contain a list of services of several types. Traditionally, these generic services (i.e. anything except special services, as Read DTCs, Variant Coding, Flashing) have been sorted into one of the following categories:

- Measurements
- Actuators
- Adjustments
- Functions
- Procedures
- Generic Services

The increasing complexity of the ECU data makes it necessary to use another way of categorizing services. Since version 4.0 there is an alternative category system, based on the internally in DIOGENES used classes:

- ACTUATOR
- ADJUSTMENT
- BINARY\_ACTUATOR
- BINARY\_ADJUSTMENT
- DATA
- DATA\_COLLECTION
- DOWNLOAD
- FUNCTION
- STATIC
- SYSTEM
- CYCLIC\_DATA
- CYCLIC\_DATACOLLECTION
- MEMORY\_BLOCK
- GLOBAL
- DIAGJOB
- SECURITY



- SESSION
- STORED\_DATA
- ROUTINE
- IO\_CONTROL

When creating a new system description, it must be decided how to categorize the services. This is dependent on the following parameter in the file Vediamo.ini:

```
[SYSTEMCONFIGURATION]
ServiceTypes = AUTO | ASK | VEDIAMO | DIOGENES
```

The ini parameter can have also the following values (to be changed with StartCenter or a text editor):

- AUTO - In this case the program decides by itself, which categories to use:
  - if the CBF file is created by the old GRIF system, the traditional Vediamo categories are used
  - if the data has been created using the new CANDELA system, the DIOGENES categories are used.
- ASK - in this case the user has to decide, every time a new ECU is inserted into the system
- VEDIAMO - all services are to be contained in Vediamo classes
- DIOGENES - all services are to be contained in DIOGENES classes

The default value is "DIOGENES".

The category system is reflected also in the tree view of the system window of Ecouste.

## 2.3.3 The Functions

### 2.3.3.1 File Menu

#### New File

This function creates a new, empty system description. The left window of the tree structure displays the entry *New System*.

#### Open File

This function accesses an existing system description. The file to be opened can be selected in a file selection window. The file is then read. Before reading the user is prompted whether an automatic alignment of the file content with the used Caesar files has to take place. In case of a positive confirmation, during reading the content is automatically adjusted to match the CAESAR files used. The progress of this action is displayed in the status bar on the bottom of the screen (ECU name, variant name, service

name). If the version of the CAESAR CBF or GBF files has changed since the last file edit, the user will be asked whether the system should update the entries. The system description can be edited subsequently.

### **Save File**

The system description can be saved with this function. If the description has not already been named, a name and the location where the file should be stored must be entered in a file selection window. The filename should be identical to the system ID, otherwise the Vediamo diagnostics server may issue a warning that the filename and the system ID do not match.

### **Save As**

This function saves a previously named file under another name. A name and the location where the file should be stored can be entered in a file selection window. This makes it possible to create multiple copies of an existing system description, and subsequently develop several variations of a system description.

### **Names of Most Recently Used (MRU) Files**

The data menu displays the names of the last four files worked on. These files can be opened by simply clicking on the selected name.

### **Close**

This function closes the system configuration. If the data of the current file has been modified, the user is given the opportunity to save the data before the program is closed .

### **Import / Export**

The *File* menu provides functions to *Import* and *Export* system description in text format.

When using the functions, the appropriate filename must be entered in the file selection window. the file is then read or written in text format.

System descriptions in text format are given the extension `.vst` (Vediamo System description in Text format).

The data appears in the same sequence as in the binary data version (`.vsb`). A caption is created for each object (i.e., ECU, variant, actuator, etc.). The associated entries follow line-by-line. A descriptive text precedes each entry in the respective line.

Example:

```
ECU-----  
ObjectType:6  
BasisVersion:3  
Filter:0
```

Qualifier:CR2  
Description:Common-Rail 2.x  
Version:4

...

DiagPin:0  
DeviceNumber:0  
DriverTypeCode:KLINE  
TrailRequired:0

...

Important:

Working system descriptions in text format carries a high risk for errors because the sequence of data must be strictly adhered to. It can be very difficult to locate errors in the text file.

### 2.3.3.2 Edit Menu

#### Settings

In selecting this menu, the following basic settings can be made:

Output level:

This program creates a file `VediamoSysConfLog.txt` that logs the process and records any potential errors, as well as any access to CAESAR data files using the CAESAR API gateway. The selection of an output level determines the degree of detail of the log information. The following s can be selected:

Module related:

Minimum output

Function related:

The execution of one CAESAR API function is logged.

Internal function details:

Actions during the execution of a CAESAR API are also logged.

Maximum:

All log information made available by CAESAR will be logged. Any errors which occur are recorded regardless of output level setting.

CAESAR directories:

The directories which contain the applied CAESAR CBF-,GBF- und driver data files are specified here. The directory name can be entered manually or selected from a menu.

Display:

In DIOGENES, diagnostic services are entered with a unique ID, called a qualifier. The qualifier serves to uniquely identify a service. Vediamo processes always use qualifiers. Each diagnostic service is also provided with a name. This name is not unique, but does provide the user with a conceptual understanding of the service. One can choose between the display of qualifiers or the descriptive names. Simply select the desired setting.

## **Update**

A checksum is stored for every CAESAR CBF- , GBF-data file used and every Java routine referenced in a system description file. During a diagnostics session, with the help of the checksum, the Vediamo diagnostics server can proofread whether the applied data matches that of the current configuration, and can exclude a system description where appropriate.

This function will update the checksum if changes were made to a CAESAR file or a referenced Java routine. In case a referenced file is not found during the update, the user specifies it using a file selection window. The entire transaction is documented in the log file as well as in the log window.

## **Check consistency**

This function checks the plausibility of the system description content. In doing so, all referenced ECU variants, services, and the assignment of services and Java routines to standard objects are verified. The use of this function can confirm if a referenced service no longer exists as a result of a changed parameterization. The entire transaction is documented in the log file as well as in the log window.

## **Filter all elements positive**

This function always refers to the content of the list on the right side of the main page. With this function, the filter setting of all entries in the list can be set to positive, if filter setting is possible.

When implementing this function an *Include subfiles?* information box will appear with buttons for *Yes*, *No* and *Cancel*. Based upon the selected button, the filter action will or will not be extended to the subfiles, or will be cancelled.

## **Filter all elements negative**

This function always refers to the content of the list on the right side of the main page. With this function, the filter setting of all entries in the list can be set to negative, if filter setting is possible.

When implementing this function an *Include subfiles?* information box will appear with buttons for *Yes*, *No* and *Cancel*. Based upon the selected button, the filter action will or will not be extended to the subfiles, or will be cancelled.

## **Reverse filter settings**

This function always refers to the content of the list on the right side of the main page. The filter settings of all the entries can be reversed with this function, that is to say, a previously positive filtered or unfiltered entry is set to be filtered negative or a previously negative filtered entry is filtered positive.

## **Automatically change error environment data**

Depending upon the parameterization, error environment data can be multiply redundant. This function allows one to control the behavior of the program while it filters the error environment data. While automatic changing is active, it searches for identical error environment data in other errors and variants and modifies it accordingly. The activation of the automatic changing option is designated by a check symbol at the beginning of the menu line. It is turned on and off by clicking on it with the cursor.

### **2.3.3.3 Menu View**

#### **Symbol bar**

This switch turns the symbol bar in the upper part of the screen on or off.

#### **Status bar**

This switch turns the status bar in the upper part of the screen on or off.

#### **Log window**

This switch turns the window displaying the log information on or off.

### **2.3.3.4 Menu Help "?"**

#### **Help topics**

Opens the Vediamo online user manual. Alternatively, the user manual can also be called up by pressing the F1 key.

#### **Information on the system configuration...**

Opens a window with information on the current program version and copyright information.

#### **System configuration entries**

System:

- This entry represents the system description itself.

Functions of the context menus:

Properties: This action pulls up a window in whose general section the following entries can be made:

- The name or ID of the system description. This name also appears as text in the tree structure entry.
- Description: An arbitrary, descriptive text.
- The name of a Java initialization routine for this system. This name can be entered manually or selected from a data selection window. This information is used by the Vediamo Ecoute Client when loading a system: Once the option "Run initialization routine automatically - after system selection" is activated, the listed Java routine is executed after every time the system is loaded. The field can remain empty if no initialization routine should be used.
- The ECU description data (CBF) and the ECU protocol data (GBF) are displayed in the CAESAR portion of the window. These entries are managed by the program and cannot be changed manually by the user.

Other entries under System:

- *ECU*

The individual ECUs that are diagnosed in the system are listed below this entry.

Functions of the context menus:

- New ECU: This function pulls up a selection window containing various ECUs available on account of the used CAESAR data files. Upon selection of a device, a new entry is made for that ECU. This automatically generates the respective variants and service entries with the help of the CBF data files contents. The process of this action is documented in the status bar on the lower end of the screen. Upon selection of a new ECU, a properties window appears in which the following entries can be made:
  - Description: An arbitrary, descriptive text.
  - Driver type: Since the release of CAESAR 2.6, it is possible to parameterize multiple GPD Refs for an ECU. This setting selects the ones to be used by the diagnostic server when the system is loaded, e.g., KLINE, CANLS, etc.
  - Connection number: The number of the CAESAR resource which communicates with this ECU. While loading a system during the run time of the diagnostics server, the attempt will be made to open a channel to the ECU using a resource with this number. If no resource with the given access number is available, then the channel to the ECU must be opened from the client application. Exception: If 0 is entered as the access number (= default setting), the first available resource will be employed to open a channel.
  - Device number: This field is only available in gateway member ECUs. As the description implies, this is where the

device number of the member is specified. For gateway member ECUs, the device number of the gateway is entered as the connection number.

- The option "Shut-down cycle for error deletion" is used when clearing errors during a diagnostics session in Ecoute. If the option is activated, the user is called upon to turn the ignition off and on again after the error clearing process. By default, this option is activated.
- The option "Shut-down cycle for flashing" is used for flashing during a diagnostics session in Ecoute. If the option is activated, the user is called upon to turn the ignition off and on again after flashing. By default, this option is activated.
- The option "Shut-down cycle for coding" is used in coding during a diagnostics session in Ecoute. If the option is activated, the user is called upon to turn the ignition off and on again after the coding process. By default, this option is not activated.
- The field "Maximum number of areas to flash simultaneously" controls how many flash areas in Ecoute a user can change simultaneously and then flash. The default value is 1.
- The field DDLID specifies the ID of the service to be defined dynamically. This specification is only relevant if the ECU protocol supports DDLIDs. The default value is "none", i.e. the diagnostics server does not utilize any DDLIDs for this ECU.
- ID of initialization services for this ECU. The services can be assigned to a dialog window. This information is used by the Vediamo Ecoute client when contact is established with the respective ECU: When the option "conduct initialization services automatically - upon contact" is activated, the entered services are executed during each renewed contact with the respective ECU. Java routines can also be assigned as initialization services in the dialog window.
- In the field "user defined qualifier" an individual qualifier can be specified which is used to identify the ECU, e.g. in Ecoute. The qualifier must be unique, and an in the Diogenes data already existing ECU qualifier must not be used.

Additional entries under ECUs:

- *ECU:*

This entry represents a single ECU. The individual variants of this ECU are listed under this entry.

Functions of the context menu:

- **Delete:** Deletes the respective ECU entry completely from the system description.
- **Properties:** This accesses the same properties window used for adding a new ECU.
- **Duplicate ECU:** This allows the respective ECU entry to be duplicated, creating a second identical entry. The ECU qualifiers are provided with running numbers (1-n) to distinguish between identical entries.

Additional entries under ECU:

*Variant:*

This entry represents a single variant of the ECU. The variant entries are automatically generated by the program based upon the content of the CBF files when a new ECU is added. A variant is a filterable entry. The filter setting can be changed by clicking on the entry's symbol. If a variant is excluded from the diagnosis, the filter setting should be set to negative. The default filter setting when a new ECU is added is positive for all variants.

Function of the context menu:

- **Properties:**  
  
This action pulls up the properties window, with the following settings for the variants:
- **Description:** An arbitrary, descriptive text.
- **The identifier of initialization services for these ECU variants.** The services can be assigned to a dialog window. This information is used by the Vediamo Ecoute client when contact is established with the respective ECU and the variants is specified: When the option "conduct initialization services automatically - upon contact" is activated, the specified services are carried out during each renewed contact with the respective ECU, after any services specified by the ECU entry are performed. If no specific variant is identified, the basic variant entry is used. Java routines can also be assigned as initialization services in the dialog window.
- **ID Block Information:** This function provides a dialog in which services for displaying additional ECU properties can be assigned. These properties are displayed in Ecoute when reading the ID block or in the flash dialog. The left



side of the dialog lists the available diagnostic services that can be used. Once a service is identified, it can be assigned using the "Add" button. It is then added to the list in the right half of the display window. The column "ID" in the list on the right has to be edited by the user. This description is displayed in Ecoute along with the result of the assigned diagnostic service.

Additional entries under variant:

*Filter entries for errors, services and variant codings:*

These entries are filterable entries. By clicking on the entry symbol with the mouse, the filter setting can be set or removed. When a service is excluded from the diagnosis, the filter setting should be negative. The default filter setting for all services in a new application is positive.

For each entry a window containing information can be opened by using the context menu. The description provided by CAESAR is used by default.

An "actuation mode off" service can be specified or selected for all actuator services and all generic services that can be used as actuators in Ecoute. Various DIOGENES parameterized diagnostic services are admitted as "actuation mode off" services.

The actuators and the affected generic services are displayed by a switch symbol in order to distinguish them from the other services in the system description. For services with an entered "actuation mode off" service this symbol is yellow, for all others it is blue. Additional information for all CAESAR/DIOGENES parameterized services, e.g., about preparations/presentations, are displayed in a tree structure.

The following entries can be filtered, given that they are parameterized:

- *Errors in environment data*

*Functions of the context menu:*

*Under the entry "error", preconditions for clearing errors can be entered with the "delete preconditions for error memory". After executing the function, services can be assigned as preconditions in a dialog window. These preconditions are performed by Ecoute before each deletion of the error memory: Basically, the preconditions entered in the basic variants are performed first. When a specific variant is identified, the specific preconditions specified for this variant are carried out subsequently to the preconditions for the basic variant.*

- Measurements
- Actuators
- Adjustments
- Functions
- Procedures
- Generic Services

- *Variant Coding*  
The filterable coding services are listed underneath this entry. Each coding service contains additional subentries, which list the associated coding fragments and their possible values. Fragments parameterized by external data files (ending: .ccf) are identified by the suffix (ex). These subentries are for information only, they cannot be filtered.

Functions of the context menu:

Here pre- and postconditions for the variant coding can be specified in form of ECU services. One or more services can be specified per condition, regardless of which Vediamo class they belong to. Java routines can also be specified. The services or Java routines that are assigned to the preconditions will be performed once upon opening the coding dialog in Ecoute. The services or Java routines that are assigned to the postconditions will be performed once upon closing the coding dialog in Ecoute.

If preconditions or postconditions were specified, two new subfolders, "Preconditions" and "Postconditions" will appear under the entry "Coding" in the tree control. In turn, these subfolders contain the corresponding services. The subfolders only appear when actual preconditions or postconditions are collected.

- *Standard objects*  
The variant-specific standard objects that can be performed by this variant are listed under this entry. Standard objects are assigned the following classes:
  - Services folder:  
Standard objects of class "Services" can be stored here
  - Flashing folder:  
Standard objects of class "Flashing" can be stored here
  - Coding folder:  
Standard objects of class "Coding" can be stored here

User-defined standard object classes can be created using the context menu of the "Standard objects" folder.

New Standard object class: Additional user-defined standard object classes can be created here. A new folder is

created under "Standard objects" for each user-defined standard object.

The folders "Services", "Flashing", "Coding" as well as user-defined standard object classes possess a context menu with the following properties:

- New standard object:  
This function accesses a window in which the name of the new standard object must be entered. As an alternative, the possibility exists to select from a list of current names. A new entry with this name is now created. Subsequently, a properties window is displayed in which the following entries can be made:  
Description: An arbitrary, descriptive text.  
Service: The service hidden behind the standard object must be specified here. These can be Java routines, procedures or functions. It can also specify general services, provided that default preparations are parameterized for them. The service can be entered manually, however it is possible to select from the eligible services.
- Delete: This deletes a standard object.
- Properties: A description of the standard object class can be entered here.
- For standard objects and -classes, in their respective context menu a Copy+Paste functionality is available for duplicating.

The context menu entries for ecus, variants and their filter entries for different services have together two functions "import" and "export" in their context menus. Thereby available functionality has the following characteristics:

- individual selection of the services to be exported /imported in a selection window
- Multiple selection in the Tree View within the ECU, variant and function level
- Export into a table-oriented text file

Export function:

After selection of the export function in the context menu a selection window opens, in which the services to be exported can be selected depending upon their filter information. After selection of the services a file save dialogue is called, in which the name of the file must be specified. Furthermore the ending of the file name must be indicated. The suggested default ending of the file name orients itself to the level of the selection in the Tree View:

- ECU level with ending .vds
- Variant level with ending .vdv (Vediamo data variant)

- Function level with ending .vdf (Vediamo data function)

Then a text file with one line for every service entry which can be exported is created. A line consists of several columns with the following column headings:

- ECU
- Variant
- Function
- Qualifier
- Description
- Filters

Columns are separated by tabulators, thus reading and probably processing of the files e.g. with a spread-sheet program is possible.

Import function:

Over the function "import" in the context menu ECU, variant and function descriptions of the described text format with the three file endings

- .vds
- .vdv (Vediamo data variant)
- .vdf (Vediamo data function)

Can be imported into a system description. After import file selection the dialogue "import" is displayed. Here the user has the possibility of selecting the kind of the services to be imported, depending upon filter information ("all", "positively filtered", "negatively filtered", "new/not filtered"). In addition the following functions can be implemented:

1) Creation of an activity log without following import:

With click on the "activity log"- Button, a change examination over the entire system is accomplished. The changes resulting on the import are listed in a text window.

Additionally, the user has the possibility to store the text window's contents for logging in a file. The file is stored by default under ... \ DATA \ VediamoData \ [current system name] \ yyyy-mm-tt\_hh-mm\_Import\_ [current system name] .log.

2) Actual import of the selected services:

After Selection of the "import" Button the data is actually imported and/or written into the existing vsb.

- *Routines:*

The system specific Java routines that can be performed with this system are listed under this entry.

Functions of the context menu:

- New routine

This function calls up a data selection window in which a Java routine program may be selected. A new entry is created when a Java routine is selected. Upon selection of a new Java routine, a window is displayed in which the properties of the routine can be specified. This window is also available for the initialization routines of systems, ECUs and ECU variants. The following entries can be made:

- Description: An arbitrary, descriptive text.
- Command Line: Command line parameters to be passed to the Java routine when executed.
- Origin path: The complete origin path of the Java routine is displayed here. A button can be used to display a data exchange dialog for changing this path. This entry is for information only. The registered path does not necessarily have to agree with the conditions of the Vediamo diagnostics server at runtime.
- Execute a Java routine synchronously in Ecoute: This controls whether a Java routine from the Ecoute application should be performed in the foreground (synchronous) or in the background. The setting also affects standard objects, in which the respective Java routine is entered. By default, in Ecoute initialization routines are carried out synchronously and all other Java routines are carried out asynchronously.

Each Java routine has an entry under "routines". The context menu of such an entry contains the following entries:

- Delete: This deletes the Java routine from the system.
- Properties: A descriptive text for the Java routine can be entered here.
- *Standard objects*  
The system specific standard objects that can be carried out by this system are listed under this entry.

- Standard objects are assigned the following classifications:
  - Services folder:

Standard objects of class "Services" can be stored here

- Flashing folder:  
Standard objects of class "Flashing" can be stored here
- Coding folder:  
Standard objects of class "Coding" can be stored here
- User-defined standard object classes can be created using the context menu of the "Standard objects" folder.  
New Standard object class: Additional user-defined standard object classes can be created here. A new folder is created under "Standard objects" for each user-defined standard object.

- The folders "Services", "Flashing", "Coding" as well as user-defined standard object classes possess a context menu with the following properties:
- New standard object:
 


This function accesses a window in which the name of the new standard object must be entered. As an alternative, the possibility exists to select from a list of current names. A new entry with this name is now created. Subsequently, a properties window is displayed in which the following entries can be made:

  - Description: An arbitrary, descriptive text.
  - Service: The service hidden behind the standard object must be specified here. These can be Java routines, procedures or functions. It can also specify general services, provided that default preparations are parameterized for them. The service can be entered manually, however it is possible to select from the eligible services.
  - Delete: This deletes a standard object.
  - Properties: A description of the standard object class can be entered here.
  - For standard objects and -classes, in their respective context menu a Copy+Paste functionality is available for duplicating.
- All entries can be edited in the left as well as in the right half of the screen.


## 2.3.4 Working with the System Configuration


- [Creation](#) of a new system description;
- [Editing](#) an existing system description;
- [Updating](#) the contents of a system description;
- [Consistency check](#) of the system description contents;

### 2.3.4.1 Creation of a New System Description


First a new, empty system description is generated using the menu function [New](#) or the  symbol in the symbol bar. Then the properties of this new system are specified:


Place the cursor on the entry "New system" in the left half of the screen, and right click to call up its menu. The function [Properties](#) will cause a window to be displayed in which the name, description and, if applicable, the Java initialization routine can be entered. Subsequently, the ECUs to be diagnosed in this system can be specified. Select the function [New ECU](#) within the entry [ECU](#) of the context menu. After the selection of an ECU, the program will generate the associated variant and service entries. Afterwards, the [properties](#) of the new ECU are specified. By default, the filter settings of all the variants and services of a new ECU entry are set. The filter information can be modified

by clicking on the symbol to the left of the entry. The menu functions [Select all](#), [Delete all selections](#), [Reverse selections](#), [Automatic erroneous data modification](#) and the  symbol in the symbol bar are extremely helpful tools for this. Java routines and standard objects can also be entered, if necessary.

Upon completion, the system description is [saved](#) through the menu function or by using the  symbol on the symbol bar.

#### 2.3.4.2 Working a Preexisting System Description

First, a system description file is opened using the menu function [Open](#) or the  symbol on the symbol bar, or by clicking on the [names of the last open files](#) in the menu. This file can subsequently be worked on as described above.

After the changes are completed, the system description is saved with the menu functions [Save](#) or [Save as](#) or through the  symbol on the symbol bar.

#### 2.3.4.3 Updating the Content of a System Description

A checksum is stored in a systems description file for every CAESAR CBF- , GBF-data file used and every Java routine referenced. During a diagnostics session, with the help of the checksum, the Vediamo diagnostics server can proofread whether the applied data matches that of the current configuration, and excludes a system description if necessary. If referenced data has been changed, the system description must be updated. This is done as follows: [Open](#) the file, execute the menu function [Refresh](#), and then [Save](#) the file.

#### 2.3.4.4 Checking the Content Consistency of a System Description

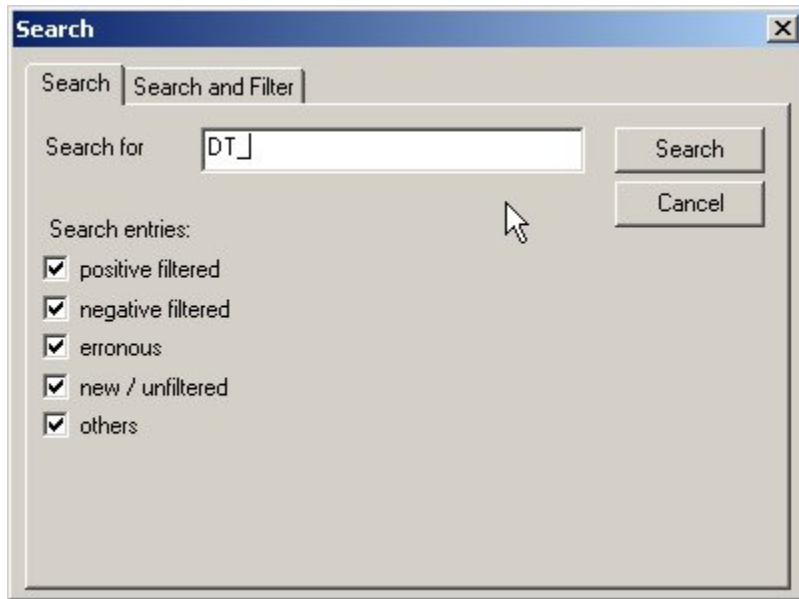
It is possible for a system description to become inconsistent if services are removed through changes in the parameterization, or if their IDs are modified. For example, it might refer to services that no longer exist. It is possible to check the consistency. This is done as follows: [Open](#) the file and execute the menu function [Verify consistency](#).

Errors found during this consistency audit are documented in the log file and in the log window. The filter setting is removed on no longer existing variants and services, and an exclamation mark is used to denote the affected entries. Instances where standard objects are found that reference no longer existent services are also logged. The individual problems can be viewed and corrected based on the record in the log window. The system description file should be saved again afterwards.

#### 2.3.4.5 Search Function

The configuration tool provides a search function with which one can locate names or parts of names in the respective windows. The text search is initiated through the menu entry *Search*, that is found in the menu *Edit*, as is depicted in the following example Ecoute-Client. The *Search* function can also be executed using the key shortcut Ctrl+F.

After selection of the function "Search", the desired text can be entered in a dialog box:



This dialog box remains permanently open (non-modal dialog) until it is closed with "Cancel". After entering the sought after text and activating the "Continue Search", the search is initiated or continued.

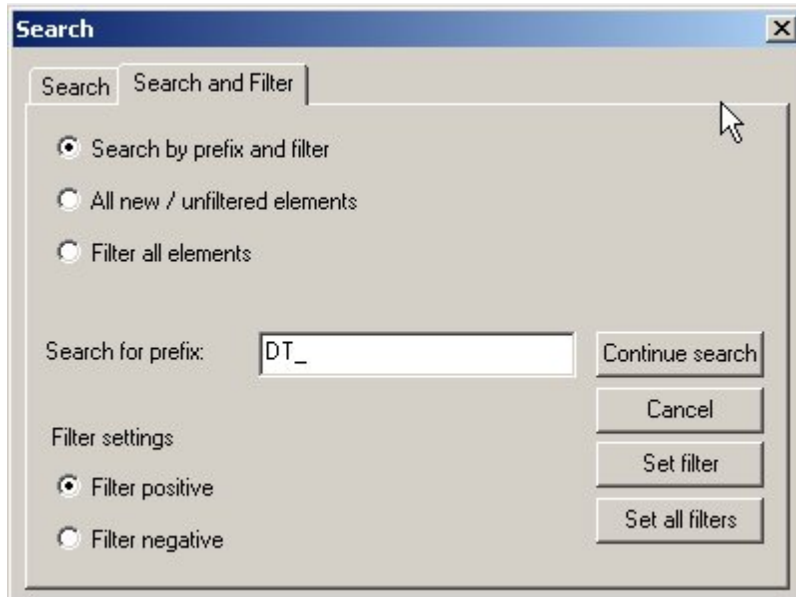
The system tree control is searched in the configuration tool. The search in a window always occurs from top to bottom. Capitalization in the search text is ignored. When a window has been searched to the end, the search is continued from the top of the window. If the search item is not found, an info box appears with a corresponding notification. If the search item is found, the corresponding line of the window is activated. The search can then be continued with the button "Continue Search" in the "Search" dialog.

It is possible to limit the text search to entries with a specific status with the assistance of five options lined up under "Search entries". The entries in question are divided into five groups:

- Positive filtered
- Negative filtered
- Erroneous
- New/unfiltered
- Others: all entries that do not have a filter status, e.g. ECUs, standard object classifications, Java routines, etc.
- In order to search all entries, all options must be marked. This is the default function when initiating the *Search* function.



The register in the upper part of the dialog box: can be used to switch to the *Search and Filter* function.



When the options button *Search by prefix and filter* is active, the specific prefix is searched for, and the found entry/entries is/are appropriately filtered.

If the buttons *All new / unfiltered elements* or *Filter all elements* are active, then the options specially required for the *Search for prefix and filter* are deactivated. By activating *Filter settings*, when the *All new / unfiltered elements* is activated, all new or unfiltered elements in the file are filtered in accordance with the filter setting.

The two radio buttons on the lower left select whether the found entries should be filtered positive or negative.

The button *Set filter* activates the desired filter setting of the current services, and subsequently continues the search automatically.

With the button *Set all filters*, the whole system description is scanned for services containing the specified prefix in the qualifier. Should any be found, the desired filter setting is automatically set.

The search algorithms in the *Search* and in the *Search and filter* differ:

- *Search* checks whether a service qualifier arbitrarily contains the particular string.
- If, e.g., the term DNU is given as a search string, then the services "Test\_DNU\_1" and "DNU\_ABC" would be found..

- With *Search and filter*, the prefix of a service qualifier alone is definitive.
- If, for example, the term DNU is given as a search text, then the service "Test\_DNU\_1" would be ignored, and the service "DNU\_ABC" would be appropriately filtered.

#### 2.3.4.6 Batch Mode

The system configuration can be operated in batch mode in order to automate the updating and checking of system description files. In doing so, the program performance can be guided by command line parameters.

The following command line parameters are possible (optional arguments in corner brackets [ ]):

Filename:

Name of the subject .vsb file. The file will be initially searched for in the actual working directory, and if not located there, then in the directory that is entered in the file `vediamo.ini` under "[SERVER] SystemPfad". This parameter specification is mandatory. [-a]

Refreshing:

When this parameter is given, all checksums in the system description referenced CBF, GBF and Java routine files are re-calculated and re-entered. If an error occurs, an error message is entered in the log file.

Without this parameter, no update takes place.

[-k]

Check consistency:

When this parameter is given, every variant and service is checked for inclusion in the DIOGENES file and classification in the VEDIAMO service classes. If an error occurs, the entry concerned is marked as erroneous and is not included when the file is saved.

Without this parameter, there is no consistency check.

Treatment of new variants and services:

[-n+]

New variants and services in the DIOGENES parameterization are added to the system description and filtered positive, i.e., they are available to the client during the server run.

[-n-]

New variants and services in the DIOGENES parameterization are added to the system description and filtered negative, therefore they are unavailable to the client during the server run.

If neither parameter n+ nor parameter n- is specified, then new variants or services are ignored.

If both parameters are specified simultaneously, an error message is entered in the log file and the program is aborted.

[-t]

Testing:

When this parameter is specified, the system description is not saved after being edited and only the log file is created.

[-l Log filename [+]

Name of the log file, in which the results and error messages are to be recorded. If the parameter "+" is given, then, existing files are not overwritten, but rather all new log information is appended to these files.

If the log filename is not given the log information is treated as in interactive operations and written in the `VediamoSysConfLog.txt` file.

[-s+ Prefix]

Definition: Filter all new services with the given prefix positive.

[-s- Prefix]

Definition: Filter all new services with the given prefix negative.

This filtering affects only newly added services, previously available ones with the same prefix remain unchanged.

The -s command line parameters can be specified multiple times in order to filter various prefixes, e.g.

Example:

-s- DNU -s- WVC

The -s parameters are treated with a higher priority than the others. If, for example -n+ (= new services filter positive) and -s- DNU was given, then a new service DNU\_XXX will be filtered negative, although according to -n+ it should be filtered positive.

No user interaction is possible or necessary during batch operations. Dialog boxes, that indicate errors in interactive operations, are suppressed (sole exception: When serious errors in connection with the log files occur). Errors are documented in the log file instead.

The program is ended when the editing of a system description is completed. In doing so, a return code is generated for the calling process. The following return codes are possible:

0 = Procedure concluded without errors.

1 = Serious errors occurred, e.g.,

- Specified system description data cannot be read.
- Specified system description data cannot be written.
- Unknown/missing/erroneous command line parameters.
- Other errors, e.g. errors during the initialization of CAESAR, parameterization for ECU not found, etc.  
Details can be derived from the log files.
- Errors during the opening/ creating/ writing of the log file.

### 2.3.4.7 Setting of Options Beyond Command Line Parameters

At the start of the system configuration, any keys for the `vediamo.ini` file can be specified in the command line. The specified keys are valid for the duration of the program session.

The command line syntax reads as follows:

```
System Configuration.exe /VI "[Section] key value ... [Section] key value ..."
```

The options `/VI` or `-VI` serve to distinguish between `vediamo.ini` keys and other (batch mode) parameters. The list of keys to be overwritten must be contained in quotation marks.

#### Directories with `cbf` files

When the configuration tool starts, analog to the diagnostics server, those `.cbf` files that are entered in the directory found in the `vediamo.ini` file listed under "[CAESAR] CBFPFAD" are applied first.

If a system description file should be opened that is not included in the current system directory, then a check is made to determine whether `.cbf` files have been stored in this directory. If yes, then as with the diagnostics server, the current `.cbf` files are unloaded, and the `.cbf` files in the new directory are loaded.

The menu entry *Edit / Return to the standard system path* resets the `.cbf` path to the default value, e.g., to create a new system description.

## 2.3.5 Special Features

### 2.3.5.1 Service Filters

Services can be filtered in various ways. The filtering of services occurs through settings in [vediamo.ini](#).  
[\[Ecoute\]Global Filter](#)

Services, which qualifier begins with these prefixes are filtered negative (providing that they have not already been filtered out by [CAESAR]UseServiceTypes.

The negative filtering can be cancelled by selecting the appropriate commands in the menu.

#### [\[CAESAR\]UseServiceTypes](#)

If this entry carries the value "STANDARD", then services such as

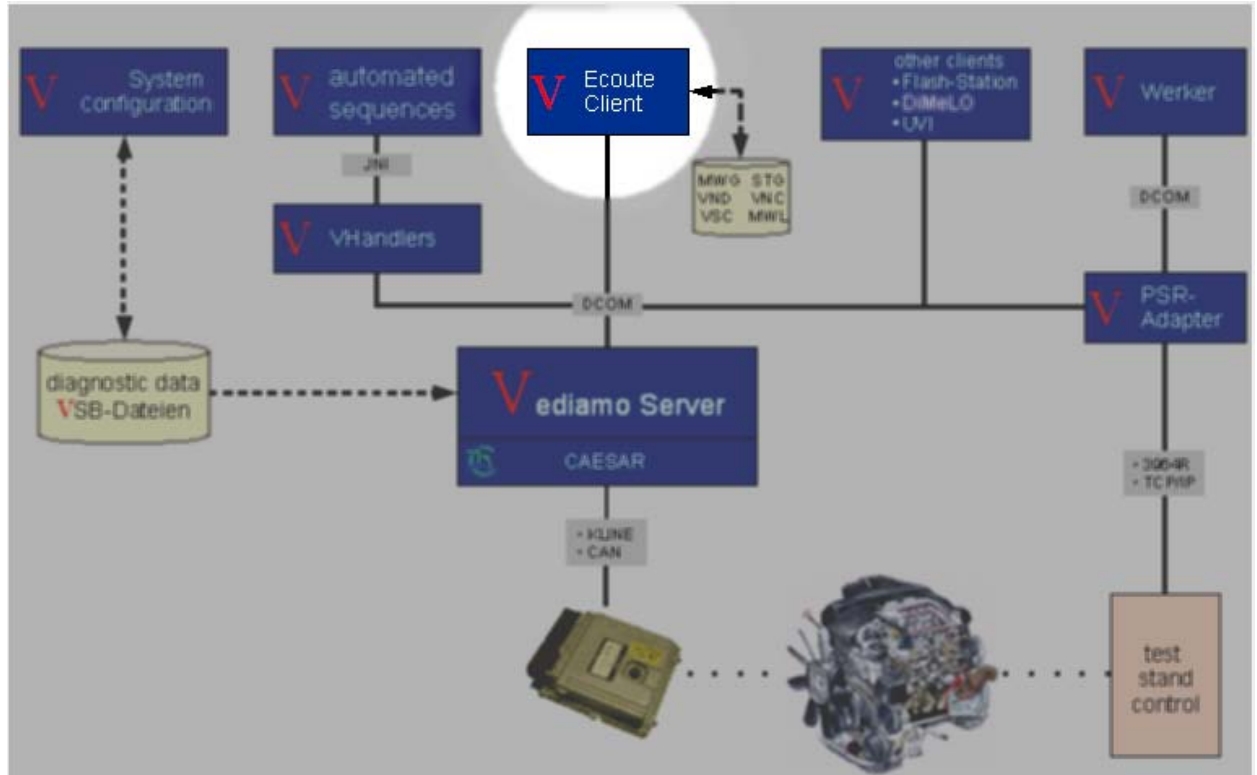
- DST\_SYSTEM
- DST\_ENVIRONMENT\_DATA
- DST\_GLOBAL, DST\_NEGRESP
- DST\_BINARY\_ACTUATOR\_INP
- DST\_BINARY\_ADJUSTMENT\_INP

will not be accepted in the VSB. They are treated as if they did not exist.

### 2.3.6 Configuration (INI Parameters)

The system configuration possesses only 4 parameters in the INI file, all of which deal with the storage of logs. Details can be found in the [INI editor](#).

## 2.4 Ecoute



## 2.4.1 Introduction

The Ecoute application provides the user with all available functions of the diagnostic server through a comprehensive interactive interface. There are specialized windows for every task that can be configured, adapted and arranged as desired. Customized solutions for special, non-standard tasks can be implemented using Java programs and integrated in the program.

On account of its scope of functions, Ecoute serves for taking ECU systems into operation in the engine test facility as well as for testing ECUs in a laboratory setup or for vehicle diagnostics (e.g. quick test).

A short description of the interface can be found in the chapter [Interface Structure](#). The files which Ecoute requires are summarized in the chapter [The Ecoute Files](#). A subsequent detailed description of the functions and actions is included in [The Ecoute Functions](#).

Finally, the [Ecoute menus](#) as well as the [keyboard commands](#) are described.

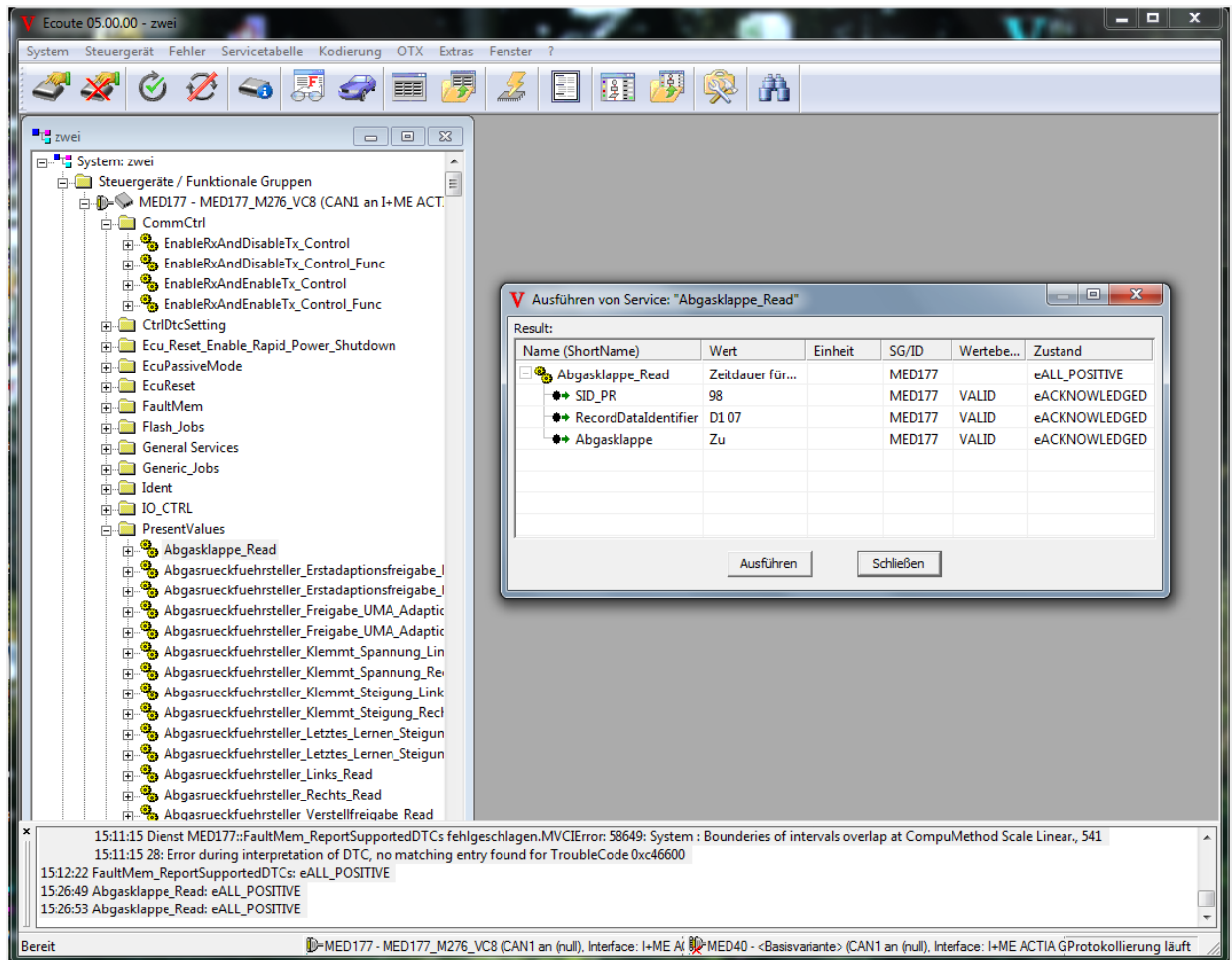
The Ecoute user interface supports different languages. The language is specified by the *Language* entry in the [COMMON] section of the [vediamo.ini configuration file](#). The languages German (DE) and English (EN) are supported in Version 3.1.

### Requirements for Ecoute Operation

In order to use Ecoute to establish communication with ECUs, the following requirements have to be met:

- Correct [installation](#) of Vediamo as well as the CAESAR hardware drivers.
- The installed and registered server must have the same version as the Ecoute used. This is automatically the case after correct execution of the installation program.
- The CAESAR parameterization required for the ECU.
- Correct physical connection between tester (diagnostic server with CAESAR hardware) and ECU.
- [CAESAR](#) hardware power supply (CAESAR Part D and Part P - eCOM in particular must be supplied with external power)
- ECU power supply. Some ECUs require power on the ignition line in addition to the operating power.
- Availability of a Vediamo system description which includes the ECUs you like to diagnose. Use the [Vediamo system configuration](#) to generate a system description. If no system description has been generated yet, you can also perform diagnostics on an ECU with the Vediamo system description (VSB). For every ECU included in the CAESAR CBF file, Ecoute provides a system of the same name for diagnostics.
- A license for at least one of the implemented CAESAR hardware components.

## 2.4.2 GUI Structure



### Ecoute User Interface

An overview of the most important Ecoute windows is provided by the following table:

#### Ecoute Windows

Selection Window	<ul style="list-style-type: none"> <li>• Display of selected ECU system</li> <li>• Display of ECU names / names of ECU variants</li> <li>• Display of ECU connection to CAESAR hardware</li> <li>• Display of standard objects (selected services with special functions such as unlocking, ECU</li> </ul>
------------------	--

	reset, etc.)
Output Window	<ul style="list-style-type: none"> <li>• Display of status messages</li> <li>• Display of results from functions / Java routines, etc.</li> </ul>
Status Line	<ul style="list-style-type: none"> <li>• Explanation of the currently selected menu entry</li> <li>• Buttons for displaying and changing the ECU contact status</li> <li>• Display of log files</li> </ul>
Measurement Window (Service group)	<ul style="list-style-type: none"> <li>• Display and examination of measurements with the following functions: <ul style="list-style-type: none"> <li>○ Textual (table) and analog (bar diagram) value display</li> <li>○ Time dependance (curve diagram)</li> <li>○ Recording of series of measurements</li> <li>○ Analyzing recorded series of measurements</li> </ul> </li> </ul>
Controller Group (*)	<ul style="list-style-type: none"> <li>• Display and operation of controllers / adjustment <ul style="list-style-type: none"> <li>○ Name of ECU to which the controller / adjustment applies</li> <li>○ Name</li> <li>○ Actual value</li> <li>○ Units of controller</li> <li>○ Difference controller / adjustment. An adjustment is designated by (*)</li> <li>○ Control units for changing the status of the controller / adjustment</li> </ul> </li> </ul>
Error Window (*)	<ul style="list-style-type: none"> <li>• Display of ECU errors with the following information: <ul style="list-style-type: none"> <li>○ Error code (e.g. B1470)</li> <li>○ Error text</li> <li>○ Information whether error is current or not</li> <li>○ Information whether error is stored or not</li> <li>○ Information whether <u>MIL</u> is turned on</li> </ul> </li> </ul>



	<ul style="list-style-type: none"> <li>○ Display of error environment data with following information:</li> <li>○ Name of error environment value</li> <li>○ Description (e.g., Geberbruch)</li> <li>○ Units of error environment value</li> </ul>
Variant Coding (*)	<ul style="list-style-type: none"> <li>• Display and selection of name of service for variant coding</li> <li>• Display and selection of coding fragments</li> <li>• Display and selection of coding fragment values</li> <li>• Coding string (hexadecimal and decimal)</li> </ul>
Flash Window (*)	<ul style="list-style-type: none"> <li>• Display and selection of flashware (area, meaning, FlashKey)</li> <li>• Start of ECU flash process</li> </ul>
Trace Window (*)	<ul style="list-style-type: none"> <li>• Display of communication data between PC and ECU (as bytes or as blocks)</li> </ul>
(*) Not included in figure	

## The System or Selection Window

The system or selection window has a special function. It displays all ECU and their diagnostic services in an hierarchical tree structure, as well as possible standard objects and Java routines. Each element can be activated by a double click (or <Enter>) (ECU: contact is established or terminated, services and Java routines: execution is started).

For each object in the tree structure which has an explanation, this explanation can be displayed in a properties window (right mouse key - context menu - properties).

For ECUs, this takes place in the [properties window](#). For services and Java routines, this is a simple window with the most relevant information on the service / Java routine.

## Sorting the Services

The diagnostic services of the ECU are broken down into several groups. Traditionally, you find the following groups of services:

- Measurement - services which serve to read a value from the ECU

- Adjustments - allow fixed settings for certain ECU parameters
- Controller - similar to adjustments, however the set values are not stored permanently in the ECU.
- Functions - carry out an action in the ECU and deliver a result
- Procedures - like functions, but deliver no result
- General services - all other services.

Alternatively the services can be categorized in groups corresponding to the DIOGENES service types. The way of classifying services is determined by the system description file. This might be especially useful when using newer ECU data, which becomes increasingly complex.

Important:

The parameters set by adjustment services are permanently changed in the ECU. The values will **not** be reset to defaults by switching off the ignition.

### Filters for Services

Services can be filtered in different ways (i.e., masked in the system window display). Filtering of services is done by settings in the `Vediamo.ini` and/or in the system configuration:

- [\[Ecoute\]UseFilters](#)
- [\[Ecoute\]GlobalFilter](#)
- [\[CAESAR\]UseServiceTypes](#)

The different configuration possibilities and the expected results are explained in the following:

[CAESAR] UseServiceTypes	[ECOUTE] UseFilters	VSB/DIOGENES System	Expected Result
(don't care)	0	VSB System	All services contained in the VSB are displayed unfiltered, regardless of the filter characteristics (as set in the system configuration) of the service.
(don't care)	1	VSB System	Only those services contained in the VSB and positively filtered there are displayed
ALL	0	DIOGENES System	All services contained in the DIOGENES system (CBF) are displayed.

STANDARD	0	DIOGENES System	<p>The service types:</p> <ul style="list-style-type: none"> <li>• DST_SYSTEM</li> <li>• DST_ENVIRONMENT_DATA</li> <li>• DST_GLOBAL, DST_NEGRESP</li> <li>• DST_BINARY_ACTUATOR_INP</li> <li>• DST_BINARY_ADJUSTMENT_INP</li> </ul> <p>are not displayed.</p>
ALL	1	DIOGENES System	<p>Services whose qualifiers begin with the prefixes included under[ECOUTE]GlobalFilter are not displayed.</p>
STANDARD	1	DIOGENES System	<p>The following services are not displayed:</p> <ul style="list-style-type: none"> <li>• Service types: <ul style="list-style-type: none"> <li>○ DST_SYSTEM</li> <li>○ DST_ENVIRONMENT_DATA</li> <li>○ DST_GLOBAL, DST_NEGRESP</li> <li>○ DST_BINARY_ACTUATOR_INP</li> <li>○ DST_BINARY_ADJUSTMENT_INP</li> </ul> </li> <li>• Services whose qualifiers begin with the prefixes included under[ECOUTE]GlobalFilter</li> </ul>

Despite the general relevance, working without a selection window can make sense. If you generally use only pre-prepared measurement and controller windows, you can close the selection window (menu *Window / System Window* or CTRL-S). This state is stored in the configuration (vediamo.ini). The next time Ecoute is started, the window remains closed. It can be opened again any time using the same menu or keyboard command.

### The Status or Output Window

This window serves to output the results of all executed actions such as system selection, execution of services, etc. This window can be kept closed as well in order to allow more room for other windows.

### Other Windows

The remaining windows are described in the respective subsequent sections.

### 2.4.3 The Ecoute Files

The DiagServer is responsible for the administration of diagnostic data and system data. There are additional files, however, which are useful especially for Ecoute. These are:

Name	Suffix	Explanation
Session	VSC	The state of Ecoute is stored in this file and can be restored during the subsequent start-up. It contains the size and position of the main window as well as of other windows (MWG, STG, Errors, etc.), the selected system and the contact status to ECUs.
Service Group	VSG	Groups of measurements (in XML format) to be displayed in a measurement window, can be defined for each system. The file contains the position and size of the window and its diagrams along with the services selection and parametrization (min/max, color a.o.)
Measurement Recording	VSR and BIN	The measurements in a measurement window can be recorded and saved in two files with the same name: the VSR file contains information about the service group, the BIN file contains the numeric values of the series. The files must not be edited by the user.
Manual Command Input	VND	Message and communication parameters are stored here for communicating via CAESAR API-I.
Snapshot	TXT HTM(L)	These files are only written by Ecoute. They contain the contents of data windows which can be stored if desired and used for documentation purposes. Either as pure text or as HTML.
Log Files	LOG	Different log files can be written as needed.

### 2.4.4 The Ecoute Functions

The most important functions of the Ecoute client are described in this section.

[Select and Close System](#)

[Contact ECU](#)

[Execute Service](#)

[Readout Error](#)

[Quick Test](#)

[Examine Measurements and set Controllers/Adjustments](#)

[Variant Coding](#)

[Flashing](#)

[OBD2](#)

[Configure Ecoute and Server Options](#)

[Macros](#)

[Java routines](#)  
[Routine Generator](#)  
[Standard Objects](#)  
[Display Trace and Monitoring Data](#)  
[Manual Command Input](#)  
[CAN Bus Simulation](#)  
[Snapshot File Storage](#)  
[Simulation of ECU Communication](#)

All functions can be accessed via the corresponding menu items. Some often used functions can also be accessed using the corresponding buttons on the toolbar. The toolbar can optionally be shown or hidden.

#### **2.4.4.1 Select System**

You select the system to be diagnosed either from the menu *System / Select...* or using Alt-A. The dialog "Select system" appears and lists the available ECU systems. The system selection list control indicates for each entry whether it is VSB or CBF based. The related information is displayed in a second column of the list (CBF or VSB). Two buttons allow you to specify whether only the systems based on one system description file, or only the systems based on the DIOGENES parameterization, or both types of systems should be displayed in the selection list. The button *Change Directory* allows you to change the directory in which the diagnostic server looks for the system descriptions and the corresponding DIOGENES data. A new folder can be selected in the dialog which appears after the button is pressed. The diagnostic server subsequently updates the affected DIOGENES data. The progress of this action is shown in the status window. Next, the system selection window is updated. Select the system to be diagnosed and confirm with OK. A message appears in the status window stating that a new system has been selected. The selected system appears in the selection window as a tree structure.

With the menu option "set standard system path" the default value of the current system description and `-.cbf` path can be set, e.g. after a change of the system folder.

Alternatively to the manual selection of a system using the system selection dialog, the name of the system to be loaded can be entered as a command line parameter when Ecoute is started. This system is then loaded immediately after the server has been initialized. It is possible, e.g., to generate entries for frequently analyzed systems in the `StartProfile.txt` file so that the StartCenter starts Ecoute with the names of these systems as parameters.

Using the button "Auto system selection", an ini-file containing predefined ECUs can be selected. The application then tries to establish contact to every ECU defined in the selected file. If an ECU with a valid diagnostic variant could be detected, and a system

description containing such an ECU is available, this system description will then be highlighted in the system selection list box.

Warning:

Opening and closing a system affects DiagServer. If other clients are active at the time, the system changes affect them as well.

#### 2.4.4.2 Close System

A system can be closed using the menu selection *System / Close*. All windows displaying ECU-relevant data (system, error, measurement, trace) are then closed.

The system is automatically closed if:

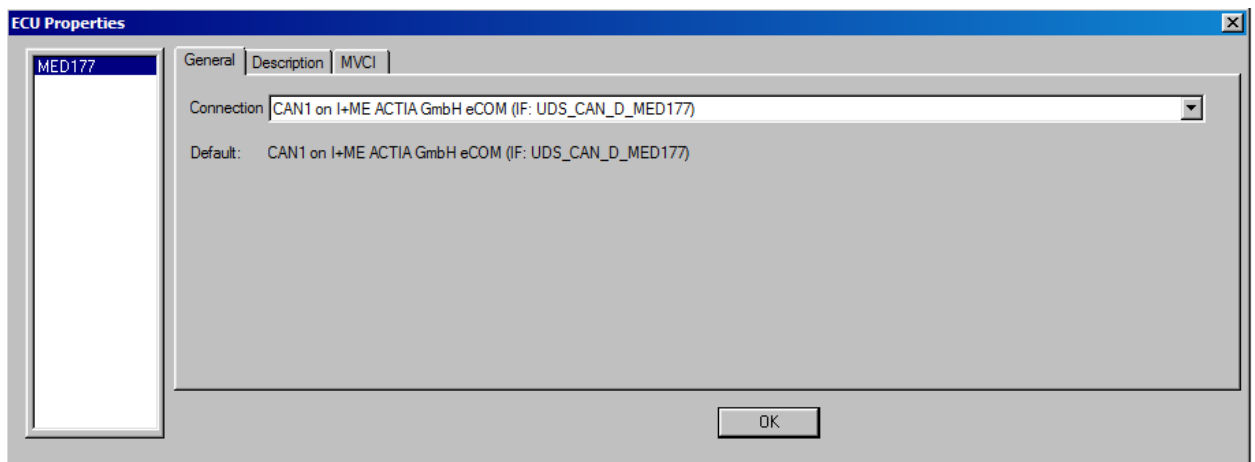
- Ecoute is ended and no further clients are using the server
- A new system is selected
- The DiagServer is re-initialized (server reset)

#### 2.4.4.3 Contact ECU

In order to establish contact, a resource (connection) must first be assigned to the diagnostic hardware.

Ecoute provides a combined ECU selection / properties dialog which allows, among other things, the assignment of connections on the CAESAR hardware to ECUs.

#### ECU Properties Window



This dialog can be called up either by making the menu selection *ECU / Properties*, or by clicking on an ECU in the selection window with the right mouse key and then select the context menu entry *Properties*.

On the left, the dialog contains a list of all available ECUs currently loaded in the system. On the right, among other things, a selection list for the connection is displayed. It shows the currently assigned connection for the ECU selected in the ECU list. In the example above, the ECU "SIM266" is assigned to the connection CANHS No. 1 on CAESAR Part Y. The drop-down list contains all currently available contacts (which are not assigned to any other ECU) and the entry *<none>*. If *<none>* is selected, the currently assigned connection is released so that it can be assigned to other ECUs.

For example, if the current selected system contains the two ECUs ECU1 and ECU2, and ECU1 is on K-line 4 and ECU2 is on K-line 5, but the two should be switched, proceed as follows:

- Select ECU1 and connection *<none>* in order to release connection K-line 4.
- Select ECU2. The previously released K-line 4 shows up in the connections list.
- Select K-line 4 for ECU2. This releases the previously taken K-line 5.
- Select ECU1 and assign K-line 5.

The changes take effect immediately after every step, not when exiting the dialog with *OK*. It is possible that a change may need a few seconds to take effect.

### Using the Multiplexers

The CAESAR hardware makes more connections available than there are communication channels. The channels can be switched between different connections using the multiplexer (Part C or E). The following scenario is possible: The computer contains CAESAR Part A. One Part B2 and Part E are connected to it. This hardware configuration makes two channels available. To connect two ECUs (ECU1 and ECU2) to K-line 1 and K-line 2, proceed as follows:

- Go to the drop-down connection list. There can be, for example, 7 possible K-lines which could be assigned to the two channels.
- Assign ECU1 to K-line 1, for example.
- K-Line 1 no longer appears in the drop-down list. 6 possible K-lines are shown.
- Assign ECU2 to K-line 9, for example.

If ECU1 K-line 1 should now be switched to K-line 7, proceed as follows::

- Go to the drop-down connection list for ECU1. **Since both available channels are now already being used, the remaining 5 K-lines do not appear in the list!** Select connection *<none>* for ECU1. This releases a channel.
- Go to the drop-down connection list for ECU1. Now six possible K-lines are listed again.
- Select K-line 7 for ECU1.

When all channels are taken, therefore, and one wishes to multiplex a channel onto another pin, it is first necessary to release the channel. (<none> connection). A new pin can subsequently be assigned.

## Gateway Member ECUs

For gateway-member ECUs, "GW member *x* on *ECUy*" is displayed. *x* represents the device number and *ECUy* the name of the gateway ECU.

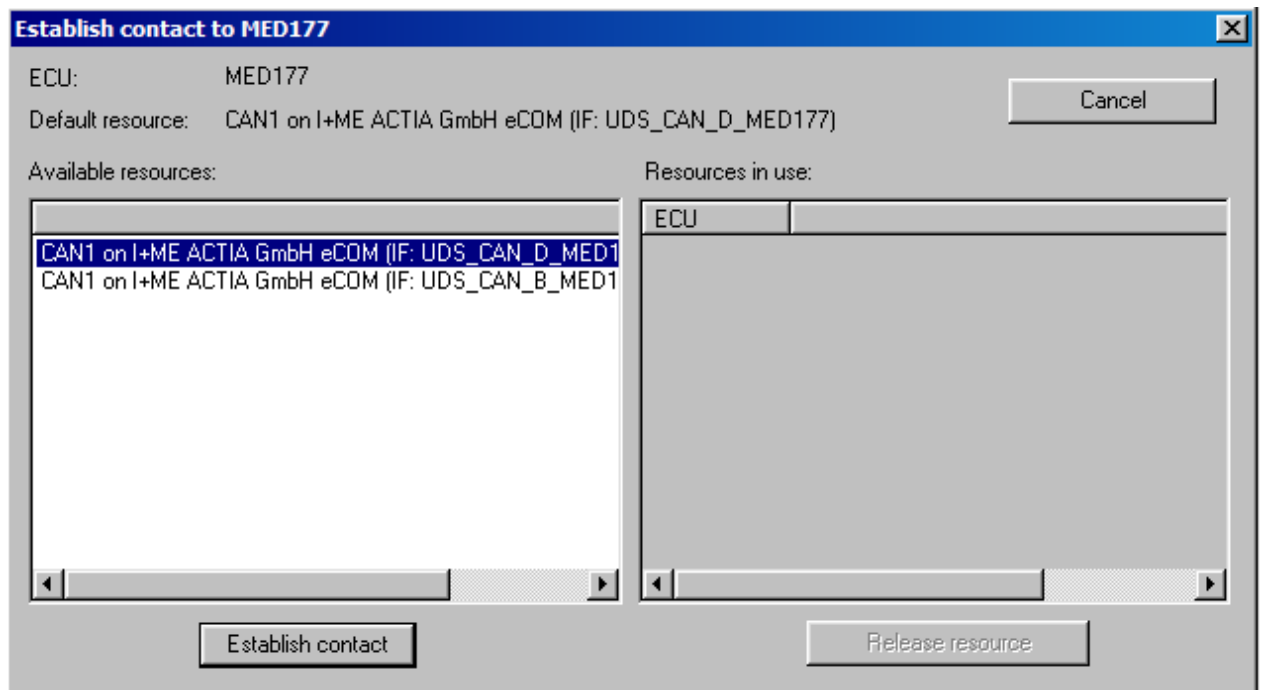
## "Description" Tab

The parameterized description of the basic variant and the current variant of the selected ECU are displayed under this tab in the *ECU Properties* dialog. The part number as well, if it is parameterized.

## "MVCI" Tab

The version of the CBF file belonging to the selected ECU is displayed under this tab in the *ECU Properties* dialog.

Wenn Kontakt zu einem Steuergerät ohne aktuell zugewiesene Ressource (Kommunikationskanal) aufgenommen werden soll, dann wird folgender Dialog angezeigt:






## Establish Communication with the ECU

The contact to an ECU can be established or terminated with the following commands:

- Function key F3 (establish) and F4 (terminate)
- Menu *ECU / Contact...*
- Clicking the ECU button in the status line
- Selection window: Double click on the ECU symbol or with right mouse key and context menu

Please observe the ECU symbol in the selection window and on the button in the status

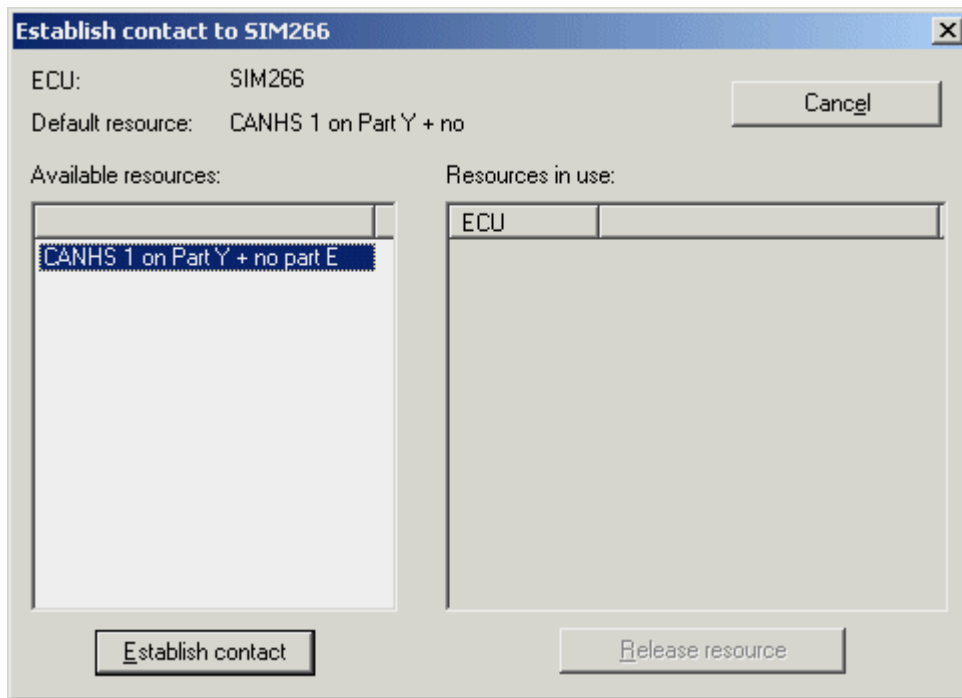
line: It should look like this before communication is established: . As soon as

communication is established, the connector is connected: .

If an ECU variant could be recognized, the variant name is displayed following the name of the ECU in the selection window once communication is established. If no variant could be recognized, you have the possibility of automatically checking whether the variant is parameterized for another ECU.

You can also establish communication with an ECU using the [manual command input](#).

If contact should be established to an ECU without currently assigned resources (communication channel), the following dialog is displayed:



The current available resources for establishing contact with the ECU are displayed in the list on the left. The default resource from the VSB is, if included in the list, already marked. If all the possible resources for the ECU are already assigned, the list is empty. In the list on the right, the possible resources - including the ECUs using the resources - are displayed. The default resource from the VSB is, if included in the list, already marked here as well.

The user can now (if no free resource is available) select an ECU from the list on the right from which he wishes to release a resource. If a resource is released, the current status of the available resources is immediately updated in the first list.

To establish contact, the user selects one of the available resources from the list on the left and subsequently presses the *Establish contact* button.

### **Read ID Block**

Select the menu entry *Read ID Block* in the ECU menu to read the ECU ID block (see also [keyboard operation](#)).

The following is displayed in a separate window:

- MB number
- Hardware version
- Software version
- Supplier
- Diagnostic version

- ECU ID
- (Diagnostic) state, e.g., "Series production"
- Production date

In addition, defined additional information is displayed in the corresponding system description. The information is fixed in the system configuration as follows: For every ECU variant, any services which provide the user with additional information on the ECU can be collected and assigned in its properties dialog (system configuration, function *ID block Information*). In order to display this information, Ecoute requests the ID of the services entered in the system configuration from the ECU and then executes them.

The service ID (e.g. "Software-Stand", "Code", "Boot-Block", etc.) is then shown in the left column in the Ecoute ID block window. The result of the corresponding diagnostic service is shown in the right column.

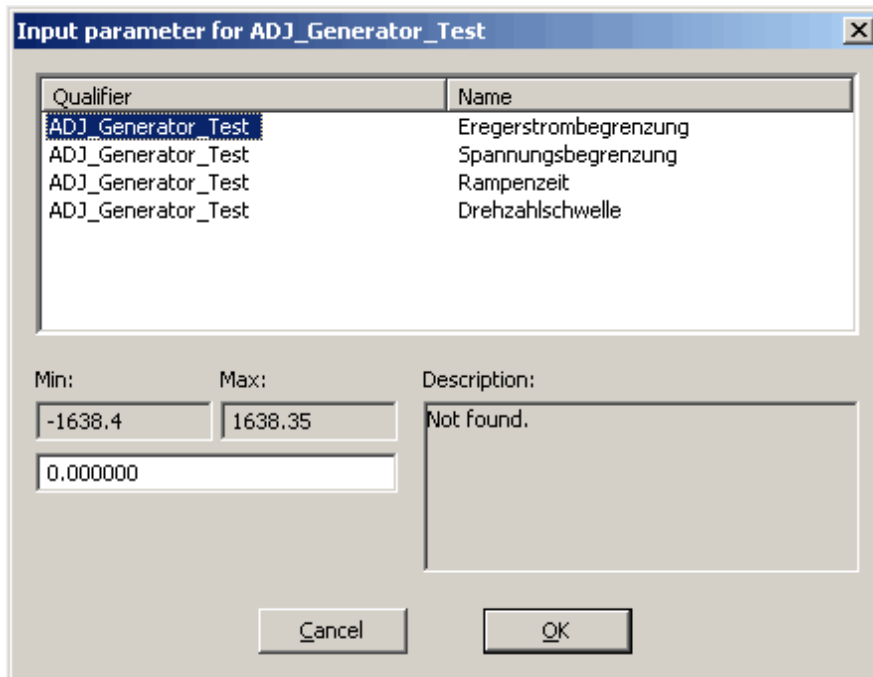
The content of the ID block window can be stored in a snapshot file using the button *Store*.

The window can be closed with ESC or the *Close* button.

#### **2.4.4.4 Execute Service**

Ecoute provides the possibility to directly execute services interactive from the selection window. The service simply has to be selected in the system tree structure (selection window) and started by double click or pressing the RETURN key. The result is displayed in the status window (output window).

If the selected service has input parameters ("Preparations" in DIOGENES terminology), a dialog is displayed before execution which prompts the user to input one value per parameter.



Important:

The execution of services might permanently change ECU parameters. The permanently changing services are adjustments, variant coding, flashing and some others.

As any diagnose service, also a Java routine can be executed by double clicking it in the system window. By using the context menu (right click), item *Execute with command line params*, the Java program can be started with any parameters in the command line. In the system description default params can be set for every Java routine.

#### 2.4.4.5 Read Error

After [communication](#) with the ECU has been established, its errors can be read. ECU errors can be read once or cyclically.

If a measurement group with the filename `Fehler_Mit_Protokoll.mwg` exists in the system directory, this measurement group is opened automatically when the error window is opened and is linked to the error window ( the actions *Read*, *Read cyclic*, *Save* and *Close* apply to both windows).

- Select *Error / Read errors* (F6 key) from the menu. If the system has multiple ECUs, a dialog appears requesting to select the ECU of which the errors should be read.

- An error window appears after the ECU has been selected. The window is labeled with "Errors" and the name of the ECU as well as the name of the ECU variant (or <Basic variant> if no variant could be recognized).
- After the window has been opened, the errors are read from the ECU. This process can take time if there is extensive error data. The errors are then displayed with code, text, current information, stored information and MIL (malfunction indicator light).

The two-part error window opens when *Read errors* is called:

Code	Text	Current	Stored	MIL On
0060	Steuergeraetefehler Sicherheitskonzept Ueberwachungsfehler		*	
0022	Saugrohrdrucksensor Diagnose	*	*	
001B	Drosselklappenpoti 1 Diagnose	*	*	
001C	Drosselklappenpoti 2 Diagnose	*	*	
002F	Kuehlwassertemperatur Diagnose	*	*	
0029	Temperaturfuehler Ansaugluft Diagnose	*	*	
00B1	Nachrichtenunterbrechung Kombi	*	*	
noAR	Nachrichtenunterbrechung ESP	*	*	

Environment of 0060	Description	Unit
Fehlererkennung wurde seit letzmaligen Fehlerspeicher loeschen durchgefuehrt (Readyness)	Ready	
Diagnosebedingungen erfuellt	On	
Fehlererkennung war im aktuellem Fahrzyklus moeglich	On	
Fehler gespeichert	On	
Pending Fehler	On	
Fehler im aktuellem Fahrzyklus vorhanden	-	
vorgemerakter Fehler	On	
Fehler bestaetigt	-	

Buttons and selection options are displayed in the upper section of the error window. The errors and error code ("Code"), error text ("Text") and the appropriate flag "Current", "Stored", or "MIL On" are displayed below the buttons and selection options. If one of the error flags is set for a certain entry, it is denoted by an asterisk ("\*") in the respective error flag column.

The error environment data with the

- name of the environment value ("Name"),
- description ("Description"),
- and unit ("Unit")

are displayed in the lower window section. The error environment data for the error selected in the upper window section is displayed.

### Logging Additional Information - Linked Measurement Group

You have the possibility to establish a group of measurements which are automatically displayed with errors and logged, if necessary. This is especially practical for test drives

during which the test driver can determine all important information by pressing a single key, rather than open several windows and click through them.

Generate a regular old-style measurement group and store it in a file named `Fehler_Mit_Protokoll.mwg`. As soon as a file is available for the selected system, it is automatically read during error reading, as well as stored when storing a snapshot. The measurement group is not displayed on the screen in the process.

A short description of the error window buttons and selection options:

### **"Clear" Button:**

This button can be used to delete ECU errors. If necessary (entry *shut down cycle required* in the VSB), the user is prompted to turn the ignition off and on again after the shut down cycle.

The content of the error window is subsequently refreshed on demand (dialog box with query).

### **"Read" Button:**

The ECU errors can be read once and displayed in the error window with this button. It is a pushbutton that does not lock when pressed. Alternatively, the ECU errors can be read once by pressing the "F6" key.

### **"Read cyclic" Button:**

The button *Read cyclic* has two states. In the <pressed> state (button is displayed recessed), errors are read from the ECU cyclically.

On opening the error window, the button is in the state <not pressed> (button is displayed elevated). This means the errors are not read cyclically.

### **"Save to file" Button:**

The *Save to file* button [saves](#) the current error information in a file. If a linked measurement window is open, its content is saved as well.

### **"Save cyclic" Button:**

The *Save cyclic* button [saves](#) the ECU errors cyclically in a file. The button can only be activated when the errors are being read cyclically.

Besides the buttons, the options *Current*, *Stored* and *MIL On* can be selected in the upper section of the error window:

**"Current" Option:**

This determines whether or not errors with the "Current" flag are displayed.

**"Stored" Option:**

This determines whether or not errors with the "Stored" flag are displayed.

**"MIL On" Option:**

This determines whether or not errors with the "[MIL On](#)" flag are displayed.

**"All" Option:**

This option determines that all errors from the ECU are displayed, independent of the error flags "Current", "Stored", and "MIL On". The default setting is for the option "All" to be selected, so that all errors can be displayed.

**"Envir. Data" Option:**

When *Envir. Data* is activated, errors are displayed with environment data. If the option is deactivated, no environment data is displayed. The lower section of the window in the example is then not displayed. When the option *Envir. Data* is deactivated, errors without environment data are read from the ECU for communication protocols (e.g., KW2000) which allow errors without environment data to be read.

**"Filtered" Option:**

It is possible to filter out errors or error environment data in the Vediamo system configuration, i.e., filtered out errors or error environment data are not displayed in the Ecoute application. The option *Filtered* allows the user to determine whether the filter mechanism for displaying errors and environment data in Ecoute should be applied or not.

**"Delay" Parameter:**

This parameter can be used to slow down cyclical error reading. The set time (in seconds) means an additional pause between read cycles. The complete cycle therefore is equal to the set time in addition to the time required for the one-time data transmission.

**Delete ECU Error**

The menu entry *Delete error* makes it possible to delete directly from the menu without opening an error window.

If preconditions for deleting errors are given in the system description for the respective ECU variant, these are performed before deleting the error memory.

The selection "Delete error with log" performs three tasks in a row:

- Read error (error window is opened)
- Store error in snapshot file
- Delete Error

### "Options" Button:

Through the introduction of the communication protocol UDS, additional possibilities for configuring the *Read errors* process became necessary which can also be used for other protocols.

The *Options* button in the *Read errors* window is for configuration. The function *Read errors* can be configured in the dialog displayed when the button is pressed.

It is possible to set:

Again:

- The delay time during cyclical reading
- The flags used during error reading

In addition:

- If reading or deleting errors should take place on a group basis, a group can be specified here.

The currently used protocol is displayed at the very top.

The button *Advanced* leads to a further dialog in which the individual protocol specific parameters for error reading can be selected.

To make configuration easier, all five input boxes are realized as "combined list fields" (= comboboxes).

The lists are loaded with the constants predefined in CAESAR, e.g., for the field "Group":

Under UDS:

Example:

EEG\_UDS\_EMITTIONS

EEG\_UDS\_POWERTRAIN

EEG\_UDS\_CHASSIS

etc.

Under KW2000:

**Example :**



EEG\_KW2000\_POWERTRAIN  
EEG\_KW2000\_CHASSIS  
EEG\_KW2000\_BODY  
etc.

The user can now select the values entered in the lists for the configuration or can manually enter numerical values.

These values are used as the default values for the current error reading session when the button "Apply" is pressed. By pressing the button "Save" the current settings are stored permanently in the file Vediamo.ini as standard settings. By pressing "Standard", the internal stored default settings are selected, which can be applied or saved in the following.

### **"Show ROE" option:**

This option button is located right at the outer edge of the error window.

A requirement for the use of ROE (Response On Event) light is, that on a change in the status of any error the ECU puts this information with DTC number on a defined CAN ID.

If the option is activated and ROE light is supported by the ECU and the related diagnostic data, then a message about the change in status of a DTC will displayed in a popup window named "ROE events" . On the first occurrence of a DTC event the window will be opened, if it is still closed. Further events are displayed in a new line in the same window. The user can close the window. On occurrence of a new event, the window is reopened.

#### **2.4.4.6 Read Permanent Errors**

Since Ver. 4.1, this function is not supported any more. It is contained in the [OBD2 functionality](#).

#### **2.4.4.7 Error Reading by Status**

Executed by the KW2000-specific function REQUEST SUPPORTED 2BYTE HEX DTC AND STATUS (\$03) , referred to as "\$18 \$03" in connection with Vediamo, or "Error reading by Status".

The function is supported only by the KW2000 protocol (and analog to the UDS protocol). Although it is possible for an application to get the name of the currently used protocol by way of CAESAR, but it is not possible to determine whether the protocol is derived from KW2000 or UDS base on this name (e.g.KW2C2PE, KW2C3PE, etc.). It is therefore the responsibility of the user to use the function in a sensible manner.

A dedicated window is displayed after the menu entry is activated.

A button with the error number is displayed in the window for every error. The color of the entry depends on the error status. A legend is displayed in the window explaining the color code. When an entry is selected using the cursor, the corresponding error text is displayed in the lower section of the window. If not all the window entries fit in the window, the window can be scrolled.

All supported error codes are read by Status once or cyclically, by clicking on the buttons *Read* or *Read cyclic*, respectively, and the window updated accordingly.

A dialog for entering the name of the file to be stored is called up using the button *Store in file*. The format of the log is identical to the previous format generated by the Java routine of the same name.

The actual error reading with the KW2000 function \$18 \$03 is currently not supported in a satisfactory manner by CAESAR or the available DIOGENES parameterization.

This function is therefore implemented for KW2000 in the Vediamo server using CAESAR-API1 calls.

For this purpose Vediamo has certain minimum requirements on the ECUs used:

- Getting the total number of provided error codes with \$18 \$E0 must be supported.
- During the block-by-block, sequential reading of the error codes with \$18 \$03, every read block should contain the number of error codes contained in the block.

The button "Permanent DTCs" starts reading permanent DTCs. All codes found in the list of permanent DTCs gets displayed with a "P" beside the code.

For a better overview during the measurement and the subsequent evaluation, additional services (measurements) to be executed by the ECU can be specified.

The measurements can be defined in the Systemconfiguration, similar e.g. to the ID block extensions.

The services are executed before reading the error codes and are then displayed in the "read errors by status window" and in the protocol file.

#### **2.4.4.8 System Quick Test**

The function *Quick test* makes it possible to read and/or delete the errors of all current addressable ECUs (e.g., for a complete vehicle).

If the command line parameter `-κ` is entered when Ecoute is started, this function is called automatically after the program start.

For the quick test, the required DIOGENES data (.cbf) must be available in a "quick test directory":

The quick test directory containing the DIOGENES data can be specified in the `Vediamo.ini` file under the entry:

```
[ECOUTE]
ShortTestDataDir=
```

A standard information file is used to specify the models and ECUs to be diagnosed. This file contains information on all models available to the user. Models are maintained in the information file with different respective engine ECUs. The default name for the file is `KurztestMercedes.ini`, the content is structured, e.g., as follows:

Example:

```
[Models]
NumberOfModels=3
Model1=BR211
Model2=BR220
Model3=BR203
```

```
[BR211]
NumberOfECUs=5
```

```
;these are the (engine) ECUs from which the user
;must select one for the quick test
OptionalECU=1,3
```

```
;ECUx = <ECU name>, <Connection>,
; <Pin with Part E/Pinmapping>,
; <Pin without PartE/Pinmapping>
ECU1=ME28, CANHS, 1, 1
ECU2=ZGW211, CANHS, 1, 1
ECU3=CR3, CANHS, 1, 1
ECU4=ARMADA, KLINE, 6, 1
ECU5=SAMH, KLINE, 11, 1
```

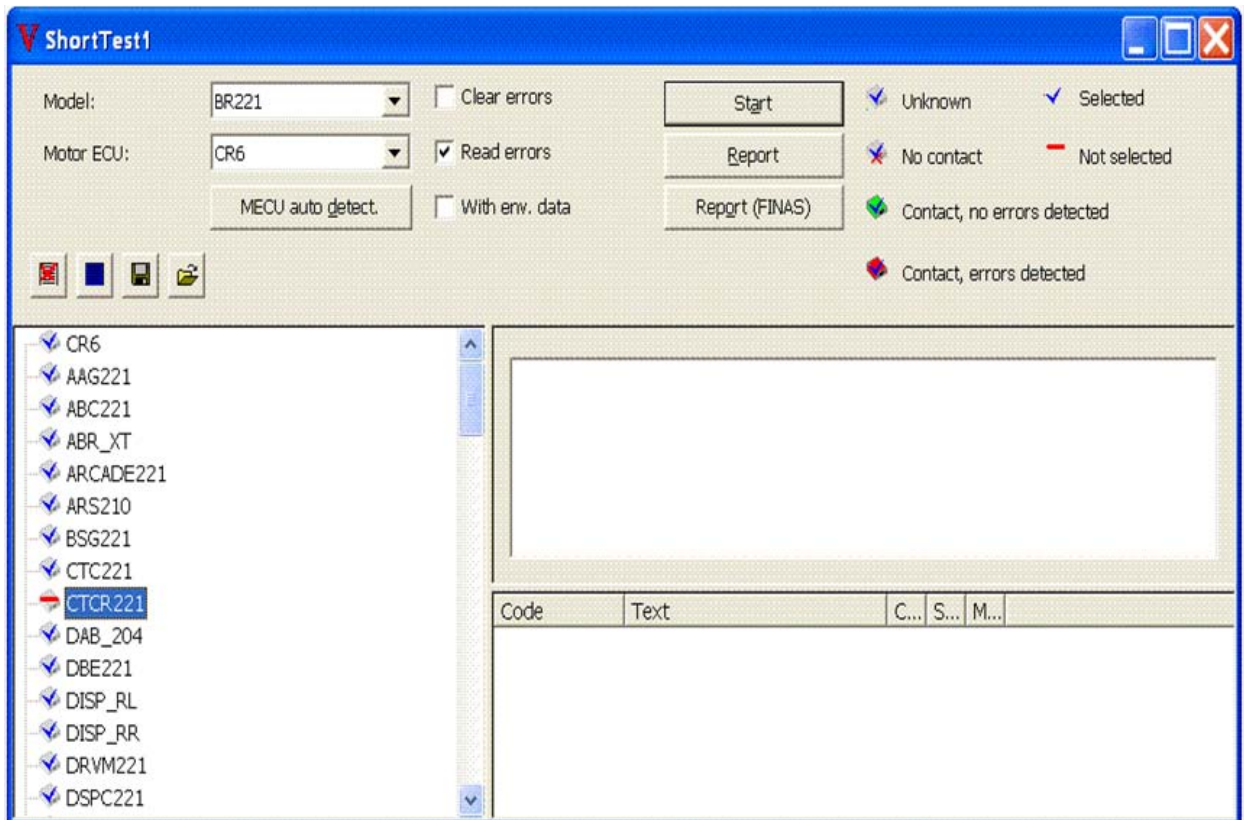
```
[BR220]
```

```
...
```

```
[BR203]
```

```
...
```

At the start of the quick test, the user can select the information file (\*.ini) to be used. The following window is displayed after the file has been selected:



The user can select a model here, and then one of the engine ECUs corresponding to the model.

By pressing the button "MECU auto detect." a dialogue is opened which tries to automatically access all engine ECUs registered for the current model. If one is detected (with a diagnostic variant unequally the base variant), then this engine ECU can be selected and be further used.

After selection of the model / engine ECU, the ECUs of the corresponding model configuration are displayed in the left section of the window.

The following additional settings can also be made in this window:

- Read errors
- With environment data
- Clear errors

The *Start* button is then used to begin the test.

For individual specification of ECUs to be considered in quicktest, there are still the following additional options:

Individual selection of one or more ECUs. The selected ECUs are diagnosed, not selected ignored.

There is a possibility to select/deselect all ECUs with a single mouse click, and to save or load the last selection configuration. The selection of ECUs of a model is done by a "check-box" for every ECU in the result window (selection by tick or stroke directly on the symbol of the ECU in the tree) and four buttons with the functions "Deselect all ECUs", "Select all ECUs", "Save filter configuration", "Open filter configurations".

Function of button "Deselect all ECUs":

When you press this button all ECUs are deselected for the quicktest.

Function of button "Select all ECUs":

When you press this button all ECUs are selected for the quicktest.

Function of button "Save filter configuration":

Allows you to save the selection status of the ECUs. When you press this button, a file-selection dialog is opened to specify the name of a file to save the selection state info in text format. The file is by default with the extension .Vkk The information is stored in the following format (example):

[General]

Version=1.0

Model=BR204

MotorECU=CR6

SelectedECUs=5

ECU1=CR6

ECU2=CGW\_204

ECU3=CLAMP15

ECU4=CTRLC\_204

ECU5=DAB\_204

Function of button "Open filter configurations":

When you press this button, you are prompted to select a file containing a quicktest configuration (.vkk). The contained filter information, where possible, is used to select/deselect ECUs of the currently selected series.

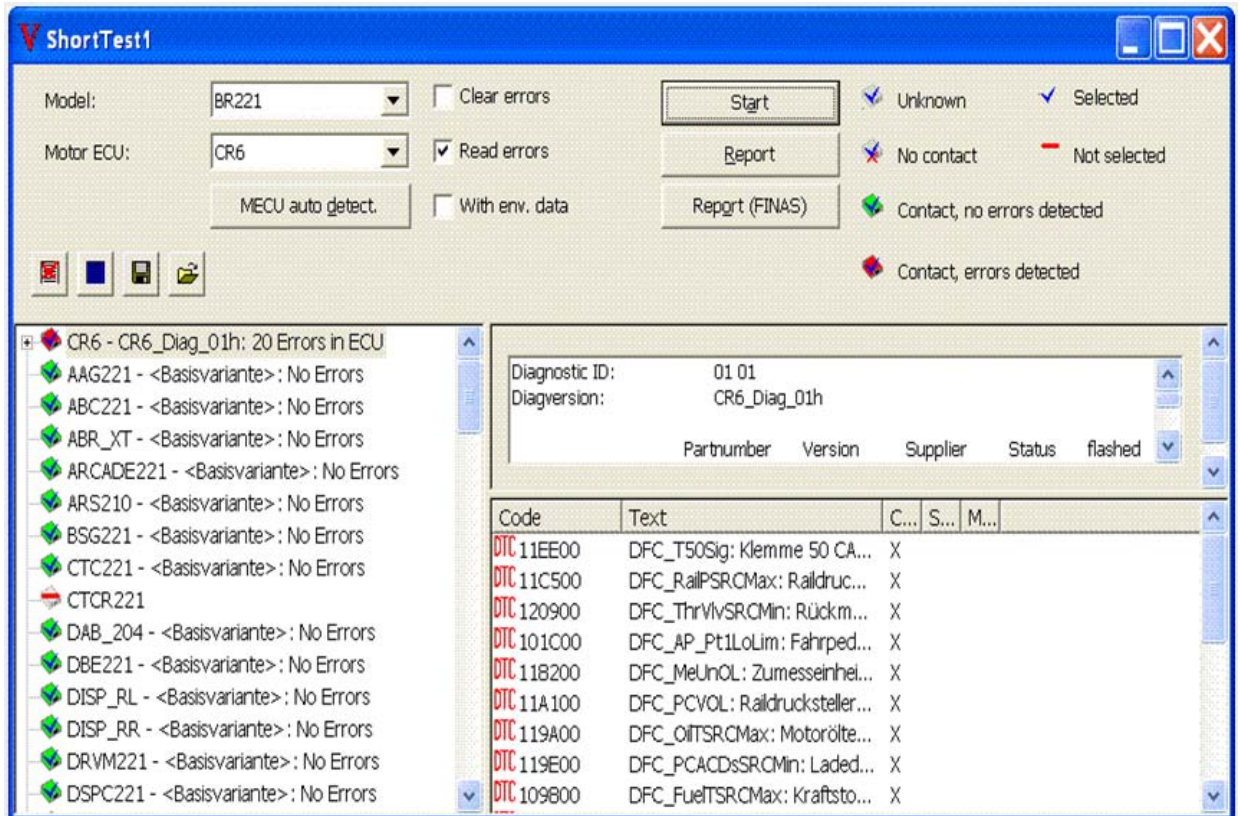
A check is run first to see if a DIOGENES parameterization is available. In case of an inconsistency, the quick test is aborted with an error message.

The actual test process is started otherwise.

An attempt is made to establish contact with every ECU included in the model configuration using the CAESAR resource given there. If contact is established, the errors are read and/or cleared depending on the corresponding settings. The test progress

is displayed in a progress indicator in the quick test window, additional information is displayed in the status window.

Upon completion of the quick test, the results are displayed in the quick test window:



The result tree on the left contains the first level of all ECUs involved in the quick test. It is filled out when the model and engine ECU are selected. The status is displayed using icons. There are four states:

- Unknown (gray symbol): The ECU has not been activated yet, e.g., because no CAESAR resource was available.
- No contact (gray symbol with red "X"): Contact attempt with ECU failed, or no variant was recognized, or the error reading failed.
- No DTC (green symbol): Number of DTCs / Events = 0
- With DTC (red symbol): Number of DTCs/Events > 0

The initial state of an ECU is <unknown>.

If an ECU has DTCs/Events > 0, it is displayed in the result tree with sub listings. These are the DTCs, which numbers are displayed in the tree.

During the test, the state changes from unknown to one of the other states. However, the sub listings do not automatically drop down.

Using the right mouse key over an ECU entry opens a window in which additional information on the respective ECU is displayed.

The section to the right of the result tree:

This section displays the environment data of the error selected in the result tree. If no error is selected, or the ECU is selected, the corresponding ECU ID information is displayed to the right of the result tree, with the error window below. No environment data is shown in this case.

When an error is selected in the result tree, the error number, error text and state (MIL, current, stored) are shown in the section on the right above the environment data.

The quick test result can be logged in a file by clicking on the *Report* button.

If a file named `ECU-Name-Mapping.txt` exists in the Vediamo quick test data, the CBF names in the quick test are supplemented by the plain text names (in the status window, the window with the quick test results, and in the quick test logfile).

An example with the list content:

Example:

[ECU-Name-Mapping]

EWM211=Electronic transmission selector lever module for BR211

TSG\_V\_L211=Front left door control unit for BR211

Presentation in quick test:

EWM211 - Electronic transmission selector lever module for BR211

TSG\_V\_L211 - Front left door control unit for BR211

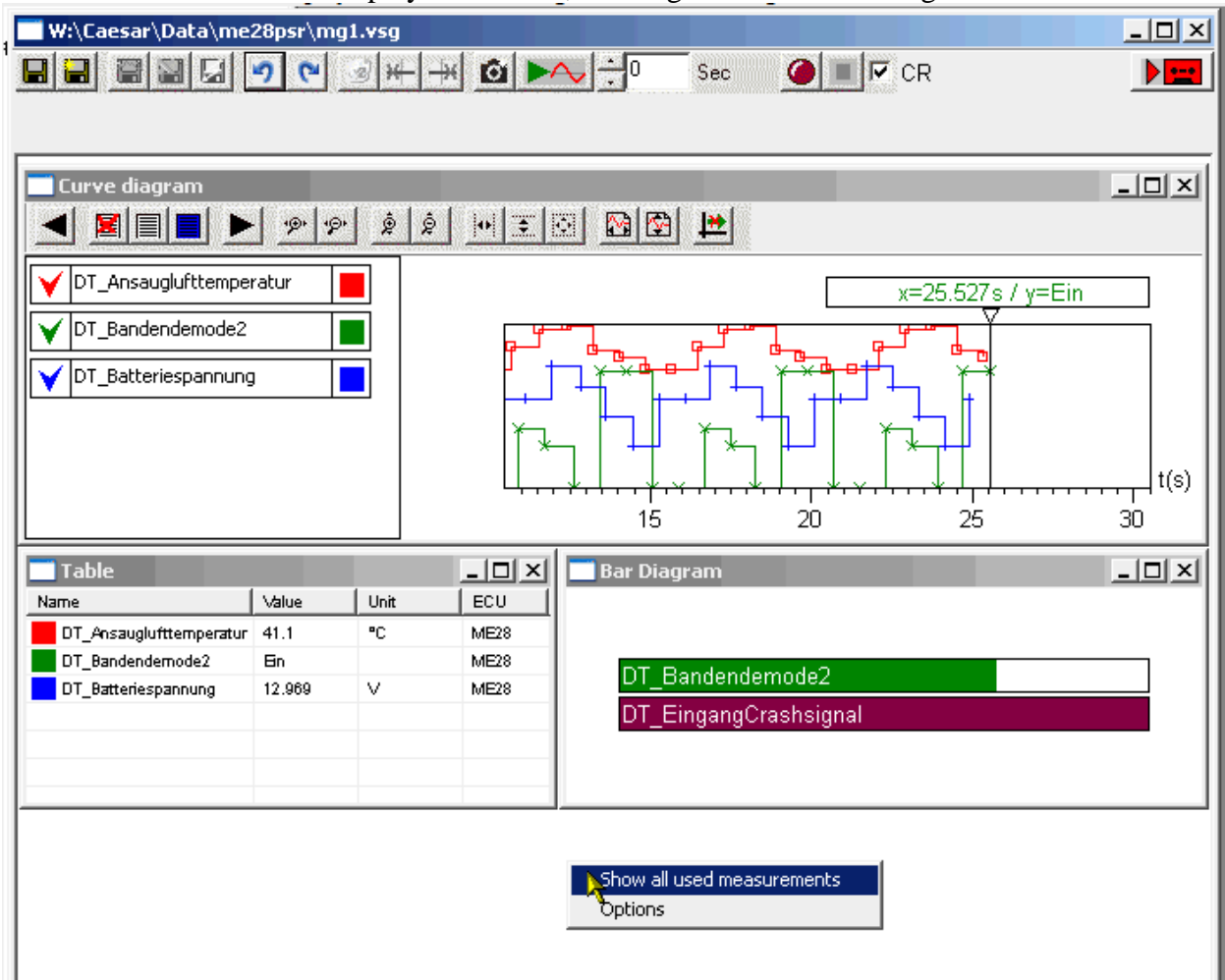
#### **2.4.4.9 Service Groups**

Measurement and other services can be executed by a double click in the system window. But for more advanced experiments a service group window can be defined with a selection of services to be displayed and controlled in diagrams of several kinds: tables (textual), bar diagrams and curves for measurement values, and an actuator window for other services. The service group, defining the properties and content of the window is stored in a VSG file.

To create a new service window, select *Service Groups / New Group* from the main menu. If VSG files are available for the current system, select *Service Groups / Open Group* to load a file.

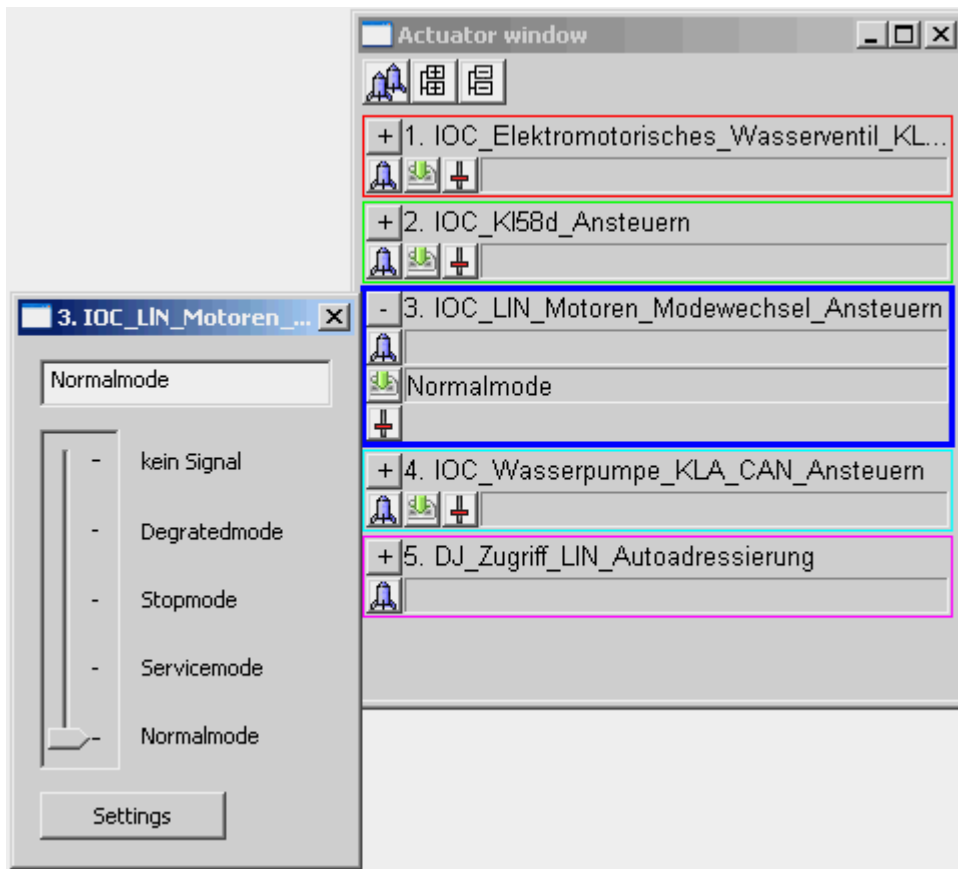
When inserted into the window, any service gets a color assigned to for a better orientation. The service is represented in any of the diagrams by the same color.

The measurements can be displayed in tables, bar diagrams and curve diagrams:



To execute other kinds of services than measurement values, the actuator window can be used:





In this diagram every service gets those kinds of controls which are necessary to handle it. The controls of one service are surrounded by a color frame. Use these controls to change input params, execute the service and read OutputRef measurement (the measurement service delivering the actual state of an actuator), if present.

### Different Handling of Services In Different Diagrams

A service inserted from the system window into a measurement diagram (table, bar diagram or curve) will be treated as a measurement service, i.e. it will be executed during every single shot or a reading cycle, and the result will be displayed in the diagram. Textual return values will be mapped to consecutive numbers, to make it possible to display them in a curve. For example, "On" will be mapped to 1, "Off" to 0. If the automatic mappings do not fit your need, you can change the mapping table in the options of the service.

A service inserted into an actuator window will be treated in a different way. It is regarded as an element changing the state of the system. The execution of such services is only possible by using the "Execute" button in the actuator window.

Nevertheless, you can drag & drop a service from the actuator window to a measurement window (table, curve, bar diagram). In this case the measurement window gets several new entries:

- one or more input parameters, marked "<P>"
- the service execution itself, marked "<S>"
- if present, also the OutputRef service of the actuator

When the service is executed (in this example: the actuator) by means of the actuator window controls, the measurement window(s) display the following events:

- one measurement value per input parameter
- the result of the execution
- the result of the OutputRef service, executed directly after the service itself.

If you do not need all of those informations in the diagram, you can delete any of it separately.

**Important:** The other way round - inserting a service into a measurement window and then into an actuator window, is not possible.

## Controls of the Measurement Window

The controls of the measurement window are placed in a toolbar. Some of them might be invisible or deactivated, depending on the current state of the window.



The function of the controls is (from left to right):

- *Save* service group file (VSG). All changes to diagrams, services, positions etc. will be stored in the file.
- *Save* service group as a new file.
- *Save* measurement series. If measurements have been recorded, you can store them in a pair of files (VSR and BIN). If no recorded values are available, the button is inactive.
- *Export* recorded values into a CSV file (comma separated values), which can be p.e. imported into Excel or a database. If no values available, the button is inactive.
- *Save* current values of all measurements into the snapshot file.
- *Undo* change (up to 10 steps). This restores the last state before an important change, like adding/deleting diagrams or services, deleting recorded data etc.

Simple resize or repositioning of windows does not count as a separate change which can be undone.

- *Redo* change. Undone changes can be made again.
- *Delete* recorded values from memory and diagrams. If recording was running, it will be stopped after a confirmation by the user.
- Delete recorded values up to the current position of the time slider.
- Delete recorded values starting from the current position of the time slider.
- *Single shot*. Read all measurements once.
- *Cyclic reading* start/stop. When cyclic reading is active, all services are cyclically executed. After pausing the cycle, the time axis of curve diagrams continue to move and the cycle can be reactivated or single shots can be done.
- Setting the *delay* (in secs) for cyclic reading. The cycles can be executed as fast as possible (delay 0) or every X seconds, as entered in the control. The delay cannot be changed when reading is in progress.
- *Record*. Since this moment, all values are recorded in memory and can be saved in file at a later time. When the check button *Recording starts cyclic reading* is checked, cyclic reading will be started automatically.
- *Stop*. This button stops recording. If cyclic reading was active, it will be stopped also. The window changes the mode into *Analysis mode*: reading measurements is blocked, you can analyse the recorded values, change display options and save the measurement values into VSR/BIN file pair.
- Check button *CR = Cyclic Recording (Recording starts cyclic reading)*. Set this check mark if you wish an automated start of cyclic reading when starting recording. Otherwise after pressing the record button you have to use the *Cyclic reading* or the *Single Shot* button.
- State of the window. If the mode is *Recording* or *Analysis*, you can see a tape cassette resp. an eye symbol.

(2nd row, from left to right)

- Current position of the time slider. It is visible only when there are measurement values in memory. If you enter a float value, the display moves to the measurement point lying nearest to the specified time (starting with 0 sec).
- Time slider. With this slider you can conveniently choose the display point. You can also use the cursor keys to change the time position.

## Keyboard Usage

You can access the most important functions during measuring also by the keyboard.

- Blank (space) - start / stop cyclic reading
- Return (Enter) - start / stop recording. If automatic saving is not activated (see configuration, below), after stopping recording you will be prompted for a file name to save the measurement series. If you just press Enter a 3rd time, the series will be saved with the default name (name of the window + date + time)

- Delete - clears the memory. Any recorded values are erased from memory. If the recorded data has not been saved up to now, you will be asked if it should be done now. Just press ESC to delete without saving, or Enter to confirm and Enter again to accept the default file name.
- Shift-Del and Ctrl-Del - delete the recorded values left or right from the current time slider position.

## The Modes of the Measurement Window

The window has 4 modes / states:

- **Configuration mode** (Free Running Mode)  
In that mode you can freely configure the window: everything including adding and deleting diagrams and measurements. You can also read and display measurements in the current configuration, but you cannot store them in a file (VSR/BIN or CSV).
- **Recording mode** (red tape cassette symbol in the right corner)  
In this mode all read values are recorded temporarily in memory. No changes are allowed that would make the values inconsistent (i.e. you cannot add new measurements, but you can add measurements already used in the window). If you delete a measurement from a diagram, it will still be included in the internal measurement list, even if it is not visible in any diagram. To see all measurements used, select *Show all used measurements* from the popup menu (right click on the empty window area).  
To stop recording mode, press the *Stop* button or the *Delete* button.
- **Analysing mode** (eye symbol in the right corner)  
You get this mode after recording measurements. Now you can neither change the content of the measurement list (i.e. add new measurements), nor read values from the ECU. As long as this mode is active, the recorded values are consistent. You can change anything affecting displaying the values: min and max values, colors, diagrams and their positions etc.  
The recorded values can be saved in a VSR/BIN file couple, which includes also the diagrams, their parameters, colors etc, or in a CSV file, which can be used in other tools, like Excel or databases.  
To leave this mode, you have to explicitly delete the recorded values or start a new recording. In both cases you will be prompted for confirmation, if the recorded values have not been saved yet.
- **Analysing only** (eye symbol in the right corner)  
The difference between this and the previous mode is just that you get in the *Analysing only* mode by loading previously saved measurement recordings (VSR/BIN file couple). Because you have values from a file without a connection to a real ECU, it is not possible to switch to *configuration* or *recording* mode. You can change the display options (colors, line thickness a.o.) and save the

measurement series in a new VSR/BIN couple, export to a CSV file, or extract a service group definition file (VSG).

## **Configuration of the Service Group Window**

Before starting an experiment or measuring session, an appropriate service group has to be configured. Most important is the list of measurements to be read and the delay factor to determine how often the services have to be read.

You can insert several diagrams into one measurement window and fill them with the same or different services. Any service used in the window will be executed always once during one cycle, and its result is sent to all diagrams displaying it.

To fill diagrams with services, simply drag & drop them from the system window or another window containing measurements. Alternatively you can mark services and copy & paste them.

Important:

When no contact to the ECU is established, in the system window you can display all variants and their service qualifiers. However, it is not possible to insert into the service window any services which are not defined in the current ECU variant. To insert services from any ECU variant, start Vediamo in [simulation mode](#) and select the variant which services you wish to insert.

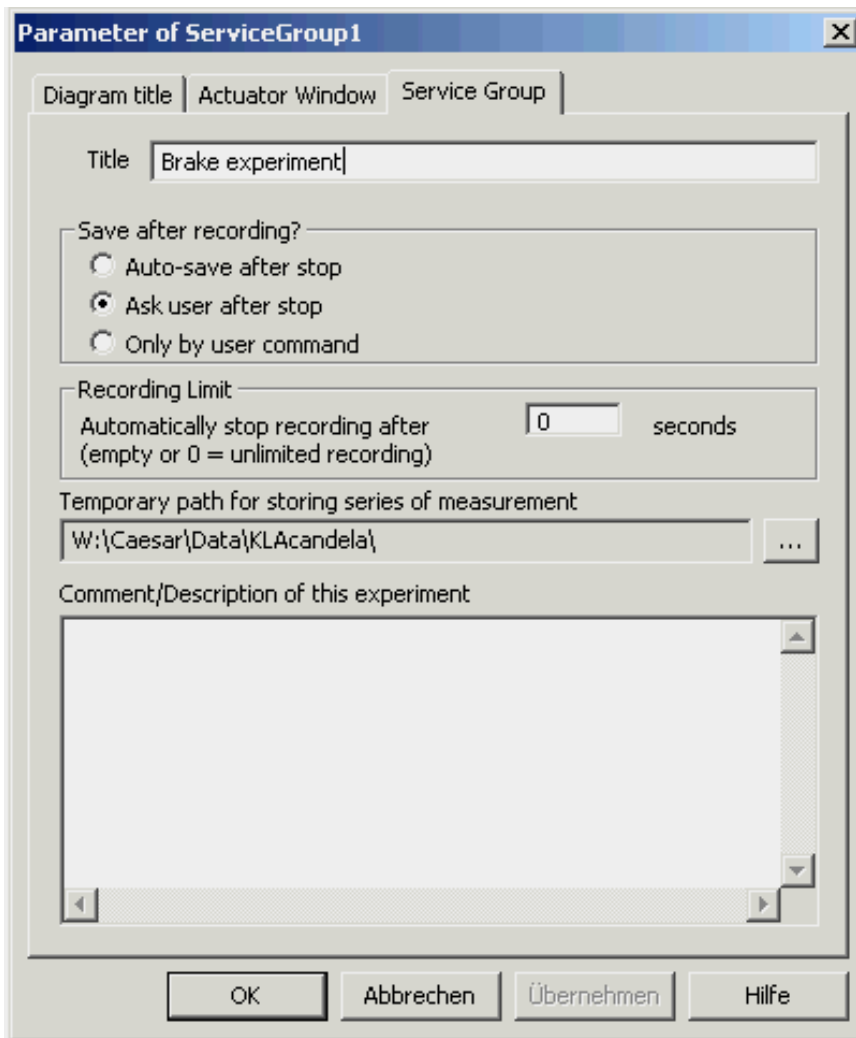
Important:

A service window may contain measurements of several different ECUs. If you use this window with a system that does not include all of the ECUs, the invalid measurements cannot be executed, so their values are not displayed.

The diagrams and services added are pre-configured and can be normally used with no or just little need to configure them. To change the settings, simply right click on the service or on the diagram to get the configuration dialog for that service resp. diagram.

## **Parameters for the Service Window**

By clicking into the empty space of the window, you get the configuration dialog for the window. Here you can change the settings affecting the function of the measurement window:



- Title of the group.  
This text is displayed in the title bar of the window
- Behaviour after stopping recording  
*auto-save* means, when pressing the stop button, all recorded values are immediately saved in a VSR/BIN file couple in the specified directory with a default file name.  
*Ask user* means, after stopping you will be prompted for a file name. You can just accept the default name (by pressing Enter), or enter another name.  
*Only by user command* means, you won't be prompted for saving after stopping the recording, but only when you start any action which deletes the recorded values (i.e. delete, start recording, close window)
- Recording limit. If greater than 0, recording will be automatically stopped after time is over.

- Path for storing the recorded values (VSR/BIN and CSV files). When you make many experiments (p.e. during test journeys), it might be useful to make separate directories for different days, subjects or cars.  
If not otherwise entered, the files will be stored in the directory of the currently opened system.
- Comment. Here you can enter any additional information in text form. It will be saved with the service group file VSG and with the measurement series files (VSR/BIN and CSV)
- Any other changes, like adding or removing diagrams is done directly by choosing a command from the popup menu or by closing diagram windows.



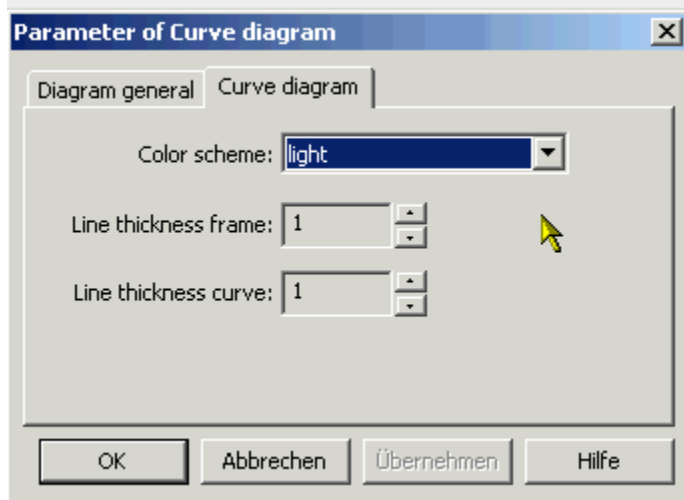
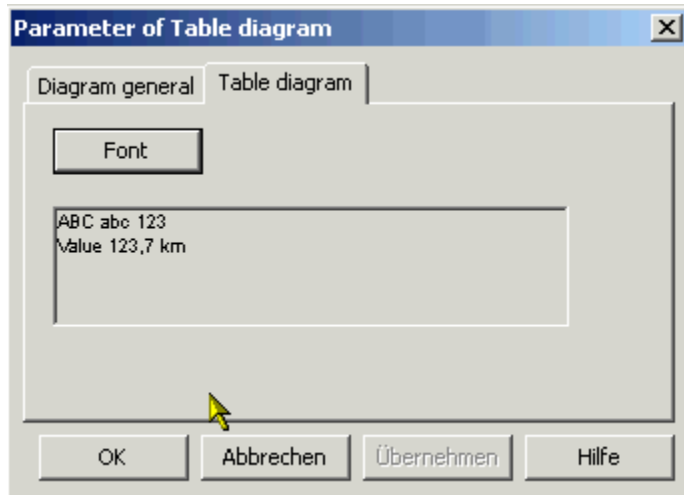
## Parameters for the Diagrams

By right clicking into the area of a diagram, but not on a service qualifier, you open a dialog window with parameters of the diagram. It has two property pages: one with common and one with specific parameters of this type of diagrams.

All these parameters are stored in the VSG file and are restored when the file is opened the next time.

### Common Parameters

All Diagrams have a title that can be edited. The default title describes the type of diagram.



### **Table Parameters**

In Tables you can change the font (font family, size, weight etc.).

### **Curve Parameters**

In this kind of diagrams you can set the line thickness and the color schemes. Please regard that in a dark scheme the lines should be lighter and in light schemes darker for a better contrast.

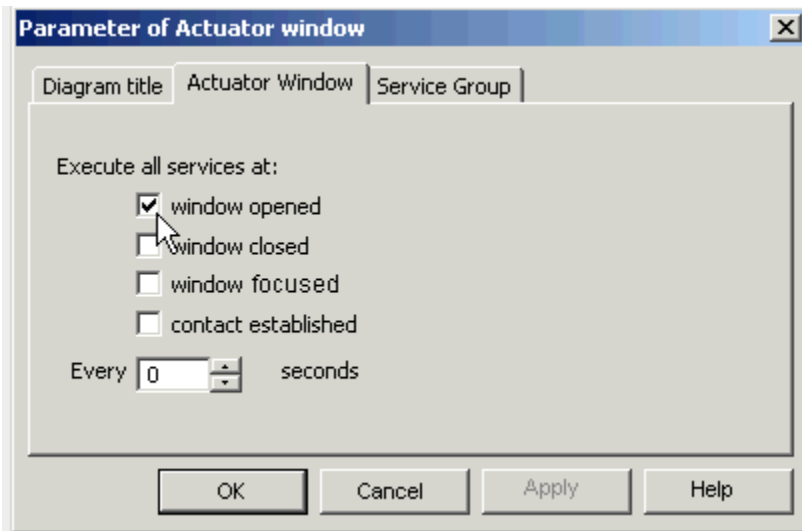
### **Bar Diagram Parameters**

Bar diagrams have no specific parameters.

### **Other Parameters**

Positions and sizes of diagram windows, table columns etc. can be changed by dragging with the mouse cursor.



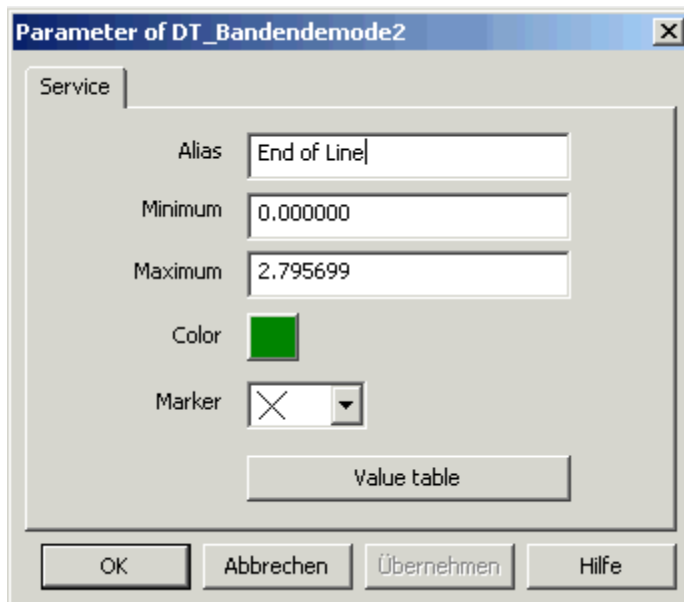


### Params of the Actuator Window

Here you can define events at which all services should be automatically executed. A popular use case is to execute preconditions at "window opened", or repeating setting actuators "Every x seconds" for some ECUs, which automatically reset all actuators after a timeout.

### Parameters of the Services

By right clicking on the qualifier of a measurement service in a diagram (in any diagram), you get the dialog window for setting measurement parameters.



First, you can give the service an alternative name (alias name), which will be displayed instead of the real qualifier or name in all diagrams of the current measurement window.

The Minimum und Maximum values determine the scaling in the diagrams (Bar and Curve diagrams). The next time a value is read from ECU, its value is checked. If it lies outof the limits, the limits are automatically adjusted to the new value. Those values apply to any diagram in which this measurement is contained.

Also the colour attached to the measurement is used in all diagrams: if the value is displayed as a green curve, the bar diagram shows it as a green bar and in the table there is an icon in the same green colour beside the name.

The marker is a symbol to mark measurement points in curve diagrams.

The value table is the means to attach numeric values to texts for services delivering text responses instead of numbers. Often used texts, as "on", "off", "yes", "no" and others are automatically represented as 0 resp. 1. Any other text received from the ECU gets the next free number.

If you want to change the automatically defined text-number pairs, press the *Value table* button. You get a table with all values used up to now with this measurement service. Here you can change the numeric values attached to the texts. After changing those values, all measurement points with this text are automatically changed to the new numeric value. So this change can be done also after recording or even after loading an already recorded series of measurements.

Parameter of DT\_Ac\_Data\_2\_Ac\_Stell\_2Byte

Service Rec Trigger Diagram title Curve diagram S

Automatically start recording when

value less than

value higher than

On the 2nd page of the service's options you can set a trigger event. If set, recording will automatically start when the value goes below the lower limit, or goes over the upper limit.

With this feature, it is possible to let the program read current states waiting for some value to be achieved and then recording starts automatically.

You can combine this with automatically stopping recording after some time (see options of the service window).

## Reading, Recording and Analysing Measurements

Regardless how many and which diagrams you have, the measurement window manages one list of measurement services used in at least one diagram. When reading (single shot or cyclic), every service is executed only once and its result is sent to all affected diagrams. It is also assured that all incoming values lay inside of the scale. If any value lies above max or below min, the limits are automatically adjusted.

In the state *Configuration (Free Running)* and *Recording* you can execute all services (request the values from the ECU or ECUs). You have the choice between a single shot and cyclic execution with or without delay:

- Single shot  
All services in the window are executed once
- Cyclic reading without delay  
All services are executed as fast as possible. In that case the order of incoming values may change, depending on several issues. Especially, if you have services of different ECUs, the ECUs execute them independently. This can mean, that you receive measurement values from ECU 1 more often than from ECU 2.
- Cyclic reading with delay  
When you record values over a longer period of time, it might be useful to read the values with a specified frequency. You can set the delay in seconds. In case that a full cycle lasts longer than the specified delay, the repeat time is adjusted. No new cycle is started before the previous has finished.

### Memory Usage:

The recorded measurement values are stored in system memory for a fast access and edition. Every value needs some dozens of bytes (depending on the length of a string result). The time you can let a cycle run before your machine gets slow (it happens when the operating system starts swapping memory contents into the swap file), depends on the amount of free RAM and on the speed of measurement reading.

Test your memory amount and the speed of memory filling with Task Manager before starting important measurement series.

If your system resources are insufficient, you can increase memory reserve (by stopping running applications and processes), reduce memory usage (by recording with a longer delay) or by regularly saving recorded data and starting new recording.

## Recording Measurement Values

During recording the program ensures consistency of the whole series. This means, in this mode it is not possible to change the list of used services - you cannot add new measurements or delete any already in use.

But you still can change the following:

- Add, close, move or resize diagrams
- Change colours, scale and value table entries
- Add, remove and move services which are already in the window

This fact has a feature which can be used for speed optimization of recording:

Performance:

Drawing diagrams highly increases the CPU load. The more drawing has to be done, the less measurements can be executed. If you want to read and record values as fast as possible, use just one table diagram during recording. To analyse the data, you can open other diagrams after recording - all recorded values will be displayed in them.

## **Saving and Editing Measurement Series**

The recorded data can be saved (as already mentioned) in a couple of files. The VSR file defines the window content and layout, including all parameters of measurements and diagrams mentioned above. The BIN file contains the binary values themselves. It can be used only in conjunction with the VSR file.

These two files must be used together and must not be edited outside of Ecoute, otherwise they may lose their consistency.

### **File Name**

When saving a measurement series, Ecoute creates a file name for it consisting of the name of the measurement window, the current date and time (the moment of beginning the recording). If you save more than one series in the same minute, the file name will be expanded by an underline and a digit. This way it is ensured that recordings won't be accidentally overwritten.

Of course, the user may change the default name in the file selection dialog.

### **Path for Saving Measurement Series**

If not stated otherwise, the files are saved in the default directory of the current system.

You may choose another path either by changing the directory in the file selection dialog (when prompted to save), or by setting the path in the window parameter setting dialog.

The newly set path applies to all following savings in this measurement window, either as VSR/BIN files, or as CSV text export files.

### **Automatic saving**

On testing journeys it might be useful to activate automatic saving (in the window parameter setting dialog). In this mode the user has only to open the appropriate window and establish contact to ECU before starting and can then start and stop recording by just pressing the Enter key. When recording is stopped, the recordings are automatically saved under the default file name.

### **Editing the Measurement Series**

After recording, the following changes can be applied to the data:

- Cut a part of the series. Select the cutting point with the time slider or the time selectionline in a curve diagram. Shift-Del deletes all points before the selected point, Ctrl-Del deletes all points after it.
- Change display options. You can add, delete, move or resize any diagrams, put measurements into them or remove them, change the colours, scaling and markers of the measurements.
- You can change also parameters of measurements. These are p.e. the lower and upper limits. If a measurement service responses with a text instead of a numeric value, the program attaches a numeric value to each text (Value table) to make it possible to display the values graphically. You can always change the numeric values attached to the texts - the program recalculates the recorded values in the whole series.

These changes can be made either after recording or even to an already saved series after loading it.

## **Diagram Handling**

### **Table Diagram**

In the table view the measurement values for the current (or any specified) moment are displayed. During reading/recording, the table contains the last read values. When you move the time slider or enter a time value in the time edit field, the last values read before the specified moment will be displayed.

Next to the name (qualifier or alias name) you see the colour assigned to the service. This color is used to display this measurement in any diagrams of the window.

The value of the measurement is displayed in the table as it comes: texts are displayed unchanged as texts.

For more clarity the measurements can be grouped and separated with a comment.

The font and the sizes of the columns can also be changed. When the names are very long or not very informative, instead of adjusting the width of the name column, you can assign alias names to the services. These alias names will be then displayed in all diagrams.

When the mouse cursor stops on the name or alias name, you can see a tooltip with the full name, qualifier and the ECU qualifier.

### **Bar Diagram**

The bar diagram shows the change of the measurement values as a horizontal coloured bar. At a glance you can see how the values change. The whole width of the diagram window is used for all values between the lower and upper limit of the measurement.

The diagram displays just the current (last read) or selected (by the time slider) values.

Clicking on the name of a measurement selects it. Above the bars a scale for the selected measurement is shown.

There are no more settings in this kind of diagram.

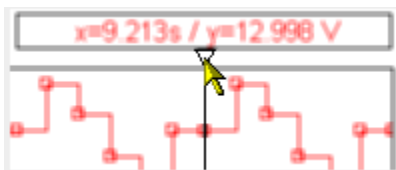
## Curve diagram

The curve diagram shows the chronological development of values.



The area on the left contains a list of measurements to be shown in the diagram. As in the other diagrams, here you can control the settings of the measurements (right click -> popup menu -> options -> parameter dialog). With the left mouse button you can select a measurement. The actions of the window's controls affecting measurements apply always to selected measurements. If none are selected, they apply to all measurements.

By clicking on the check mark next to the name, you can switch displaying this particular curve on and off. The controls' actions are ignored for measurements which are switched off, regardless of their selection state.




The right area contains the curves and the focus line. The focus line can be moved by dragging the triangle with the mouse.

If you drag the line with the left mouse button, it moves over the values of the curves. When dropped, it jumps to the nearest value point. All other diagrams also change to the selected display point.

If you drag it with the right mouse button, the line moves the curves also. The selected display point is not changed.















The x axis of the curve diagram can be "detached" from the window's time by clicking



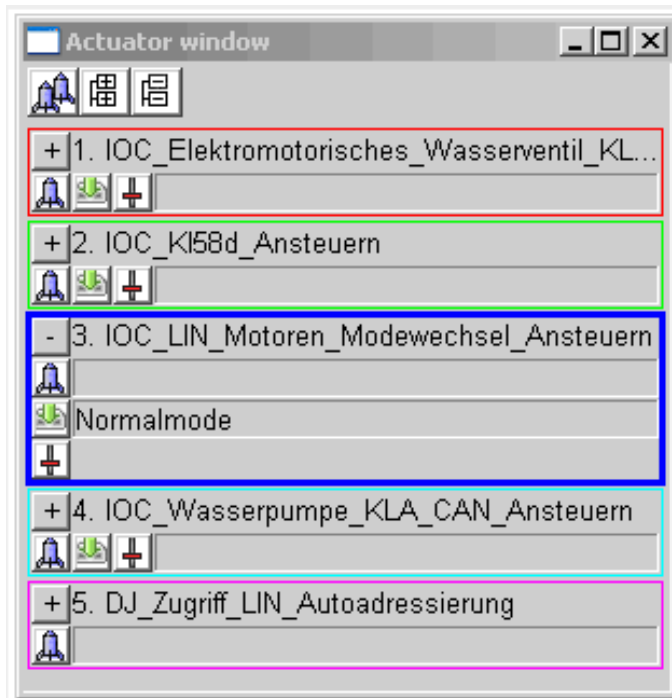
the  button. In this state, changing the selected display point in the curve diagram does not affect the other diagrams.

This can be used p.e. to analyze earlier parts of the curve during still running recording.






During measurement reading, new values always appear at the focus line, if the diagram is not in the "detached" state. All the curves are moved to the left as time passes. They are moved even if no new values are received, until you stop reading or recording by pressing the *stop* button.

<b>Controls in the Curve Diagram</b>	
	Press this button to move the curve to the previous point of selected measurements
	Press this button to unselect all measurements
	Press this button to select all measurements. This does not affect the display state: unchecked measurements stay not displayed.
	Press this button to move the curve to the next point of selected measurements
	Press this button to zoom into the time (x) axis
	Press this button to zoom out of the time (x) axis.
	Press this button to zoom into the y axis of selected measurements
	Press this button to zoom out of the y axis of the selected measurements
	Press this button to select a section of the x axis which shall be expanded to the full length
	Press this button to select a section of the y axis which shall be expanded to the full height (affecting selected measurements)
	Press this button to select an area of the diagram to be expanded to the full diagram size (x and y directions). This also affects the selected measurements in y direction.
	Press this button to show the full curve on the x axis. If the reading is in progress, all values from 0 to the current point are expanded to the full axis length.
	Press this button to maximize the curves of selected measurements to the total height of the diagram.
	Press this button to detach the x axis from the time of the whole window. When detached, changing the position of the focus line does not affect the other diagrams. Also the position of the time slider does not affect the detached curve diagram.




## Actuator Window



A color frame contains all controls needed for using the service.

-  expand frame
-  collapse frame
-  execute service. The param values entered previously (or loaded from file) will be used.
-  open window for entering input param values. After setting params no execution is performed.
-  open slider control for convenient settings. Changing the value changes the param value and executes the service. The last used param value is stored for later usage.

There are also three more buttons:

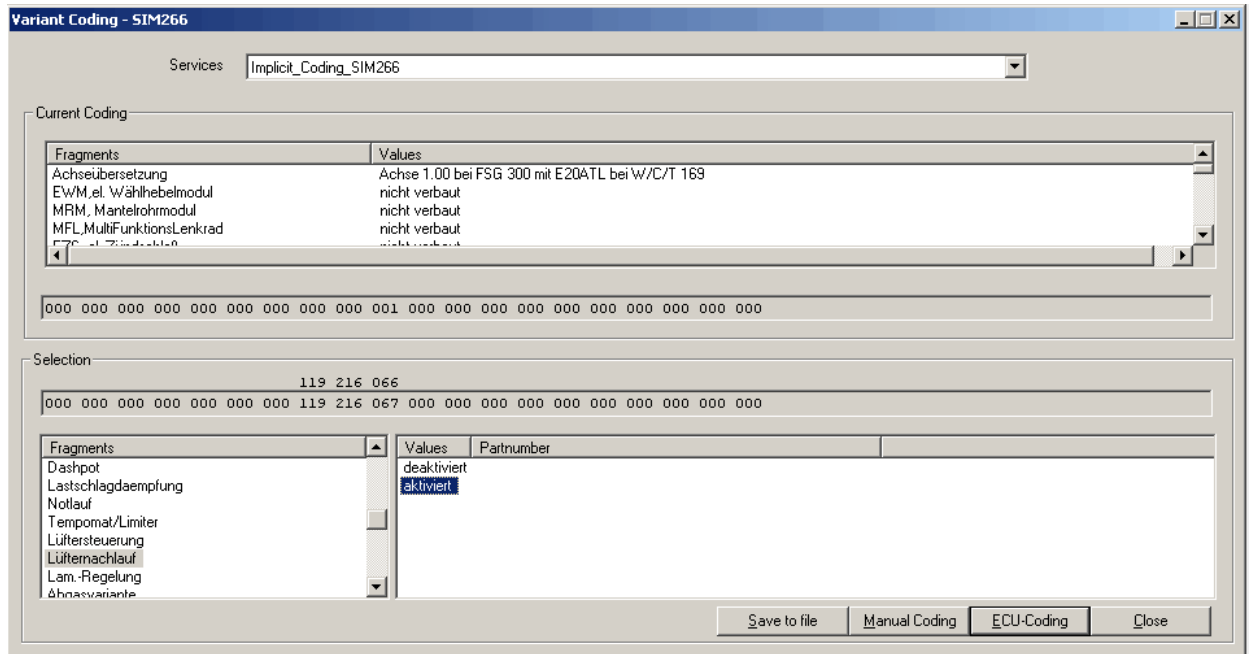
-  execute all services
-  expand all frames
-  collapse all frames



## 2.4.4.10 Variant Coding

The current state of the variant coding can be read and modified with Ecoute. Proceed as follows:

- Select *Coding / Variant coding* in the Ecoute main menu.
- The variant coding dialog is opened.



If preconditions for variant coding are specified in the system description for the current ECU variant, these are executed when the window is opened. If services with input parameters are specified as preconditions, an appropriate dialog is displayed in which the user can enter the parameters.

### Services

All the ECUs coding services are displayed in this dialog field. When a service is selected from the selection list, all fragments and the respective values in the "selection window" are displayed in the lower section of the dialog.

### Current Coding

The current coding string from the ECU is read and displayed when the dialog is opened. Fragments and values (max. 4 lines) matching the coding are displayed in a list field. If the coding is not found in the parameterization, "unknown" is entered under values. The coding string can be displayed either in decimal or hexadecimal format depending on the setting in `Vediamo.ini`. The coding string format is specified by the

"*VarCodStringFormat*" entry in the [ECOUTE] section of *Vediamo.ini*. This entry can have the value "Decimal" or "Hexadecimal". The value "Decimal" means that the coding string is displayed in decimal form. The value "Hexadecimal" means the coding string is displayed in hexadecimal form. The coding string is in decimal format by default.

## **Selection**

All parameterized fragments of the selected service are displayed along with their values in this section of the dialog. The field *Coding string* contains no values when the dialog is opened. As soon as a value is selected from the list, the corresponding coding string is displayed immediately. In addition, the difference between the current coding string and the selected coding string is evaluated and the position where the two strings differ is marked with the numerical value of the difference (decimal: absolute value of the difference of the two values; hexadecimal: bitmask of changed bits).

Externally parameterized coding fragments: Coding fragments which are parameterized by way of external files (ccf suffix), are denoted in the dialog by the suffix (ex).

## **Save to File**

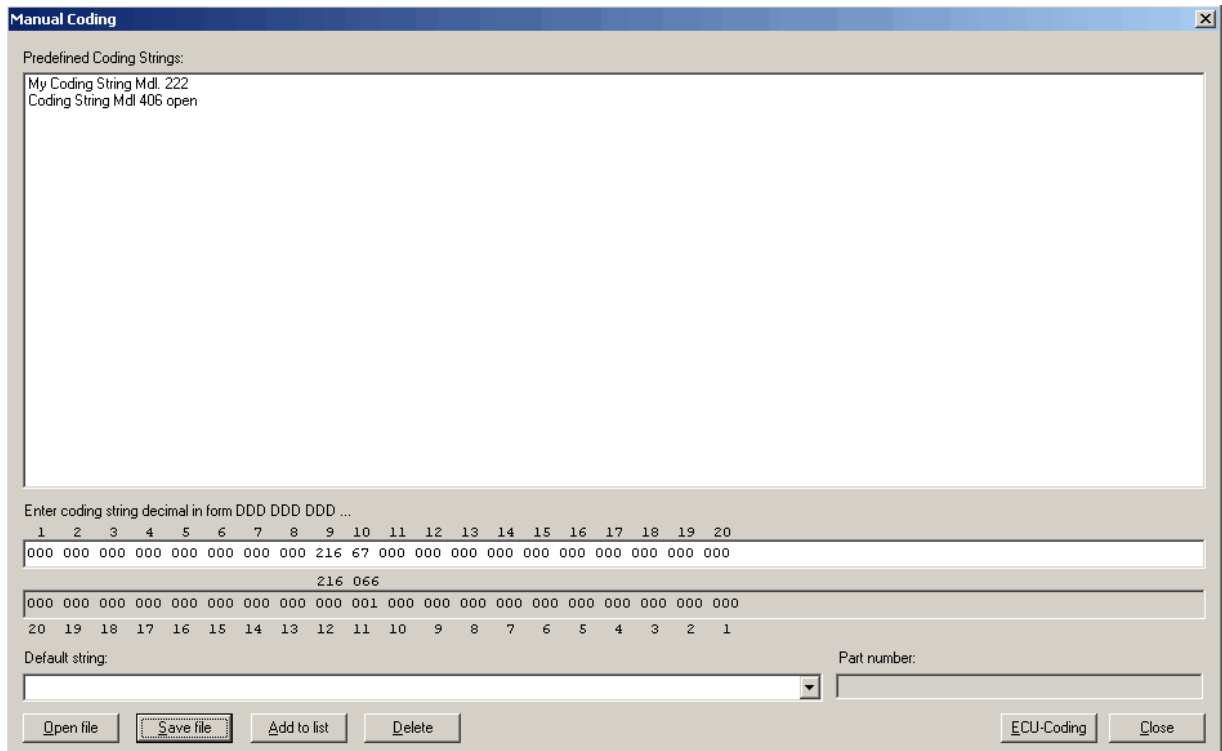
The variant coding can be stored in a snapshot file by clicking this button.

## **Close**

Clicking this button closes the variant coding dialog. After the button is pressed, any postconditions regarding variant coding included in the system description are executed, in the same way as the preconditions described above. If the option *shut down cycle for coding* is activated in the VSB and coding was performed in Ecoute, the shut down cycle dialog is then opened. The dialog prompts the user to turn the ignition off and then on again.

## **Manual Coding**

Clicking on the button *Manual coding* opens a dialog, in which the coding string can be edited manually.



The coding string in the edit field is taken from the selection field. If the selection field does not contain a string, the coding string is taken from the field of the current coding string. The user can edit the coding string and execute the variant coding by clicking on the *ECU-Coding* button. The dialog is closed without coding by clicking on the button *Close*.

You have the possibility of naming your own coding strings and storing them in a list (*Add to list* button). Strings in the list can also be deleted (*Delete* button). The compiled list can be stored in a file (*Save file*), or previously compiled lists can be loaded from a file (*Open file*). This is particularly useful when the number of combinations is very high and working with your own selection lets you work more effectively.

The coding string loaded from the file is displayed in a decimal or hexadecimal format, according to the parameter `VarCodStringFormat` in the file `Vediamo.ini`. The format in which the string is stored in the file might be decimal, hexadecimal or even mixed (p.e. "123 001 5D 7F 111") and it has no influence on the display and entering format.

#### 2.4.4.11 Flashing

Ecoute allows you to read and modify the current state of the ECU flashware. Proceed as follows:

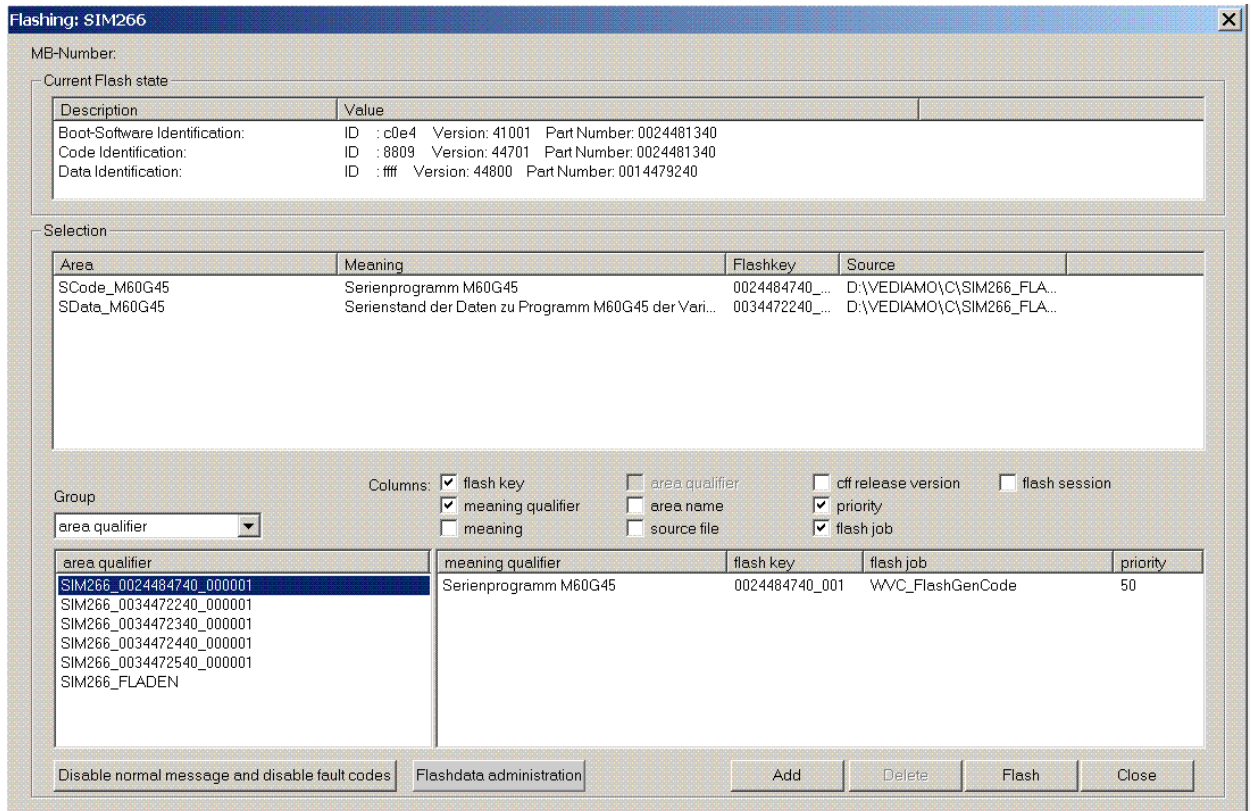
- Select *Coding / Flash* from the Ecoute main menu.

- The flash dialog then appears.

If you have no flash data for the contacted ECU, no window opens at this point. The message appears in the status line, that no flash data is available for this ECU.

**Important:**

Note that the DIOGENES name in the ECU data (CBFs) must be the same as that entered in the flash data (CFFs). This prevents the wrong data from being flashed on the ECU.



Ecoute allows you to flash multiple areas in a single flash process. The number of areas to be flashed at the same time in a flash process is specified in the system description under ECU properties. The default setting is one area per flash process.

**Current Flash Status:**

This section of the window contains a list field, in which information on the flash data can be displayed. The information is specified in the system configuration as follows: For every ECU variant, any services which provide the user with additional information on the ECU can be listed and assigned in the properties dialog (system configuration, function *ID-Block Information*). To display this information, Ecoute queries the ECU for the ID of the services entered in the system configuration and then executes them. The

service ID (e.g., "Software Status", "Code", "Boot Block", etc.) is displayed in the left column *Description* and the result from the execution of the respective diagnostic service in the right *Value* column.

### **Selection:**

This section of the window has two list fields. The first list field displays the flashware assembled by the user. The second list field displays the lists of the available flash areas, flash meaning and flash keys. The user can choose the grouping criterium for the displayed data by selecting it in the *Group* combo box. The selected group is displayed in the lower left list box.

The check boxes described as *Columns* can be used to select which information could be displayed for the selected group. The positions of the columns can be changed by drag & drop. They are saved automatically when the dialog window is closed, and restored when it's opened the next time. Clicking on the column title changes the sort order of the list items.

If "1" is entered in the system description for *maximum number of areas to flash simultaneously*, than only one flashware can be flashed in Ecoute during one flash process. In this case, the user has the possibility of selecting one meaning or a FlashKey for the currently selected area and flashing the ECU by clicking the button *Flash*.

If the *maximum number of areas to flash simultaneously* becomes greater than "1", then *several* meanings to be flashed can first be added to the first list field using the button *Add*. When the button *Flash* is clicked, all the meanings in this list will be set in the sorted order. Afterwards, the set meanings will be flashed in the correct order. The status of the flash process will be shown in the progress display. After the flash process has ended, the user is prompted to turn the ignition off and back on again after the shut down cycle.

If a meaning parameterized as "late binding" is selected, double clicking or using the <RETURN> key will cause an *Open File* dialog to appear. The user can select the appropriate file from this dialog. The name of the selected file is entered in the list field under the column *Meaning*. The flash process can then be started with the button *Flash*, just like in the case with early binding.

The segment information for every flash meaning can be displayed. Select a flash meaning in the right list box and press the right mouse key. A dialog opens which displays the segments of the currently selected meaning. A segment is understood to be the starting address of the area and the length of the flash region. Note that the segment information of an area can change depending on the set meaning. Depending on the diagnostic data, an area/meaning combination can have several segments.

Button “disable normal message and disable fault codes”:

Since employment of CAESAR 3.0, no automatic Disabling of normal message and fault codes of the remaining bus participants takes place while flashing an ECU in the vehicle (vehicle is contacted via the OBD socket / tester is connected to the gateway controller, e.g. to the Powertrain CAN).

This mechanism was provided there so far by CAESAR 2.x (only keyword protocols were supported).

Starting from CAESAR 3.0 , due to the fact that several protocols may be used in one vehicle, the mechanism "disabling normal message and fault codes" must be provided by the application.

The Disabling of the remaining bus participants is made with a functional message by a fixed Identifier. Over this Identifier then cyclically "tester present" must be sent, in order to keep Disabling upright. To stop Disabling, another message must be sent, or sending "tester present" must be terminated. The same applies to the suppression of error registrations (however with other messages and on other identifiers).

"Disabling normal message and fault codes" is protocol related, i.e. for each protocol the mechanism must be activated separately.

The ECOUTE Client supplies such a mechanism over the button “disable normal message and disable fault codes”.

By application of this mechanism, a higher flash performance is to be expected .For flashing at the vehicle, a mechanism is implemented, which switches the remaining bus participants during the flash procedure mutely and suppresses error registrations. The mechanism currently supports both the Keyword and UDS protocol. The activation/deactivation of the mechanism must be made by the user manually by pressing the related button at the suitable time. The mechanism is activated / deactivated over API 1. The description of the services is specified in a file "SpecialFunctions.ini", which is located in the Vediamo program folder. The file can be edited by the user.

Button "Flashdata administration":

When pressing the button "Flashdata administration", a window providing the following functions will be displayed:

- The Tree-Control on the right side shows the currently in caesar registered .cff files.
- The function "Add" opens a file-selection dialog in which other .cff files or a directory can be selected. Newly selected files / files in the selected directory are immediately registered at caesar. Probably occuring errors are displayed in the status window.
- The "Remove" function deregisters the currently selected file at caesar.

- The function "Load standard list" unloads all currently loaded flash files and loads instead the flash files listed in the file \\ Flashfiles.cfg If the file does not exist, no flash files are unloaded and an error message is displayed instead. If no system is loaded, this button is disabled (grayed).
- The function "Save standard list" stores the names of all currently loaded flash files in the file \\ Flashfiles.cfg If no system is loaded, this button is disabled (grayed).
- The function "Load list" prompts for a .cfg file name, unloads all currently loaded flash files and loads instead the flash files listed in the specified .cfg file.
- The function "save list" prompts for a .cfg file name and then stores the names of all currently loaded flash files in the specified file.
- "Close" closes the Flashdata administration Dialog.
- Under every file entry in the tree the related flash file contents (Areas, Meanings, Flashkeys) are displayed. When the user changed any flashdata in the Flashdata administration Dialog, the superior flash dialog will be updated subsequently: The list of selected meanings will be deleted, the list of available flashware is rebuilt.

See also "Flash an ECU"...

Warning: Flashing overwrites the ECU software.  
Erroneous flashware can possibly destroy the ECU.

#### **2.4.4.12 OBD2**

The OBD2 function is similar to a Scan-Tool resp. OBD2 or EOBD regulation and gives the Ecoute user direct access to the most important diagnosis data (DTC - diagnostic trouble codes, current data, OBD tests for the lambda sensor, ignition, catalysator etc) without specific ECU files (\*.cbf files).

To access these information, ten modes are defined:

- Mode 1 - Current diagnosis data (PIDs)
- Mode 2 - Freeze Frame (PIDs)
- Mode 3 - exhaust relevant stored DTCs
- Mode 4 - clear exhaust relevant stored DTCs
- Mode 5 - Test results of oxygen sensors
- Mode 6 - Test results of diagnostic functions (MIDs)
- Mode 7 - exhaust relevant DTCs of the current or last driving cycle
- Mode 8 - Controlling the On-Board System
- Mode 9 - Vehicle information (Info IDs)
- Mode 10 - exhaust relevant permanent DTCs

The standard of OBD2 function is defined in the norm ISO-15765-4 (Road vehicles - Diagnostics on Controller Area Network (CAN) - Part 4: Requirements for emissions-related systems)

## Functionality

The OBD2 function can be opened and closed by the menu item *System / open OBD2* or *close OBD2*, or by clicking the OBD2 buttons in the Ecoute tool bar.

## The OBD2 window

After opening the OBD2 function two separate windows are opened: the OBD2 System window and the Display window.

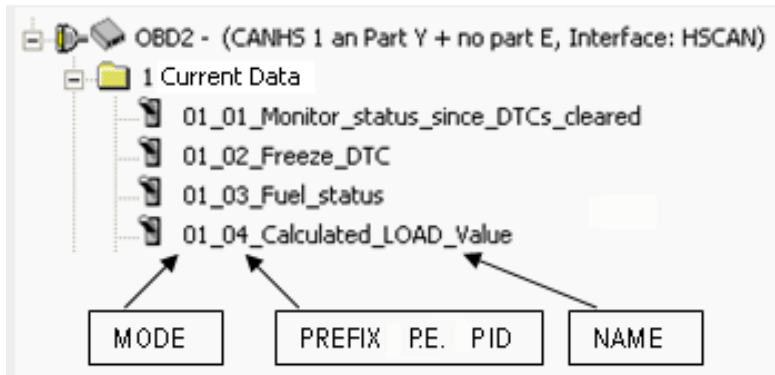
The screenshot displays the Ecoute software interface with three main windows:

- System Window:** A tree view on the left showing OBD2 modes (1-10) and their associated services (PID, TID, MID, and info ID). The 'Vehicle Information' branch is expanded, showing various data points like '09\_01\_MessageCount\_VIN' and '09\_02\_Vehicle\_Identifier'.
- Display Window:** A table titled 'Test results' showing data for Mode 6. The table has columns for 'ECU/Qualifier/Presentation', 'Test', 'Min', 'Max', 'Unit', and 'Metric'. It lists various sensors and monitors such as '06\_01\_Exhaust\_Gas\_Sensor\_Mo...', '06\_21\_Catalyst\_Monitor\_Bank1', and '06\_42\_Exhaust\_Gas\_Sensor\_He...'. The '06\_01' row is highlighted in red. A 'Toolbar' is visible above the table, and a 'Read all modes' button is in the top right.
- Status Window:** A log window at the bottom showing system messages: '10:31:32 Server initializing...', '10:31:36 OK', '10:31:49 Available Caesar-Hardware:', and '10:31:49 610275;Part Y + no Part E(OBD)'. The status bar at the bottom indicates 'Ready' and 'Protocol is running'.

## OBD2 System window

The OBD2 System window has the same structure as the Ecoute System window. It contains tree branches for all the ten OBD2 modes. Every branch contains all defined services (PID, TID, MID and info ID):



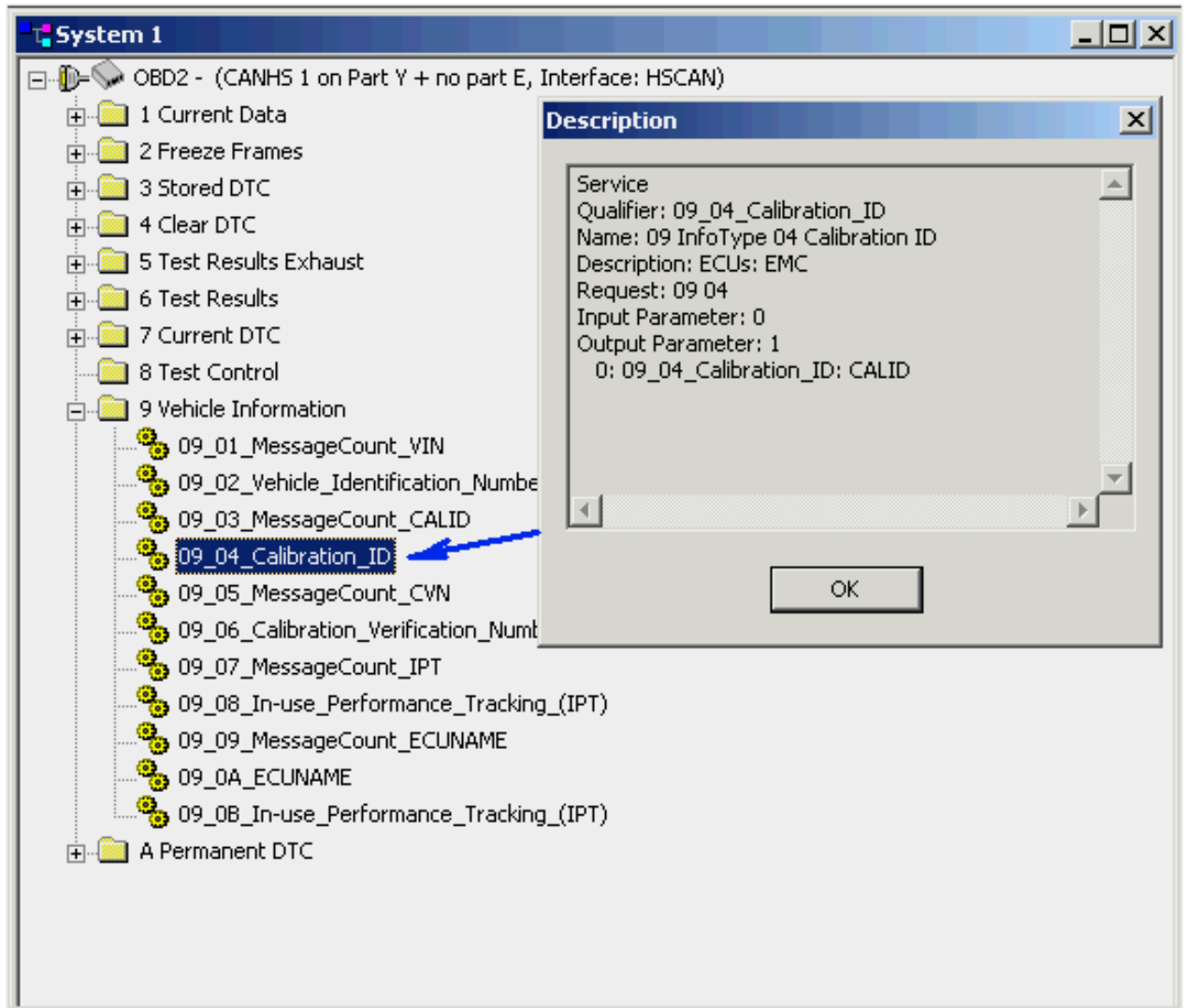


### Functionality:

The OBD2 System window displays all defined services regardless if they are supported by the current ECUs. By double click (or select and press ENTER) the service is executed and the result(s) displayed in the status window.

Executing a service not supported results in the information that the execution is finished, without any result.

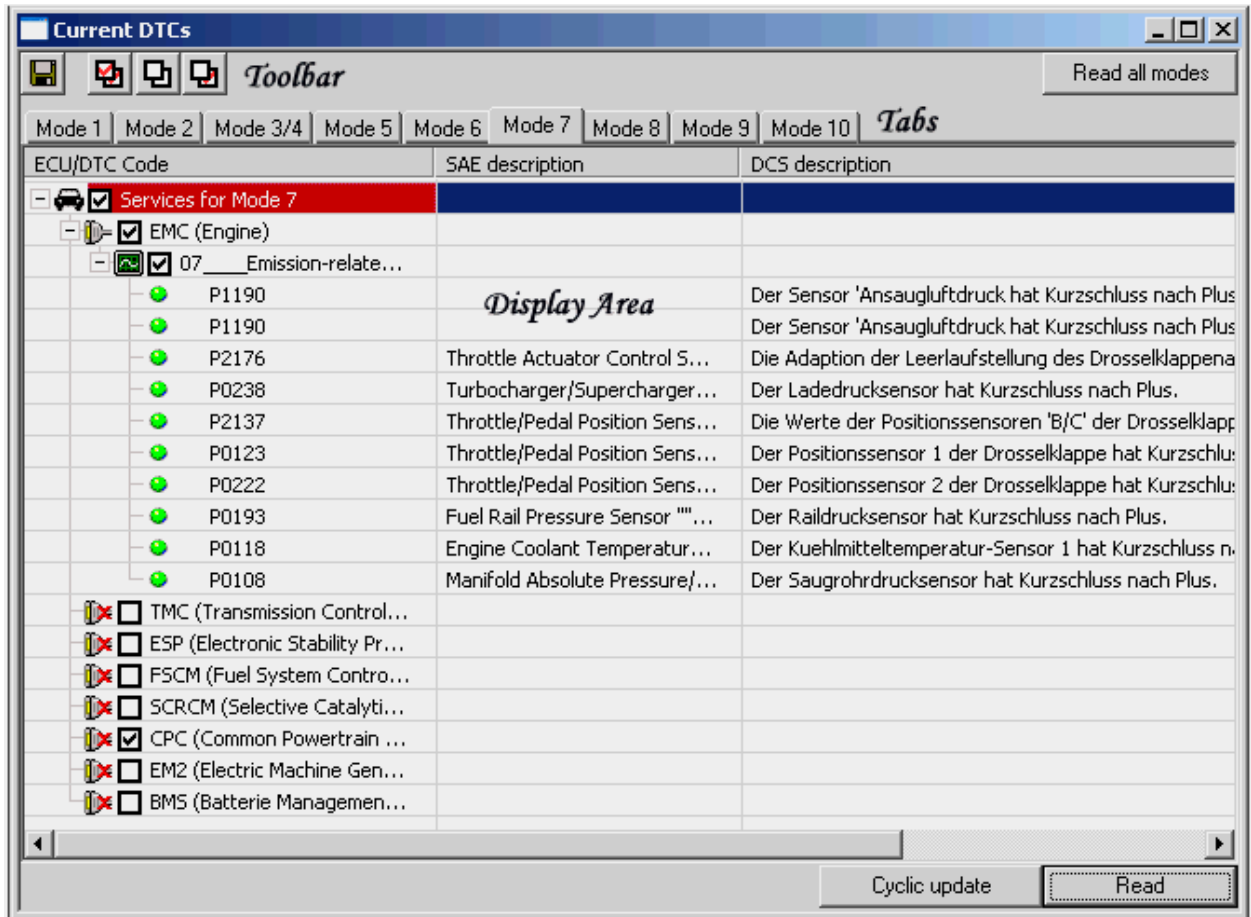
From the context menu of the mode branch, the function *Execute All* calls the execution of all supported services of this mode for every ECU separately. In this case every result is displayed in a separate line, followed by the metric (shortcut for the meaning). From the context menu of every service, the properties of the service can be opened.







## OBD2 Display Window

The OBD2 display window consists of a big display area, a common and a special toolbar with buttons and a tab bar.

The tabs are used to select which mode to be executed and displayed. Changing the mode causes also activating or deactivating special buttons in the lower special tool bar.



### Functionality of the common tool bar buttons

	Save log. All values read, contained in all mode pages, will be saved in a txt or html log file.
	select all elements of the tree on every mode page
	deselect all elements of the tree.
	Restore selection state as it was when window was opened.
Read all Modes	Execute all selected services of all mode pages.

The selection state of all services and ECUs, as well as the width of the table rows (can be changed by drag&drop), are stored in the file *OBDWin.ini* when closing the window. During opening it the next time, the states will be restored again.

The special tool bar contains up to three buttons, depending on the chosen mode page:

Read	execute all selected services on the current mode page
Cyclic update	cyclically repeat execution of selected services of the current mode page
Clear errors	delete stored DTC by using mode 4

Differences between the modes in the display window:

The OBD modes are selected by clicking on the tabs. Every mode has a special table structure. The left column contains a tree view with all ECUs and their services:

- Mode 1
  - level 1: ECUs
  - level 2: Service - all supported services (Mode\_PID\_Name)
  - level 3: Results - one service can deliver more than one result value
- Mode 2
  - level 1: ECUs
  - level 2: Fehlercodelevel - Anzeige aller gespeicherte Fehler
  - level 3: Freeze Frame Services - all supported services (Mode\_PID\_Name)
  - level 4: Results - one service can deliver more than one result value
- Mode 3 /4
  - level 1: ECUs
  - level 2: Read stored DTC service
  - level 3: single DTCs - Code, SAE and DCS/DCA description
- Mode 6
  - level 1: ECUs
  - level 2: Service - all supported services (Mode\_MID\_Name)
  - level 3: Results - one service can deliver more than one result value
- Mode 7
  - level 1: ECUs
  - level 2: Read current DTC service
  - level 3: single DTCs - Code, SAE and DCS/DCA description
- Mode 9
  - level 1: ECUs
  - level 2: Service - all supported services (Mode\_InfoType\_Name)
  - level 3: Results - one service can deliver more than one result value
- Mode 10
  - level 1: ECUs
  - level 2: Read permanent DTC service
  - level 3: single DTCs - Code, SAE and DCS/DCA description

Every ECU and service has a checkbox. By marking it, the user decides which services to execute. Unmarking an ECU, all services of this ECU are not executed.

## Output files / OBD Log

The results can be saved in a text or HTML file. By clicking the "Save log" button, the user is prompted for additional information to be saved: name, company, vcehicle information, and the name and format of the log file.

To make the file more readable, the name/description of the values are put to the end of the lines.

#### Timing Problems:

With fast PCs (double and quad core) and the eCOM diagnosis hardware some timing problems might occur. If you cannot access the ECU by OBD2 communication, although normal communication works, the problem can be solved by increasing the values for the parameters `OBD_P2_MAX` (above 250) and `OBD_REQREPCOUNT` (above 1). See also [INI Parameters](#) for [SERVER]

#### **Important**

Higher values for the above parameters slow down the OBD communication during the initialization.

### 2.4.4.13 Configure Ecoute and Server Options

#### **Ecoute Interface**

You can manipulate the Ecoute windows as needed with the mouse. The positions and sizes are automatically stored in the [configuration file](#) when the program is ended, and are restored the next time the program starts. The service group, measurement group and actuator group windows are stored in the VSG, MWG and STG files.

#### **Ecoute Session**

The following Ecoute properties are stored in a session file (VSC) to be restored the next time the program starts:

- Selected system, ECU contact status and channel assignment
- Size, position and partitioning of the status, selection and main window
- All open windows (error, measurement, actuator and trace) with their positions and sizes
- Presentation options

#### **Further Options**

A window for setting numerous options can be reached from the main menu under *Extras / Options*. The set options are stored in [Vediamo.ini](#). They can be modified there using the [INI editor](#) or a regular text editor.

The options are divided into three groups and each group is edited in its own window. The tabs at the upper window edge (CTRL-TAB) make it possible to switch between the different windows.

## General Options

### Switching Between Name and Qualifier Display

DIOGENES services are entered in DIOGENES using unique IDs called qualifiers. Qualifiers uniquely identify a service. Vediamo Java routines always use qualifiers. In addition, each diagnostic service is given a name. This name is not unique, but allows the user to give the service a meaningful ID.

In Ecoute, it is possible to switch between displaying qualifiers and names using the options dialog. Simply select the desired setting. The display of names or qualifiers applies to all windows (selection window, measurement group, actuator group,... ) of the Ecoute application.

### Snapshot File

Ecoute allows the storage of snapshot files. The path and filename for the snapshot file can be set in the options dialog (*Extras / Options*). Select the "..." button directly next to the "Snapshot File" entry field in the options dialog. This opens a standard file selection dialog. Navigate to the desired storage directory and enter the filename of your choice.

### Hardware Options

Hardwareoptions are edited in the startcenter application.

### Start Options

#### Load Last Session

If the field *Load last session* is activated, the last selected session file is automatically loaded the next time Ecoute is started.

#### Establish Contact Automatically

Ecoute establishes contact automatically with ECUs in the following three different cases:

1. After selecting a system (initial contact)
2. When contact is lost due to external effects
3. When the user's attempt to establish contact fails

All cases can be configured independently using the options dialog. If automatic contact is active, the user can see the appropriate message in the status window.

In cases 2 and 3, contact is attempted until it is either established, or the system is locked or switched, or the option is reset.

## **Establish Contact Automatically After System Selection**

If the option *After system selection* is set, contact is attempted with all ECUs contained in the system immediately after the user has opened the system. If more ECUs are in the system than free CAESAR channels, an error message is displayed for each ECU which cannot be assigned to a CAESAR channel. If the first contact attempt for an Ecoute system selection fails, a further option (case 3) determines whether contact with the respective ECU should be attempted cyclically.

Important:

For manual command input (Ecoute), no automatic initial contact is ever performed, i.e., the option for automatic contact is not evaluated when a VND file is loaded.

## **Establish Contact Automatically After Losing Contact**

If contact with an ECU is lost due to external influences (physical connection is pulled, disturbances on the line, ECU doesn't reply, etc.), the attempt will be made to re-establish contact with the respective ECU if the option *After contact loss* is set.

Important:

For manual command input, contact will not be established automatically, independent of this setting.

## **Establish Contact Automatically After Failed Contact Attempt**

If the option *After failed contact attempt* is set, cyclical attempts to establish contact with an ECU are made after an unsuccessful attempt. This also applies to initial automatic contact after system selection.

Important:

This does not apply to contact attempts coming from Java routines. It also does not apply to contact attempts using [manual command input](#).

## **Automatically Execute Initialization Services**

The initialization services contained in a system description can be started after the system has been loaded or after communication has been established. The Ecoute client starts the services independent of the related options which can be set. A system description can contain multiple initialization routines. The system init routines are executed if the option *After system selection* is set, the ECU and ECU variant init services are executed if the option *After contact* is set.

An init service is started under the following circumstances:

1. System init routine (only if the option *After system selection* is active): Each time a new system is started or selected, the Ecoute client checks whether the option *After system selection* is set. If it is, the Ecoute client informs the diagnostic server, that the system init routine should be executed. The diagnostic server executes the Java routine, if one is included in the system description.
2. ECU init services and ECU variant init services (only if the option *After contact* is active):  
 After each time that contact has been established to one of the ECUs in the system, the Ecoute client checks whether the option *After contact* is set. If it is, the Ecoute client informs the diagnostic server, that the init services of the relevant ECU should be executed. The diagnostic server then executes the ECU-related init services of the ECU first and immediately afterwards, the variant-related init services of the identified variant or the basic variant, if these are included in the system description.

If multiple Ecoute clients are present, the diagnostic server ensures that init services are only executed once after a new system selection or contact with an ECU (prior to new contact or contact loss).

The relevant option settings are stored in the `vediamo.ini` file under the following key:

```
[Ecoute]
ExecuteInitSequence = <Key>
```

The following applies for <Key>:

- 0 No init services are executed
- 1 After system selection
- 2 After contact
- 3 After system selection and after contact

## Logging Options

Ecoute allows the communication with the ECUs and the diagnostic server to be logged in different ways, and the logs to be stored in files. The type of log can be specified in this window. In addition the display shows in which files the log data is stored.

## DCDI Channel (Ecoute Client)

This specifies in which format the ECU communication is displayed in the trace window that can be called up in Ecoute using the menu selection *Extras / Trace display*. The complete name of the protocol file is also shown. This name is always combined with the qualifier of the relevant ECU. The following trace formats are possible:

### Data Blocks

The request and response are both shown in hex format and logged in Ecoute when this option is activated.



### Bytes & Timing

The communication between ECU and tester is logged in detail if this option is activated. This function is called monitoring. Monitoring should only be used for debugging on the ECU communications protocol level.

A button allows a dialog to be called up to select the directory in which this ECU-related data is logged. During logging, these files are stored under the name <ECU>Trace.log in a subdirectory with the name of the system.

### **DCDI Channel (Diagnostic Server)**

All data for logging the ECU communication which is displayed in Ecoute must be transmitted by the diagnostic server to the Ecoute application. This transmission process requires a certain amount of time. For time-critical diagnostic procedures therefore, it makes sense to record the log data directly in the diagnostic server. The complete name of the logfile in the diagnostic server is shown in the window. The behavior of the server can be controlled with the following entries:

#### Diagservice & Data Blocks

The qualifiers for the executed diagnostic service, the request and response in hex format, and additional ECU-related CAESAR logging information is recorded by the diagnostic server.

#### Bytes & Timing

The communication between ECU and tester is logged in detail (monitored) when this option is activated, in the same format as already described further above.

It is not possible to select both options at the same time.

A button allows a dialog to be called up to select the directory in which this ECU-related data is logged. During logging, these files are stored under the name <ECU>Kanal.log in a subdirectory with the name of the system.

In the trace window in Ecoute, when tracing a CAN bus, the filter settings for Bytes & Timing are displayed:

- Filtering active / not active
- Active filters
- Name of currently used filter file

If the communication is based on a CAN protocol, the following applies for logging the DCDI channel communication:

Accumulated CAN monitoring data is filtered. By default, only the CAN messages with the CAN IDs of the respective ECU from the current CBF are logged. In place of the CAN ID, the ECU qualifier is shown in the log.

In the `Vediamo.ini` file, there is a key:

```
[CAESAR]
"MonitoringFilterCANIDs"= 0 | 1 .
```

If this key has the value 0 then, as before, no filtering of the monitoring data takes place. If the key has another value, or is missing completely, the monitoring data is filtered (= default setting).

Two comparators per ECU are used as criteria for the filtering:

`CP_REQUEST_CANIDENTIFIER` and `CP_RESPONSE_CANIDENTIFIER`. When a new CAN message is read from the monitoring buffer, a check is made whether the CAN ID corresponds to one of these comparators. If not, the message is filtered (if filtering is activated in `Vediamo.ini`).

The ECU qualifier is entered in the log in place of the CAN ID.

In addition, filter specifications can be made in a text file for any desired CAN IDs.

The text file format is given in the following example:

Example:

```
;Assignment CAN-IDs →ECU identifier
2016,ECU
2024,ECU
;
786,Unknown$312
528,Unknown$210
```

Lines that begin with a semicolon are interpreted as comment lines and are ignored.

For filter specifications, the CAN ID is listed in decimal form first and then, separated by a comma, the identifier to be displayed.

The content of the file can be changed during runtime. The filter information is updated after contact has been established with the ECU.

The name of the filter file to be implemented is specified by the `Vediamo.ini` entry

```
[CAESAR]
MonitoringFilterCANIDFile=
```

Example:

```
[CAESAR]
MonitoringFilterCANIDs=1
MonitoringFilterCANIDFile=C:\ProgramData\Vediamo\Config\CANIdFilter.txt
```

## Status Messages

If the option *Generate Status Logfile* is activated, all outputs in the Ecoute status window are also written into the file whose name appears in the option window.

## Setting Options using Command Line Parameters

When starting Ecoute, any keys from the `vediamo.ini` file can be entered in the command line (next to the name of the session or system description to be loaded). The entered keys are valid for the duration of the running diagnostic session.

The command line syntax looks like this:

```
Ecoute.exe /VI "[Section] Key Value ... [Section] Key Value ..."
```

The option `/VI` or `-VI` serves to differentiate between session/system description names and `vediamo.ini` keys. The list of keys to upload must be enclosed in quotes.

When Ecoute is started with the `/VI` option, the superseding `vediamo.ini` keys are transmitted to the server application prior to booting the server. There they are used temporarily in place of the corresponding values in the `vediamo.ini` file.

## Exceptions

This mechanism only affects the Ecoute instance which starts the server. If the server is already in operation, it ignores the keys specified in the Ecoute command line.

The key that defines the language setting (`[Common]Language`) on the server side cannot be overwritten, since this key goes into effect directly after the server application is started by the operating system (even before Ecoute can transmit any keys for writing over).

Specified keys are not only effective in the server, but in Ecoute as well, i.e., the temporary values are used when Ecoute accesses `vediamo.ini` keys as well. The exception for Ecoute are the hex coded entries which contain information on the various Ecoute windows.

If the server was started with overwritten `vediamo.ini` values, the following information is shown in the information window for every Ecoute client that takes up contact with the server (Example):

```
"Server/Application was started with partially overwritten
initialization parameters: [Ecoute] ProgramLogpath c:\Logtest\Client
[Server] ProgramLogPath c:\Logtest\Server"
```

Example:

```
Ecoute.exe /vi "[CAESAR] USE_SIPartEDriver 1 [CAESAR] PinMapping 1"
```

Starts the server with activated CAESAR Part E and activated pin mapping.

Example:

```
Ecoute.exe /vi "[CAESAR] USE_SIPartEDriver 1 [CAESAR] PinMapping 1 [Ecoute]
UseFilters 0"
```

Starts Ecoute with filter function off, and the server with activated CAESAR Part E and activated pin mapping.

If `vediamo.ini` entries are changed and written during a diagnostic session, e.g., from the Ecoute *Options* dialog, then modified key values which were uploaded at the start of the session are only updated temporarily for the duration of the session. The changes are not made permanently in the `vediamo.ini` file.

## Specify Text Language in Applications

The text language for the Ecoute applications (menus, messages, etc.) can be displayed in different languages. The translated text must be available in a DLL (e.g., `Ecoute_Res_EN.dll` for English). The language is selected using the entry `LANGUAGE` in the [vediamo.ini](#) file [Common] section.

In Ecoute, text which is not defined in the application but rather comes from the DiagServer is also output. The language version of these texts is determined by the same entry in the `vediamo.ini` file which is located on the server. The translated server text is in the `VCommon_Res_XY.dll`, where `XY` is the language abbreviation.

## Specify Language for Diagnostic Data

The texts for the ECU parameterization and messages from the CAESAR hardware can be displayed in different languages. Appropriate CTF files must be available. The language is selected using the entry `LANGUAGE` in the [CAESAR] section of the [vediamo.ini](#) file on the server.

Furthermore, the language for each ECU can be changed during runtime using the *ECU properties* dialog. This change affects this one ECU and impacts all client applications connected with the same DiagServer.

### 2.4.4.14 Macros

In Ecoute macros serve to record, store and subsequently deliver and execute certain frequently called function sequences. Macros are stored as text files with the `.mak` extension.

The macro functions can be reached using the menu selection *Extras / Macro*.

The following functions are available:

#### *Record Macro:*

After this function is called, the subsequent actions performed in Ecoute are recorded. The dialog "Record macro" is displayed at the upper right, listing the recorded macro commands. Recording is ended with "End macro recording". The name of the file in which to store the recorded macro has to be entered in a dialog.

#### *Execute Macro:*

When this function is called, a previously recorded macro file can be selected in a file selection dialog and subsequently executed. Only one macro at a time can be

recorded or executed.

The name of a macro file can also be entered as a command line parameter when Ecoute is started. Ecoute then executes the respective macro file after starting.

The names of the last three executed macros can be selected directly in the menu *Extras / Macro*.

When recording and executing macros, only the sequence of the individual function calls is considered. The time intervals between the individual function calls cannot be recorded or reproduced.

The following actions in Ecoute can be recorded and reproduced in macros:

- Select system
- ECU: Establish/terminate contact
- Read/delete errors
- Store error data in file
- Close error window
- Open measurement/actuator window, read values
- Store measurement/actuator window in file
- Close measurement/actuator window
- Adjust actuator
- Display graphically, open, read for x seconds, and store measurement values. The measurement values in the graphic window are read cyclically for as long during macro execution as the graphic window was open during recording.
- Execute all services, if necessary with preparations. Can be selected in the control tree under the menu entry *Services*. (Including Java routine)
- Variant coding: setting/coding individual fragment values of a selected coding service. (No manual coding)
- Flashing
- Quick test
- Delay: A delay interval can be inserted into a macro. First a delay entry must be inserted in the macro manually with an editor. Example, wait 5 seconds: "Delay||5000". During the macro replay the execution is interrupted with a delay line for the interval indicated there (in ms).
- End program

When replaying macros, input parameters of services can be specified alternatively manually. To the inquiry of input parameters the macro file must be worked on with an editor:

If a parameter in a macro line is to be queried, the sequence "???" must be specified in place of the parameter as a marking, afterwards optionally a text, afterwards again optionally "???" and a default value.

The text serves as note for the user. Example: Macro line after the macro recording:

...

```
SetInputParam||CR3||ADJ_Injektorklassierung_ZYL4||0||Kl. 3  
ExecuteService||CR3||ADJ_Injektorklassierung_ZYL4
```

...

Now the input parameter 0 of the service ADJ\_Injektorklassierung\_ZYL4 is to be queried each time during the execution.

The macro file can be adapted in such a way:

...

```
SetInputParam||CR3||ADJ_Injektorklassierung_ZYL4||0||??Inj.Klassierung Zyl.4:??Kl. 3  
ExecuteService||CR3||ADJ_Injektorklassierung_ZYL4
```

...

Macros can be selected as an initialization service. Macros with the special name <SystemName>\_SysInit.mak and <ECU Qualifier>\_EcuInit.mak are treated as initialization services by Ecoute. When a new system is loaded, Ecoute checks whether a macro ...SysInit.mak is in the current system directory and execute it if available. If contact has been established to an ECU, Ecoute checks whether a macro ...Ecunit.mak is in the current system directory. If so, it is executed. The name of the system or the ECU qualifier must precede the names of these macros.

#### 2.4.4.15 Java Routines

Routines (Java Routines or processes, called scripts in earlier versions), if at hand, are displayed in the selection window in their own directory. They can be started just like every other service with a double-click (or select and press <ENTER>). The precondition for proper execution is that Java Runtime is properly installed and all settings in the INI file are correct as well as the Java program that implements the routine is correct.

Every Java routine can have predefined command line params stored in the system descriptioni file. By double-clicking the routine in the system window, it is started with the predefined params. It is possible to have several entries starting the same Java file, but with different params.

To start a Java routine with params different than stored in the system description, call the context menu by right-clicking the routine symbol. An input window will appear where you can overwrite the original parameters.

As soon as a Java routine starts, a window opens showing active Java routines. This window can be used abort the Java routines if necessary.

The outputs and user interactions depend on the executed Java routine. Please read the documentation on [handler functions](#) for the possibilities offered by a Vediamo Java routine. The basic possibilities for output from or user interaction via a Java routine are:

- Outputs in the Ecoute status window
- Receive yes/no query responses from user
- Receive text input from user
- Signal tone

#### **2.4.4.16 Routine Generator**

The routine generator allows to create and edit complex routines using a GUI without the need of knowing Java. The routine generator appears as a window in the Ecoute Application. It consists of four areas (symbol collection, drawing sheet, properties dialog, tool bar). It can be opened beside the other Ecoute windows and therefore services from the system window can be dragged and dropped directly into action elements.

To open the routine generator with a new or an already existing routine, use the menu *System / Routines / New Routine* or *System / Routines / Open Routine*.

A routine can also be started directly from the system tree window or from the menu, when defined as standard objects.

#### **2.4.4.17 Use Cases**

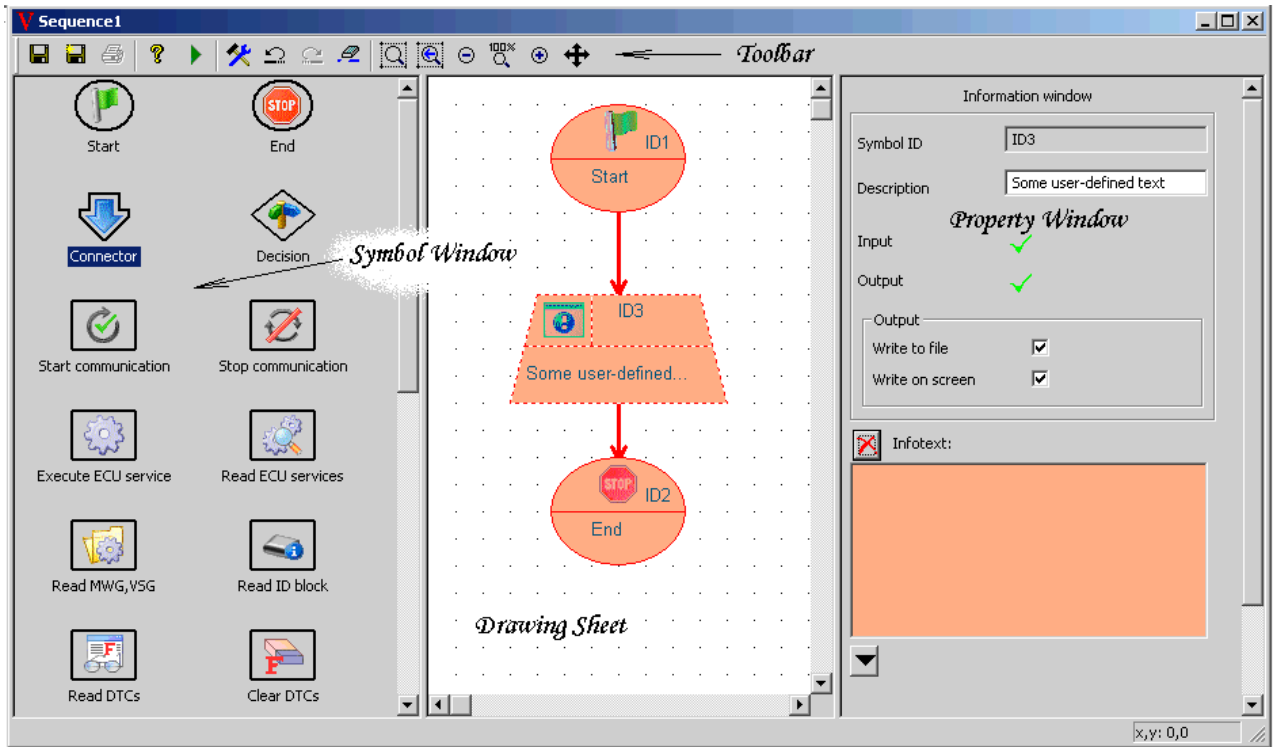
the routine generator allows the user to define complex routines by graphically designing a flow chart consisting of the following functions:

- one-time or cyclic reading of data
- setting adjustments and controlling actuators
- decisions based on the results of other actions or on user input
- getting input from the user or from a text file
- delays and time stamps
- using variables and assignments
- regular expressions











#### **2.4.4.18 Elements of the Window**

The window of the routine generator consists of a frame with controls (Toolbar) and three main areas:






- left area - Symbol window
- center area - drawing window
- right area - property window



The control elements are grouped in the tool bar. Depending on the state, some of them may be inactive to prevent wrong usage. The following table describes all controls of the routine generator.

<b>Toolbar of the routine generator</b>	
	Save Button - save routine in a *.xml file without changing the file name.
	Save As Button - save routine in a new *.xml file
	Print Button - prints the entire routine or the content of the drawing sheet.
	Validate Button -manual control/validation of the routine basing on the scheme. The validation result is visualized graphically.
	Execute Button - execute the routine.
	Options Button - Opens a window to display and edit the options of the routine generator.
	Undo Button - undo up to 10 actions performed by using the controls..
	Redo Button - redo the undone actions.
	Delete Button - delete action sybbols in the sheet.
	Zoom Fragment Button - Zoom a fragment defined by the user.





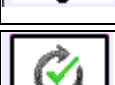








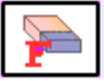












	Zoom Button - Display the last fragment.
	Zoom out Button - shrink the sheet
	Zoom in Button - enlarge the sheet
	Toggle Button - Zoom off / display entire area.
	Move fragment Button - Moves a defined area of the sheet.

### Symbol window (left area):



The left area contains all usable action symbols. The symbols can be individually reordered using the button *Options*. Drawing the mouse cursor over a symbol displays a tool tip.



The symbols can be inserted (by drag&drop) into the drawing sheet. Rotating and resizing is not possible. After inserting, the right area displays the property window of the new action element.

Action Symbols of the Routine Generator	
	Starting point of the routine. Must be used exactly once.
	End point of the routine. Must be used exactly once.
	Connection between two actions. Determines the order of execution.
	Compares two values - results of an action or user input - using predefined operators. the comparison result yes/no determines the execution of one of two paths in the diagram.
	Establish contact to an ECU.
	Close contact to an ECU.
	Execute a diagnostic service. The result of the service is stored for later usage.
	Read several measurement data.
	read data from a VSG or MWG file..

	Read the ID-Block of an ECU.
	Read the Diagnostic Trouble Codes of an ECU.
	Clear the DTCs of an ECU.
	Open an edit window. The entered text is stored for further usage in the routine.
	Display a message box with a user defined message.
	Opens a text file and stores its content for further usage.
	Opens a message box and requests the user to make a decision. The result yes/no is stored for further usage.
	Stops execution for a defined delay.
	Creates a time stamp. The value in ms is stored for later usage.
	Output some text into a file or the status window.
	Opens a text file, reads and stores its content for further usage.
	Defines a variable with value 0. To change the value, an assignment must be used. The value is stored for later usage.
	Assigns a value to a variable
	Filters a text by a regular expression. The filter result is stored for later usage.
	Executes another routine.

The frame of any action symbol has a special meaning:

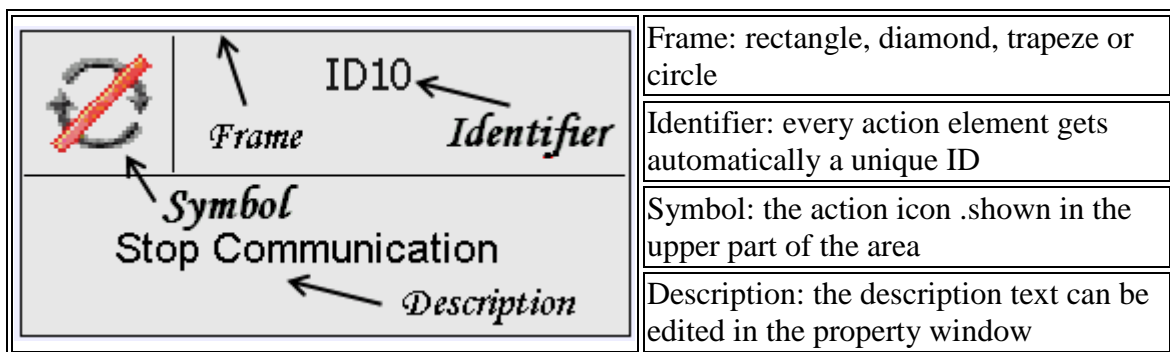
-  Rectangle means action to be executed
-  Diamond means a decision (yes/no)

-  Trapeze means displaying a dialog window
-  Start / end of a routine

### Drawing Sheet (central area):

In this area action symbols can be inserted, connected, reordered and edited. The raster can be switched on and off. If the content exceeds the window, a scrollbar allows to move it.

### The Action symbols



### Property Window (right area):

When a symbol in the drawing sheet is selected, the properties of the element can be set in the property area on the right side. There you can change p.e. the description, logging, output or input.

When no action symbol is selected, the properties of the sequence are displayed.

**Important:** There are two types of routines: protected and unprotected. The type can be chosen in the *Save As* window.

Unprotected:

The routine is saved in a normal XML document file. It can be read, edited and stored by any user.

Protected:

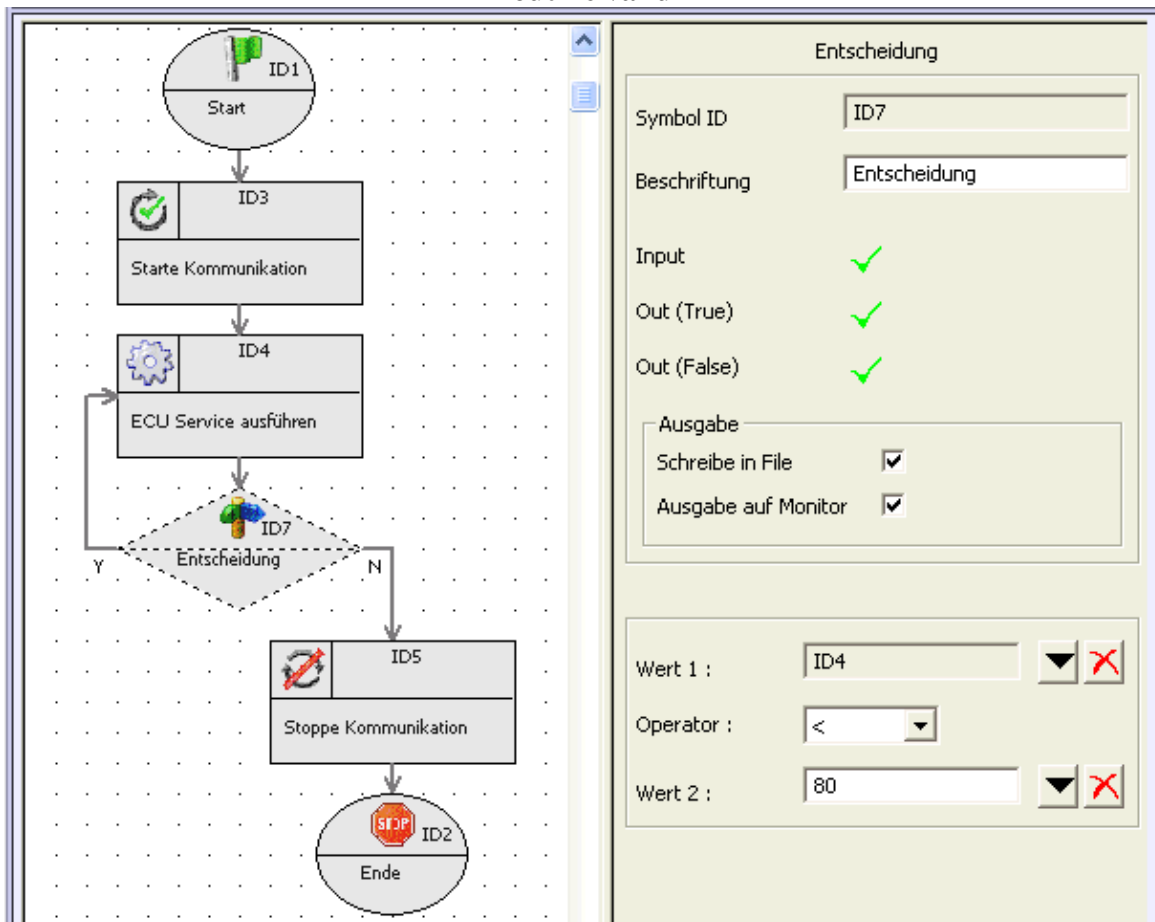
The routine is saved in an encrypted XML document file with a CRC sum. A password can be set. The routine can be protected against reading or writing (editing).

### Validation

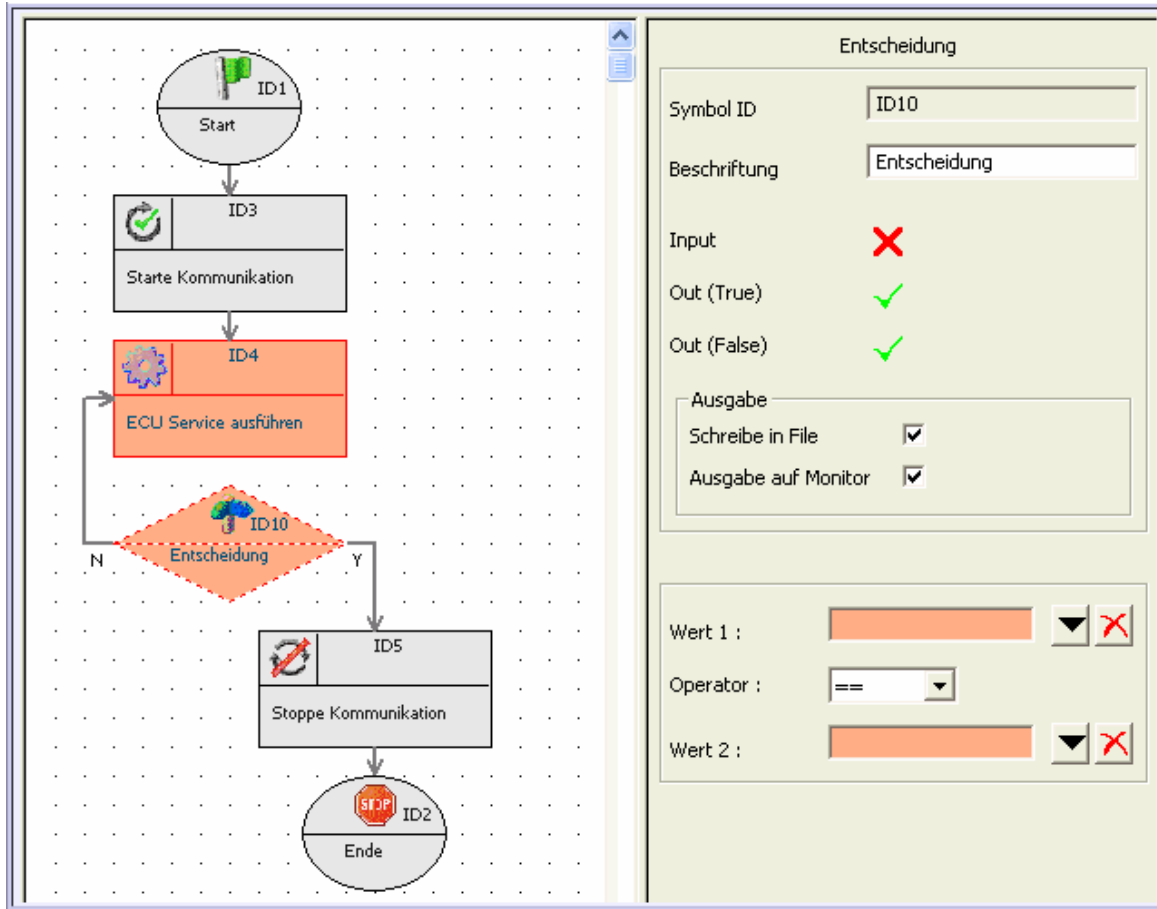
During editing and after loading the routine is validated. The result is visualized graphically.

- Routine is valid
  - the action symbols are grey. In the property window all connections have green OK-symbols.
- Routine is not valid
  - erroneous action elements have a red background. In the property window a red cross marks which property is not correct. Input fields with red background are obligatory.

### Routine valid



## Routine not valid



### Automatic Validation

Automatic validation after every change can be switched on or off in options

(button  in the toolbar)

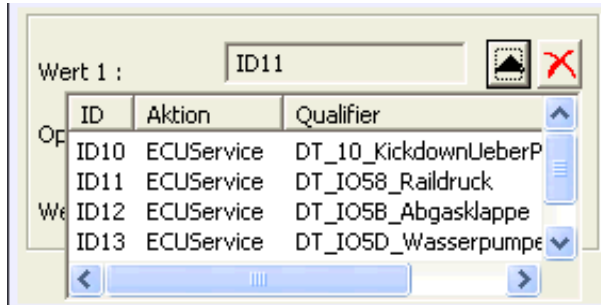
### Manual Validation

When automatic validation is switched off, it can be performed manually by clicking on


the button  in the toolbar.

### Property Window - Using the Return Values Of Other Actions

Depending on the focused action symbol in the drawing sheet, the property window contains context sensitive input fields. Here you can use the results of other actions or let the user input values manually at runtime.



In any actions, where the result of other actions might be necessary, a list box allows to choose out of all available results. A selection of one of the results overwrites the content of the text field. When a result is chosen, the field cannot be edited manually. It can be changed only by selecting another result from the list box.

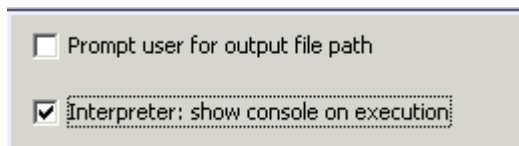
The button  deletes the content of the text field and allows entering values manually.

## Saving the Results

During the runtime, all performed actions can be logged into a file and/or in console. By default, both options are set. They can be changed in the property window of every action.

<p>Output</p> <p>Write to file <input checked="" type="checkbox"/></p> <p>Write on screen <input checked="" type="checkbox"/></p>	<p>Write to file: setting this option causes the actions to be logged into a file. The default path of the file is the directory of the sequence file. The file name consists of the file name of the sequence with a time stamp, and the extension LOG.</p> <p>In the property window of the routine, the options for the log file path can be changed. At the end of the routine execution, a file open dialog appears requesting the user to choose a path and file name.</p> <p>Write to screen: Setting this option causes all log information be displayed in the status window of Ecoute.</p>
---	--

Changing the default log file path:



Log file path:  
To overwrite the default log file path name, open the routine's property window (click on an empty area of the drawing sheet). Selecting the

option "User input for log file path" causes an opening of a file dialog after executing the routine. In this dialog the user can choose the path to save the output log.

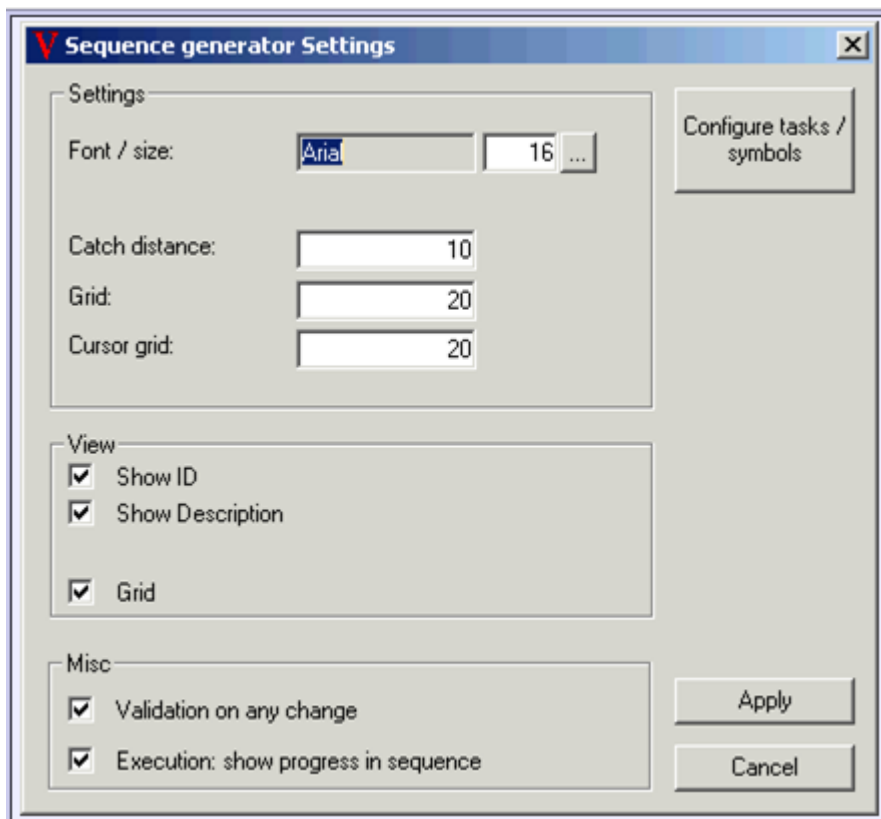
Open a console:

The option "Open console during execution" in the routine's properties causes a separate log window to be opened for log output.

## Configuration Of The Routine Generator

Clicking the button "routine generator options" in the toolbar opens a window with several options. Especially the button "Configure action symbols" allows to re-sort, display or hide the action symbols.

The other option names are self-explaining.



#### 2.4.4.19 Standard Objects

Standard objects (e.g., unlock) are often referred to as aliases because they reference another service (e.g., depending on the ECU, the standard object *Unlock* is followed by the DiagJob DJ\_Entriegeln or the service FN\_Zugriffsberechtigung) and give this service a common name. This makes it possible to give services with the same functional content the same name, despite different names in the [DIOGENES data](#). Standard objects are available on two different levels. In one case there are system specific standard objects under the ECU system. These standard objects generally affect multiple ECUs. In the other case, independent standard objects can be defined in the [Vediamo system configuration](#) for individual ECUs.

Standard objects receive special treatment in Ecoute. For each object entered in the subdirectory *Name*, a menu entry is generated in the menu *Name*. If no menu with this name exists, it is created. This makes it possible to customize the Ecoute interface using standard objects and assort frequently used services (including Java routines) in their own group.

To execute a standard object, either start it from the selection window (double-click or <ENTER> or select the corresponding menu entry.

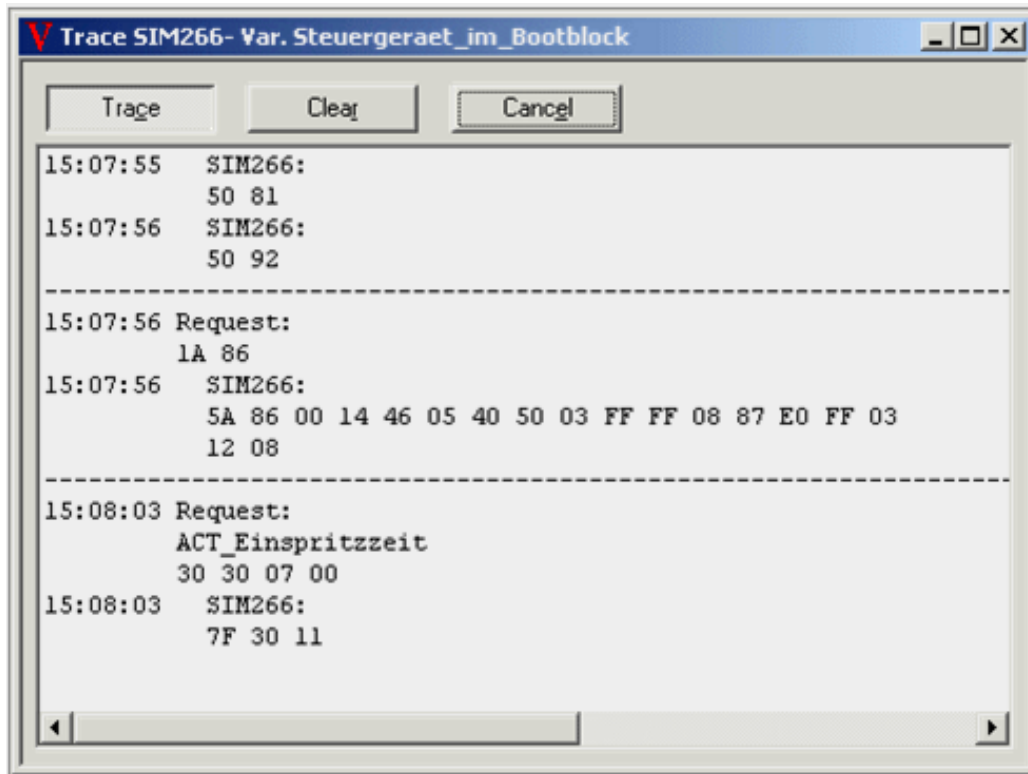
Important:

To create a menu entry with an underlined character (hotkey), an "&" must precede the letter in the name. If you enter a standard object "&Special" in the directory "&Errors" in the system file (more on this in the System Configuration section), the additional item *Special* which reacts to the "S" key is displayed in the Ecoute *Error* menu. If the "&" is not included, a second menu named *Errors* (without an underline) is created and the entry *Special* has no hotkey.

#### 2.4.4.18 Display Trace and Monitoring Data

The function *Extras / Trace window* allows ECU-related communication information with the respective ECU to be displayed in a separate window. The display can be stopped or started again. The displayed information is stored automatically in a logfile. If multiple ECUs are available, the appropriate ECU must be selected after the function has been selected. The communication information for this ECU is displayed in the following window and format:





The window is closed by using the button *Cancel*.

The button *Trace* can stop or start the output display. When *Trace* is activated, the button is displayed as pressed in. When the trace is stopped, the button pops up again.

The complete contents of the window can be deleted with the button *Clear*.

The key combination CTRL+C copies any text selected in the window to the clipboard for further processing.

The output is automatically written to a file named <ECU>Trace.log (<ECU> is the name of the ECU, e.g., ME20). Any new information is appended to the end of this file.

## Channel Monitoring

This function allows a trace window to be displayed for any available CAESAR connection. Ecoute/Vediamo is in "listen only" mode regarding the ongoing communication in this case, i.e., no data is sent to the ECU.

The data is always displayed in the format "Bytes & Timing".

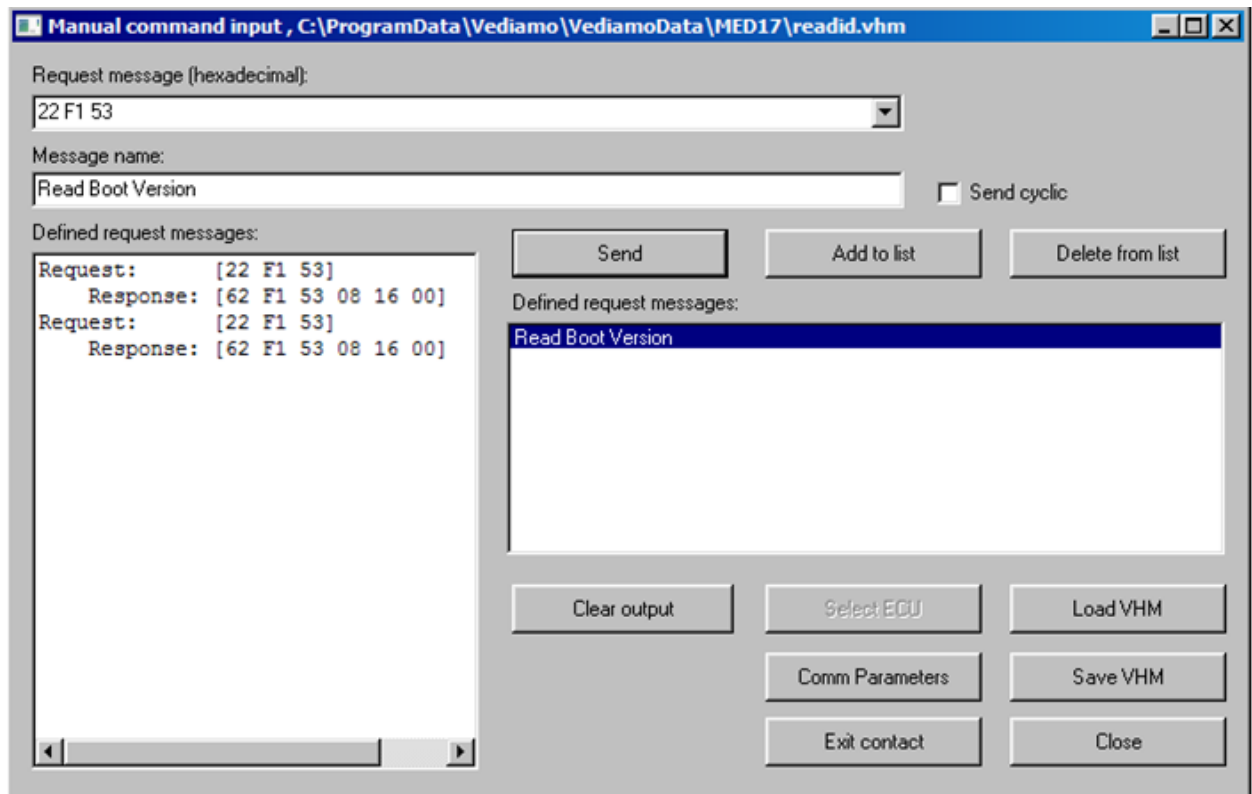
After starting the function the CAESAR protocol, the connection and if a CAN protocol should be used the baud rate must be specified.

Important:

Channel monitoring requires a CAESAR resource. If, e.g., K-line 1 is used, this resource is not available for normal diagnosis until the monitoring dialog is closed.

#### 2.4.4.20 Manual Command Input

Manual command input is opened using the menu entry *ECU / Manual Command Input*.



The individual components of the manual command input interface:

Input/output field *Request message (hexadecimal)*

- Input messages:  
Enter the message to be sent in hexadecimal form (byte values). You can store messages in a [VND](#) (see: add to list). In addition, the preceding messages are available.
- Display of defined messages:  
If you have loaded a VND, the bytes of the message selected in the *Defined request messages* list are displayed here.

Input/output field *Message name*

- Input messages: Input a name for the message defined in the *Request Message* input field.
- Display of defined messages: If you have loaded a VND the name of the message selected in the *Defined request messages* list are displayed here.

Option *Input payload only*

The option *Input payload only* specifies how the input of the input line should be treated - either a message with the entered content is sent or a protocol compliant message is created first and the entered bytes are contained as the payload.

Option *Display payload only*

The option *Display payload only* specifies, how the contents of the received messages should be displayed - either only the payload, or the complete messages including header and checksum.

Option *Send cyclic*

The option *Send cyclic* specifies, whether the sent request message should be automatically resent after the (last) reply from the ECU or not.

Button *Send*

Request message can be sent to the ECU using the *Send* button. During reception or cyclical sending and receiving, the *Send* button has the label and function *Cancel*. Response message series with which the ECU reacts to some requests can be terminated with this button. After the last message (or the termination of the reply series), the button regains its original label and function.

Button *Add to list*

The button *Add to list* can be used to insert new defined messages in the VND. Put in the request message first and then the corresponding message name.

Button *Delete from list*

Use *Delete from list* to delete the message selected in the *Defined request messages* list. If you subsequently store the VND, the deleted message is permanently removed from the VND.

Output field *Sent and received messages*

All communication telegrams are displayed in the output field *Sent and received messages*. The output is dependent on the setting of the option *Display payload only*.

Output field *Defined request messages*.

This output field *Defined request messages* displays the currently defined request message.

Button *Select ECU*

If the opened system has more than one ECU, the ECU to be contacted can be selected using the *Select ECU* button (see *Establishing contact for an opened system*). For unopened systems, this button opens an ECU parameter input dialog in which the protocol and ECU address can be entered (see *Establishing contact without an opened system*)

Button *Load VND*

Use this button to load a VND.

#### Button *Establish contact*

The button *Establish contact* controls the contact to the ECU and shows the current communication status. The button has two states:

*Establish contact*

ECU is not initialized. Use the button to establish contact.

*End Contact*

ECU is initialized. Use the button to end the contact.

#### Button *Comm parameters*

Protocol-specific communication parameters can be set using the *Comm parameters* button.

#### Button *Save VND*

After pressing *Save VND*, a *Save file as* dialog opens. Select an appropriate directory for your VND and store it under the filename of your choice.

#### Option *Monitor*

Turn the monitoring trace on the diagnostic server on and off with this option.

#### Button *Close*

End the manual command input with *Close*. The state of contact with the ECU before the window was opened will be restored when closing the window.

Manual command input can be opened independent of having a system loaded or not. The processes for establishing contact differ, however. The two cases are therefore described separately in the following.

### **Manual Command Input, Establish ECU Contact with an Opened System**

- If only one ECU is available in the system, contact can be established using the *Establish contact* button.
- If multiple ECUs are available in the system, use the *Select ECU* button. The *Select ECU* dialog opens. Select the ECU which you like to contact. Then press the *Establish contact* button.
- Once contact with the ECU is established, the label on the button changes to *End contact*. In addition, the message `Contact to ECU <ECU name> established.` appears in the status window.
- If contact to the ECU cannot be established, the label on the button remains *Establish contact*. In addition an error message is displayed in the status window (e.g. `ECU cannot be initialized (activation failed). Reason: ComCoordinator: 02005: Timeout P2`).

#### Important:

With an opened system, the communication parameters (ComParameter) from the corresponding CAESER / DIOGENES data (CBF file and GBF file) are used, i.e., usually no communications parameters need to be set in order to establish contact.

## Manual Command Input, Establish ECU Contact without an Opened System

If no system is opened and you want to establish communication with an ECU, there are two different cases:

1. VND available:
  - Press the *Load VND* button. Select the desired VND using the *Open file* dialog.
  - Select the connection
  - Press the *Establish contact* button.
  - Once contact with the ECU is established, the label on the button changes to *End contact*.
  - If the contact to the ECU cannot be established, the label on the button remains *Establish contact*. In addition, an error message is displayed in the status window (e.g. ECU cannot be initialized (activation failed). Reason: ComCoordinator: 02005: Timeout P2).
2. VND not available:
  - Press *Select ECU* and select a protocol. A dialog window is displayed in which protocol-specific communication parameters can be set
  - Enter the values for the displayed communication parameters and click *OK*
  - Press the *Establish contact* button.
  - Once contact with the ECU is established, the label on the button changes to *End contact*.
  - If the contact to the ECU cannot be established, the label on the button remains *Establish contact*. In addition, an error message is displayed in the status window (e.g. ECU cannot be initialized (activation failed). Reason: ComCoordinator: 02005: Timeout P2).
  - Store your settings in a VND.

## Manual Command Input, Enter Communication Parameters

Pressing the *Comm parameters* button opens the *Communication parameters* dialog.

This dialog lists all the valid communication parameters for the ECU. The names of the individual communication parameters are shown in the list *Parameter names*.

To change the value of a communication parameter, proceed as follows:

- Select the communication parameter to be changed in the *Parameter name* list. The entry field now shows the current value of the communication parameter.
- Change the value of the communication parameter to the desired value (decimal).
- Press the *Set* button. The communication parameter is set to the value you have specified.

The button *Default* sets all communication parameters back to their default values. The default values are those which were set when contact has been established (successfully) the first time or the values read from the GBF file if no contact has been established with the ECU yet. Example:

Initialization communication parameter ME20: KLINE protocol  
CP\_TRIGADDRESS = 1  
CP\_RESPSOURCEBYTE = 1  
CP\_RESPONSEMASTER = 1  
CP\_REQTARGETBYTE = 1

Example:

Initialization communication parameter CR3: CAN protocol  
CP\_REQUEST\_CANIDENTIFIER = 2016  
CP\_RESPONSE\_CANIDENTIFIER = 2024  
CP\_BAUDRATE = 500000

## **Manual Command Input, load VND**

A VND (Vediamo-Nachrichten-Datei = Vediamo message file) serves to store ECU and communication parameters required for establishing contact with an ECU, as well as a list of frequently used messages and names.

If such a file is available, it can be loaded using the *Load VND* button. A standard *Open File* dialog opens and all files ending with VND are displayed.

After selecting the VND, the IDs of the messages it contains are displayed in the *Defined request messages* list.

When working with an ECU selected from the system (with a VSB file loaded), the parameters contained in the VND file are ignored; if no system is open, the ECU and communication parameters are set to the values contained in the file.

The next time manual command input is opened, the VND file used last is automatically opened.

## **Save VND Files**

A VND file with the current ECU and communication parameters as well as the defined request messages is saved when the *Save VND* button is pressed (after entering the filename in a standard file dialog).

If messages and/or parameters have been changed, the user is offered to save the changes in a VND file while closing the *Manual input entry* dialog.

## **Define Messages**

The user has the possibility to enter the contents of a message to be sent in hexadecimal format (input line *Request message*). He has the choice between entering a complete

message and only entering the payload (option *Enter payload only*). A name for the message can be input in the *Message name* line (any text). Pressing the *Add to list* button copies this message to the list of defined messages (only names are displayed in the *Defined request messages* list box). A defined message can also be modified (by renewed entering) or removed from the list (by pressing the button *Delete from list*). The defined message can subsequently be stored in a VND file.

## Send Messages to the ECU

Messages can be sent to the ECU a number of different ways. As a rule: When the button *Send* is pressed, the message shown in the input line is sent.

The message in the input line can be either:

- put in directly
- selected from the history of the input line
- or selected from the list of defined messages (in this case the selected message is copied to the input line).

The *Send* button is the dialog's default button. This means that it can be activated not only by pressing it directly, but also by hitting the return key or by double-clicking an entry in the list of defined messages.

If only payload bytes rather than the complete message have been entered, the program creates a complete message from the payload with the correct checksum byte to send to the ECU.

An entry with the sent message then appears in the windows *Sent and received messages*. When the ECU responds to the message, this/these message(s) are also displayed. The User has the choice whether complete messages or only payloads are displayed.

## VND Files

Vediamo message files contain information for manual command input. They are editable ASCII files. The structure of the files corresponds approximately to that of an INI file. Comment lines are possible, beginning with "/" to the end of the line. Spaces are ignored. Since the parameters are ECU specific, the file is only suited for ECUs of the same type.

The files consist of three section:

- ECU parameters
  - Communication protocol
  - ECU address
- Initialization communication parameters
  - Communication parameters required by CAESAR in order to establish the communication with the ECU (e.g., CP\_TRIGADDRESS)

- Message definitions.
  - Request telegrams and their IDs (e.g., read ID = 3C 00 )
  - The message ID is listed first in line, than an "=" and the hexadecimal payload, separated by blanks. If the bytes are in parentheses, the bytes are to be interpreted as a complete message, otherwise they are to be considered as a payload.

### **Example for ECU Parameters**

Example:

[ECU]

Protocol = KW2000P

Address = 01

The parameter "Pin" becomes decimal and "Protocol" is specified as a keyword. "Address" and other communication parameters are entered in hexadecimal.

### **Example for Initialization Communication Parameter:**

Example:

[COMMUNICATION]

CP\_REQTARGETBYTE=69

CP\_RESPONSEMASTER=69

CP\_RESPSOURCEBYTE=69

CP\_TRIGADDRESS=69

### **Example for Message Definitions**

Example:

[MESSAGES]

Read ID = 3C 00

Read error = (81 01 F3 1A 8F)

### **Expert System**

Manual command input gives the user the capability to submit any requests to the ECU directly on the byButton "Send / Stop sending": This button has two states. When clicked, it becomes and stays pressed, the messages are cyclically sent. Clicking it again changes the state to de-pressed and stops sending. Button "Close": If any messages or parameters have been changed, the user is requested to save the contents in a VRS file.

For loaded systems, it is also possible to pull any desired service from the tree view per drag and drop to the manual command input window.

The request stored in the diagnostic parameterization of the selected service is then displayed in the manual command input. The qualifier of a deposited service is entered in



the field *Message name*, the request message in hex format (if it can be determined) is entered in the field *Request message*.

If the service is deposited in the list field *Defined request messages*, it is also automatically added to the list of pre-defined messages.

The user has the possibility to change or manipulate individual bytes before submitting the request, and therefore can perform specific tests with the ECU.

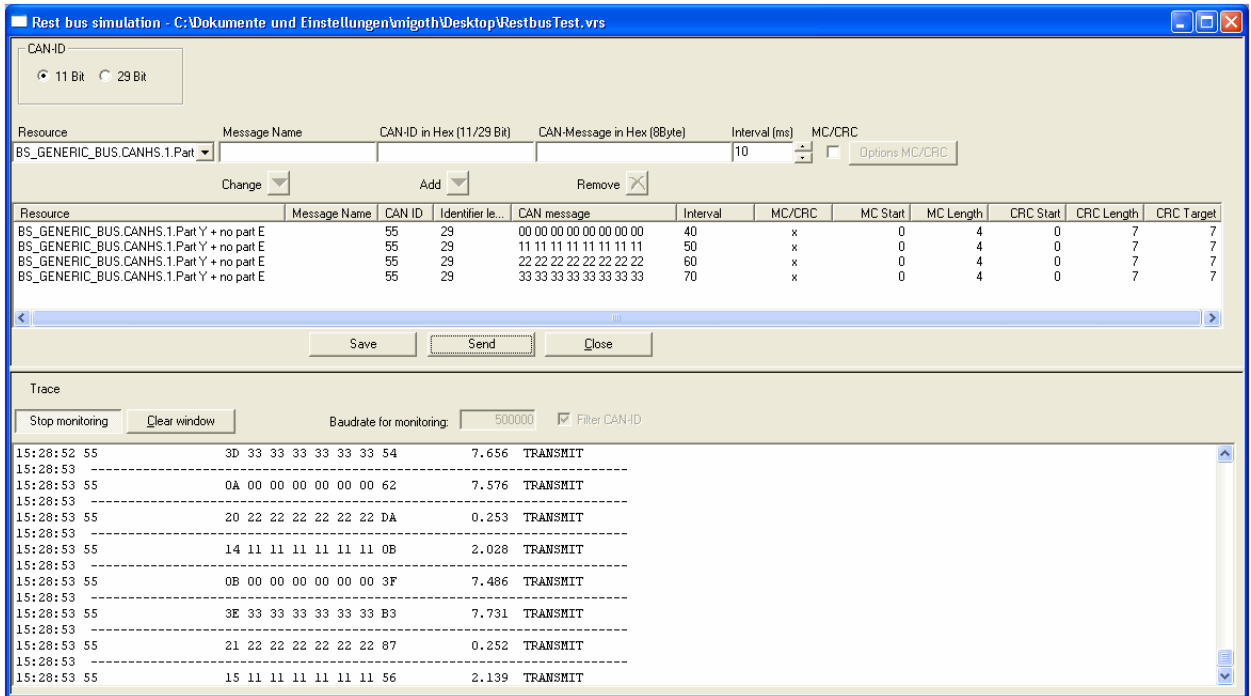
In addition, any hex byte sequence can be entered in the *Request message* field using copy & paste (e.g., out of the editor).

#### **2.4.4.21 CAN Bus Simulation**

This function allows to simulate CAN messages of ECUs not attached to the network. This is needed for testing ECU functions expecting special messages from other ECUs without the need to have a model of the complete network.

The function can be activated with the menu item *ECU / CAN Bus Simulation*. A window opens in which the following parameters can be set:

- CAN-ID (Hexadecimal)
- Message to be sent (Hex) - either 11 or 29 bit long
- Time interval in ms
- Ressource
- Message counter
- CRC checksum



**Parameter "*CAN-ID*"**:

Setting the length of the CAN ID to 11 or 29 bit. Default is 11 bit.

**Parameter "*Ressource*"**:

The list box contains currently available CAN resources:

- HSCAN: BS\_GENERIC\_BUS.CANHS
- LSCAN: BS\_GENERIC\_BUS.CANLS

The first one is used by default.

**Parameter "*Message Name*"**:

Name of message that can be defined by user.

**Parameter "*CAN-ID in Hex*"**:

Enter the hexadecimal CAN-ID here.

**Parameter "*CAN-Message in Hex*"**:

Enter the hexadecimal CAN message here. The length is limited to 8 byte (64 bit payload).

**Parameter "*Interval*"**:

Enter the time interval (in ms) in which the message should be repeated. Change the initial value of 0 to a positive number before pressing the "Send" button. Changing of the interval while sending is active, is not possible.

### Checkbox and Button "MC/CRC and Options MC/CRC":

This feature enables the generation of a message counter and / or a checksum for the related CAN message. The "options" button displays a dialog where the necessary parameters can be configured: For the CRC, the area must be specified for which the CRC should be calculated, and the byte position where the CRC will be stored in the CAN message. For the messagecounter, the bit position and it's length in bits must be entered. Already specified parameters are displayed in the related list entry. If a parameter is omitted / remains empty, the related function (MC bzw. CRC) will not be activated.

**CAN bus simulation - options (MC/CRC)**

Message Name: TEST

CAN-Message: 11 11 11 11 11 11 11 11

Message Counter (MC), values in bits

Note: the following values belong to the 8 bytes long CAN message. START: bit position of MC in the CAN message. LENGTH: length of the MC in the CAN message, beginning at START. The counter must not cross byte boundaries.

START: 0 bit (0...63)

LENGTH: 8 bit (1...8)

Checksum (CRC), values in bytes

Note: START: position of the first byte of the area on which the CRC shall be calculated. LENGTH: length of the area in bytes. TARGET: byte position, where the CRC shall be stored in the CAN message.

START: 0 in byte (0...7)

LENGTH: 8 in byte (1...8)

TARGET: 6 in byte (0...7)

OK Cancel

### Button "change":

By use of the button "change" the change message entry into the list of messages to be sent. **Button "add":**

Use this button to add the edited message entry into the list of messages to be sent. Depending on hardware used, up to 10 independent messages can be configured and simulated.

**Button "remove":**

Use this button or the "delete" key to remove the selected entry from the list of messages.

**Button "Send / Stop sending":**

This button has two states. When clicked, it becomes and stays pressed, the messages are cyclically sent. Clicking it again changes the state to de-pressed and stops sending.

**Button "Close":**

If any messages or parameters have been changed, the user is requested to save the contents in a VRS file.

**Button "Save":**

The full configuration can be saved in a \*.vrs (Vediamo Rest bus Simulation). A "file save" dialog will appear requesting to select path and file name. The default file name is ...\\VediamoDaten\\yyyy-mm-tt\_[Corrent/last system].vrs

To use this file, load it by menu item "ECU / Load CAN bus simulation".

**Trace Analysis:**

After setting or loading the configuration and starting sending, the trace window shows the data flow as single bytes with a time stamp. The window content cannot be saved.

The toggle button "Start/Stop monitoring" is used to activate/deactivate monitoring.

The button "Reset window" clears the content of the monitoring window.

For a better readability, use the checkbox "filter CAN-ID" to display only messages of the simulated ECUs.

**Important:**

The CAN bus simulation is an extension to standard communication functionality.

Therefore not all communication hardware supports this feature.

Especially the eCOM (PDUAPI) does not support CAN bus simulation.

#### 2.4.4.22 Snapshot File Storage

Snapshots of the error, actuator and measurement windows can be stored in a file by pressing the button *Save* or the key combination CTRL + F2. A dialog appears when *Save* is pressed in which the user can enter data, comments and the name of the snapshot file. A recommended name for the file appears in the dialog. This name can be accepted or changed. (Button ... directly next to the input field *Snapshot* file).

The default file format of the snapshot file is HTML. Snapshot files can be logged either in HTML or in text format. The choice is made using the file extension. The ".htm" or ".html" endings determine HTML format. In all other cases a pure text file is saved.

All snapshots to be saved are stored with the ID block entries in the snapshot file. The snapshots can be attached to already existing files. The last used snapshot file is displayed when the dialog is opened. The snapshot can be written into the file by acknowledging with *OK*.

The following information is stored in the file when it is saved for the first time:

- Version, e.g. Ecoute version 05.00.00
- System name
- ID block
- CBF version
- User data (to be filled out by user)
- Comments

The ID blocks of all ECUs available in the system contacted during a session are saved. If no communication exists as the target file is selected, the ID blocks are saved when communication is established. If a communication is terminated and re-established, a check is made during communication is being established whether the ECU ID block is already included in the file. In that case this ID block is not saved. Otherwise this ID block is written into the file (ECU swap).

If the target path has already been set, the error, actuator and measurement window snapshots can be stored in the file using CTRL + F2. No comments or user data can be entered in this case.

#### 2.4.4.23 Simulation of ECU Communication

Communication with ECUs can be simulated in Vediamo without actual physical contact to an ECU.

##### Simulation Modes

Simulation mode in the server can have three different states:

State	Mode	Description
Normal	0	Normal state, communication with a physically present ECU, default setting.
Simulation	1	ECU communication is simulated.
Record simulation data	2	Communication with the physically present ECU is recorded in order to play it back in simulation mode.

The states are controlled using an entry in the [Vediamo.ini](#) file:

```
[Server]
Simulation=<Mode>
```

This `vediamo.ini` entry becomes effective after the server starts or after a server reset with subsequent re-initialization. The entry can be changed permanently in `vediamo.ini` using the INI Editor. You can also start Ecoute with simulation option in the Start Center.

Remember however that the simulation mode concerns the server itself, not a single client. Therefore it is not possible to work in simulation mode with one client when another client already started without simulation.

In simulation mode, Vediamo can be used without any CAESAR hardware. No password is requested when starting the server in simulation mode and the CAESAR hardware ID is checked.

The CAESAR communication API is not called in the simulation. API-I calls (Ecoute: manual command input), reception of monitoring data ([Trace](#)), as well as editing COM parameters are therefore not possible.

Each client can check whether the system is in simulation mode. This state is clearly displayed in Ecoute and in the machine operator by the text "SIMULATION" in the title of the main window.

The service filters described in the system description will function without modification in simulation mode.

## **Recording Simulation Data**

A mechanism is implemented for recording the ECU behavior during regular diagnosis and playing it back during simulation.

SIM data recording is activated by the `vediamo.ini` entry `[Server] Simulation=2` when the server is started or re-initialized.

The recorded data is stored in the appropriate SIM files in the system directory.

The following data is recorded:

- ECU
- Last variant
- Services
- Qualifier, Result
  - If a service is executed numerous times, the sequence of the up to 10 last occurring results are recorded.
- Errors
  - The P-codes of the errors from the last "Read errors" process and their environment data (if read errors with environmental data is activated) are recorded.

SIM files which are not (yet) available are generated during SIM data recording. Already existing files are updated.

The recorded data is written into the affected SIM files at the latest when the current system is closed.

It cannot be ruled out that the performance of Vediamo is negatively affected by recording mode.

## Simulation Data

The simulation data is stored in easy to edit text files for which the following applies:

- There is one SIM file per ECU in the system directory. If a system description contains multiple ECUs with the same name, these use the same file. The name of the file is formed from the ECU qualifier and the extension ".Sim", e.g. "ME28.Sim". It is not possible to select or modify the filename or the file from the client. Changes can only be made by copying, shifting, editing etc. directly in Windows. The file is read anew with each ECU:Init() by the server.
- All required data such as names, descriptions, dimensions and value limits, etc., not contained in the SIM file are taken as possible from the available parameterization.

## ECU Settings

The following functions are simulated:

- ECU: Init() and Exit()
- The variant entered in the SIM file is used. If no SIM file is available, the default values of the basic variant are used. The function GetIDBlock() returns only the block ID. A menu with the functions relating to the ECU is displayed by clicking on the ECU entry in the system tree in Ecoute. In simulation mode, this menu is expanded to include the function *Select variant*. This menu entry is displayed only in simulation mode. Executing the function in simulation mode causes a dialog with a selection of all the ECU's parameterized variants to be displayed. After a variant has been selected, it is activated internally in the Vediamo server for simulation. The system tree is updated accordingly.

Entries in the SIM file (Example):

Example:

[General]

Version=1

SIM file version

;ECU=CRV

Comment, for information only. The ECU is determined by the name of the SIM file.

Variant=C47

Variant to be simulated. Default: empty= basic variant

Error="TimeOut P2"

Default: empty. If a value is entered here, if establishing contact in the simulation fails, the entered value is returned as error.

ECUInitDelay=100

Delay in ms required to establish contact in the simulation. Default=0.

ReadAllErrorsDelay = 500

Delay in ms of a simulated "Read errors" operation. Default=200.

ServiceExecutionDelay = 0

Delay in ms of a simulated "Execute service" operation. Default=200.

## Results from Diagnostic Services

When a service is executed, the value specified in the SIM file is returned as the result. If no value is specified in the SIM file, "?" is returned. It is also possible to specify a list of values separated by commas in the SIM file. In this case, the values are returned sequentially, the first value for the first call, the second value for the second call, and so on. When the last value in the list has been reached, the subsequent call returns the first value again.

If a preparation is entered in the Ecoute dialog, it is checked for validity but has no influence on the result of the execution.

The dimension of the result can be taken from the parameterization.

The relation between set value and a possibly available [OutputRef](#) is not simulated. The corresponding entry in the SIM file applies to the simulation of [presentations](#) of

- general services
- measurements
- actuator and adjustment states
- functions

The entries for all of these Vediamo classes are identical in the SIM file.

Entries: An entry in the SIM file is provided for each service to be simulated.

Example:

[DT\_01\_Oeltemp]

Service entry.

Value=85.0

Single value.

...

[\_S\_DT\_01\_Bordspannung]

Next service entry.

Value=12.4,12.3,12.8,13.4

List of values.

...



## Read Errors

Read errors returns the P-codes specified in the SIM file and if applicable, the list of environment data with the values entered in the SIM file. If no SIM file is available or the SIM file contains no information on errors, no errors and no environment data is returned.

Example:

```
[P0110]
    Error entry. (P code)

EESActive=1
    Default=0.
EESStored=1
    Default=0.
EESMIL=0
    Default=0.
;All entries with a semicolon are comments
    Comments can be used for structuring and explaining
ENV_Drehzahl =0.0
    Environment data-qualifier and value.
ENV_MotorTemp=-40.0
    Environment data-qualifier and value.
...
[_E_P1605]
    Next error entry...
...
```

## Variant Coding

The simulation uses the "offline varcoding" provided by the CAESAR API. Coding services and fragments are shown. Simulated reading and coding of fragment values is only possible within the framework of the CAESAR API. An appropriate parameterization must be available. In addition, CAESAR hardware with the corresponding access level must be connected.

## Flashing

Flash processes cannot be simulated.

### 2.4.4.24 Clamp 15 Handling

Since the omission of clamp 15 as pin in the OBD socket, as a replacement the state of the ignition switch is available as signal on the diagnosis CAN. CAESAR is able to evaluate this signal, must however be configured accordingly. Refer to CAESAR documentation concerning clamp 15 omission for further information.

In Vediamo this configuration can be already made system related in the system description, but a system description-independent standard attitude is also available. There is a INI file, which contains the configuration of the clamp-15-parameters dependent on the model (file name: „Ignition.ini “). The Ini file is located in the program directory.

It contains model-dependent the following data:

- Recognition "automatic", over "hardware pin" or "from CAN"

The selectable entries in the file read in such a way: AUTOMATIC, HARDWARE, CAN

- File name clamp 15 CBF
- ECU name for clamp 15 recognition
- Variant (model-dependent)
- Ignition read service
- CAN wake up service

File contents example:

[Models]

NumberOfModels=3

Model1=BR211

Model2=BR220

Model3=BR203

[DEFAULT]

IgnitionMeasurement=HARDWARE

CBFFilename=

ECU=

Variant=

ReadIgnitionService=

CanWakeupService=

[BR211]

IgnitionMeasurement=AUTOMATIC

CBFFilename=Clamp15.cbf

ECU=CLAMP15

Variant=1\_Bit\_Auswertung

ReadIgnitionService=

CanWakeupService=

[BR220]

IgnitionMeasurement=CAN

CBFFilename=Clamp15.cbf

ECU=CLAMP15

Variant=2\_Bit\_Auswertung

ReadIgnitionService=  
CanWakeupService=

[BR203]  
IgnitionMeasurement=HARDWARE  
CBFFilename=  
ECU=  
Variant=  
ReadIgnitionService=  
CanWakeupService=

The "ReadIgnitionService" and "CanWakeupService" entries are for information only - they are not used further.

In Ecoute, under the menu option "extras" there is a menu option "clamp 15 handling" for adjusting the behavior of the ignition recognition depending to the related model. If the menu

option is selected, a dialogue will be opened, in which the model can be selected. With click on "OK" the configuration in accordance with selected model is set. The last set configuration/model is stored into the Vediamo.ini and loaded and set with the next Vediamo start again.

Key in Vediamo.ini:

[Servers]

...

IgnitionMeasurement=BR203

If a VSB contains a valid configuration (all information available), then the configuration of the VSB will overwrite the currently selected configuration. Renewed setting of the configuration by Ecoute/other Clients/test sequences is possible. If the model is set per dialog in Ecoute, and the specified clamp 15 CBF does not exist (= is not in the list of Caesar's currently used .CBFs), a warning is provided; the configuration is however nevertheless set. Other Clients, as well as test sequences receive this information as return value when setting the configuration.

## 2.4.5 The Ecoute Menus

### System Menu

#### Select

Use this item to load the system description you want to work with. An already opened system is closed in the process.

#### Close

A system can be closed using the menu time *System / Close*. All windows which display ECU relevant data (system, error, measurement, trace) are closed in the process.

### **Open OBD2**

Opens a OBD2 connection. With this protocol diagnostic data can be read without the need of special ECU dependant files. A special OBD2 window is opened.

### **Close OBD2**

Closes the OBD2 connection and the OBD2 window.

### **Open session**

A session file can be opened using the menu entry *Open session*. This is done with the help of a file selection dialog which displays all files ending with *.vsc*. Select the session to be opened with the help of this dialog. The active session file is displayed in the title line.

### **Save session**

Use the menu entry *Save session* to save the changes in a currently open Ecoute session to the active session file.

Ecoute can be started with a specific session. The complete path of the session file must be passed to the program via command line parameter. Optional the program can be started automatically with the last stored session. Therefor the option *load last session* must be activated in the [options dialog](#) .

### **Save session as**

The menu entry *Save session as* allows an Ecoute session to be stored under a different name. This is done using a file storage dialog. The filename and storage location can be specified.

### **Close session**

The menu entry *Close session* closes a session file. After this action, the selected system and all open windows are closed. The main window is restored in accordance to the INI file.

### **List of last used files (MRU list)**

The names of the last four opened systems or session files are displayed in the *System* menu. These files can be opened simply by clicking on them.

## **Exit Program**

This item closes Ecout. If no further Vediamo clients are active, the DiagServer is also automatically ended.

## **ECU Menu**

### **Init contact**

Init contact starts the process to establish contact with the ECU. This can also be achieved using the F3 function key, double-clicking on the ECU in the selection window, or clicking on the ECU's button in the status line.

### **Exit contact**

Ends communication with the ECU. Alternatively: F4 key, double-click in the selection window, ECU's button in the status line.

### **Read ID block**

This command (or using ALT-I) reads the ID data from the ECU. In case of multiple ECUs, the ECU must be selected first. Contact has to be established with the ECU in order to read the ID block.

## **Communication Parameters**

Experienced users can change communication parameters using this menu entry.

## **Manual Command Input**

This command activates a window for communicating with the ECU on a low level (CAESAR-API-I). Byte level messages can be transmitted here replies can be analyzed.

## **Properties**

This opens a window with the ECU properties.

## **CAN bus simulation**

Simulating CAN messages.

## **Load CAN bus simulation**

Loading pre-configured CAN messages for simulation.

## Channel Monitoring

A [trace window](#) for any existing CAESAR connection can be displayed with this function.

## Error Menu

### New error window

Opens a new [error window](#), even if a window is open already. The window is filled with the current error information.

### Read errors

Opens a new or updates an already open [error window](#).

### Read permanent errors

Opens a new or updates an already open [error window](#) containing permanent errors.

### Clear errors

Clears the error information in the ECU. If changes are visible during the next error reading depends on the respective ECU.

### Read, save and clear

To simplify frequently repeated actions, this function can perform three different functions:

- Read errors
- Log read errors in a snapshot file
- Delete the errors in the ECU

### Quick test

Performs a [quick test](#). The error memory of all ECUs with which communication is currently established (e.g. for a complete vehicle) is read/or cleared.

### Read errors by Status

Reads all supported error codes (KW2000: "REQUEST SUPPORTED 2 BYTE HEX DTC AND STATUS (\$03)").

Protocol specific. Only meaningful for KW2000 and UDS.

## **Service Group Menu**

### **New service group**

Opens a new [service group window](#). It can be filled with services and stored as a service group file (VSG)

### **Open service group**

Opens a stored service group.

### **Open/Analyze Recorded Series**

Loads a previously recorded series of measurements (VSR file).

## **Coding Menu**

### **Variant coding**

Opens a window for reading and setting the ECU variant. If entered correctly in the system description file (VSB), the services which bring the ECU into the correct mode (preconditions) are executed first. If this does not happen, the services can also be started manually from the selection window.

### **Flash**

Opens the window for flashing a specific firmware into the ECU.

## **Services Menu**

### **Execute Routine**

You can select and start a service from one of several groups or a Java routine. These items are useful if you have to work without a system window.

This menu is also traditionally used to insert customized menu entries with the help of standard objects.

## **Extras Menu**

### **Find**

The function "find" can also be executed using the keyboard shortcut CTRL+F. For the user, there are various ways to search for a text in vediamo related data:

- In the current system

- In the Ecoute windows
- In files

"Searching in the current system" or "in the Ecoute windows" and "searching files" is made in two different Windows, between the user can change using a tab button.

## **Searching in the current system**

### **Input field for text to find:**

The input field is identical with the field from searching in files. Entered keywords appear in both windows. The input field offers a history containing the last used keywords. A keyword can be pasted from the clipboard.

### **Search in current window**

If "Search in current window" is selected, the next found result in the currently active window will be selected and made visible. The related window will be searched up from the beginning or up from the currently selected position.

### **Search in tree:**

If "search in the system tree" is selected, the entire currently loaded system is searched at once, and hits are added to the result list. The result list contains the qualifiers/names of the services containing the wanted keyword. If the user clicks on a service in the result list, the related entry in the system tree is selected. Services in the result list can be added to a measurement window using drag and drop.

## **Searching in files**

### **Input field for text to find:**

The input field is identical with the field from searching in the current system. Entered keywords appear in both windows. The input field offers a history containing the last used keywords. A keyword can be pasted from the clipboard.

### **Input field for directory:**

Specifies the directory that contains the files to be searched through. The default is the current system path.

### **Search in files:**

Files with the selected extensions will be parsed.

### **Result list:**

Files containing the specified keyword will be added to the result list. The list also shows the file's pathname. If the user clicks on a file entry in the result list, the related file is opened and displayed.

## **Options**

Opens the window for editing the Ecoute and server options relevant for Ecoute.



## **Flashdata Administration**

Opens a dialog for handling flash files (see flash dialog).

## **Font**

The font to be used in the measurement, actuator and error windows can be selected using this menu entry.

## **Clear status window**

This command clears all entries in the status window (output window).

## **Server Reset**

The DiagServer is re-initialized. This is necessary if work should continue with modified settings or an updated parameterization. The current Ecoute state (open system, opened windows) is cached and restored after the reset.

In some situations (modifying certain settings which require a reset), the user is asked if the server should be rebooted.

## **Clamp 15 Handling**

Opens the window to select a model used for clamp 15 handling.

## **Show trace**

Opens the trace window.

## **Macro**

A sequence of Ecoute command can be stored here or a stored sequence can be played back.

## **Config toolbar**

This feature allows the toolbar to be adapted to individual needs. The configuration is made in an own window:

On the left side, the as functions available toolbar buttons are listed, on the right side the currently assigned toolbar buttons.

Using the "add" button, an entry selected on the left side can be added to the toolbar and will then be listed on the right side, too.

Similarly, using the "Remove" button an entry selected on the right side can be removed from the toolbar.

The "Up" and "Down" buttons refer to the current contents of the toolbar and also on the list on the right side. With their help, the currently selected entry in the list is moved up or down, thus determining the the order in which the buttons appear on the toolbar appear.

With the entry "Separator" anywhere a dividing line can be inserted, e.g. to make individual functional groups to be drawn between them.

The toolbar can be docked at the top/bottom left/right side of the main window by dragging it to the desired location.

## **Window Menu**

### **Cascade, Tile Horizontally, Tile Vertically**

These commands arrange the open windows.

### **Status**

Opens or closes the status window (output window).

### **System window**

Opens or closes the selection window (system window).

### **Routine manager**

Opens or closes the window for controlling Java routines.

### **Macro Window**

Opens or closes the window for monitoring a played-back macro.

### **Update current window**

The content of the active error, actuator or measurement window can be updated using this menu entry or the F5 function key. The key combination CTRL+F5 updates the contents of all windows at once.

### **Show toolbar**

Shows or hides the toolbar.

## "?" Menu

### Help topics

Opens the online help.

### About Ecoute

Displays Ecoute's "business card".

### Keyboard Operation

Shows that part of the online help with all keyboard commands.

### Additional information

For the ECOUTE user it is possible to provide own additional support informations in .chm- and .PDF-files. Any time a new system is loaded, the following files are displayed:

- All .chm- and .PDF files located in the directory ..\Data\VediamoDaten\ are listed below the "?" menu.
- All .chm- and .PDF files located in the directory ..\Data\VediamoDaten\ [current system] are listed below the "?" menu.
- All these files are additionally located in a new folder in the system tree.

When selecting a file in the menu or by double clicking it's entry in the system tree, the file will be opened and its contents showed...

## 2.4.6 Keyboard Operation

In the age of windows and the mouse, it is no longer taken for granted that a program can be operated from the keyboard. But it is a significant convenience, e.g., during test drives, that all important Ecoute functions can also be executed without a mouse. Exceptions are the adjustment of window sizes and positions, and other functions which are rarely required and then only in the preparation and configuration phase of test drives. Such actions as reading errors, displaying a group of selected measurements, and much more, can be started by only a few keystrokes. The specified hotkeys are based on the early DOS diagnostic programs MODI and ISOSCAN.

The following tables contain a complete list of all Ecoute keyboard commands.

Key	Action
F1	Online help
F3	Establish contact to the ECU

F4	End contact to the ECU
F5	Update active window
F6	Open a new error window, if no error window has been opened yet. If an error window exists, update the contents.
F7	Read all measurements
F8	Read all actuators
F9	Execute last macro
F10	Start quick test
F11	Open OBD2 window
F12	Manual command input
Alt+A	Select system
Alt+B	Delete status message
Alt+D	Activate service menu
Alt+E	Read once (e.g., measurements)
Alt+F	Flash ECU
Alt+G	Show properties of current service in tree control
Alt+I	Read ID block
Alt+K	Activate coding menu
Alt+J	ECU properties
Alt+L	Clear all errors
Alt+O	Open session
Alt+P	Save snapshot (e.g., error window)
Alt +Q	Display trace
Alt+S	Activate system menu
Alt+T	Activate ECU menu
Alt+V	Execute variant coding
Alt+Z	Close system
Alt+F	Activate error menu
Alt+M	Activate measurement menu
Alt+R	Activate actuator menu
Alt+N	Activate windows menu
Alt-F4	Exit Ecoute
Alt+F6	Read permanent errors
Alt+F7	Open service group

Alt+F8	Open actuator group
Alt+F9	Open macro selection list
Ctrl+F	Find function
Ctrl+F5	Update all windows
Ctrl+F6	Read, save and clear errors
Ctrl+F4	Close active service group, actuator group, error window
Ctrl+F7	Measurements: New group
Ctrl+F8	Actuators: New group
Ctrl+F9	Record macro
Ctrl+F10	Set standard system path
Ctrl+TAB	Toggle between different windows
Schift+F6	Open a new error window
ESC	Close active window
TAB	Jump within the active window

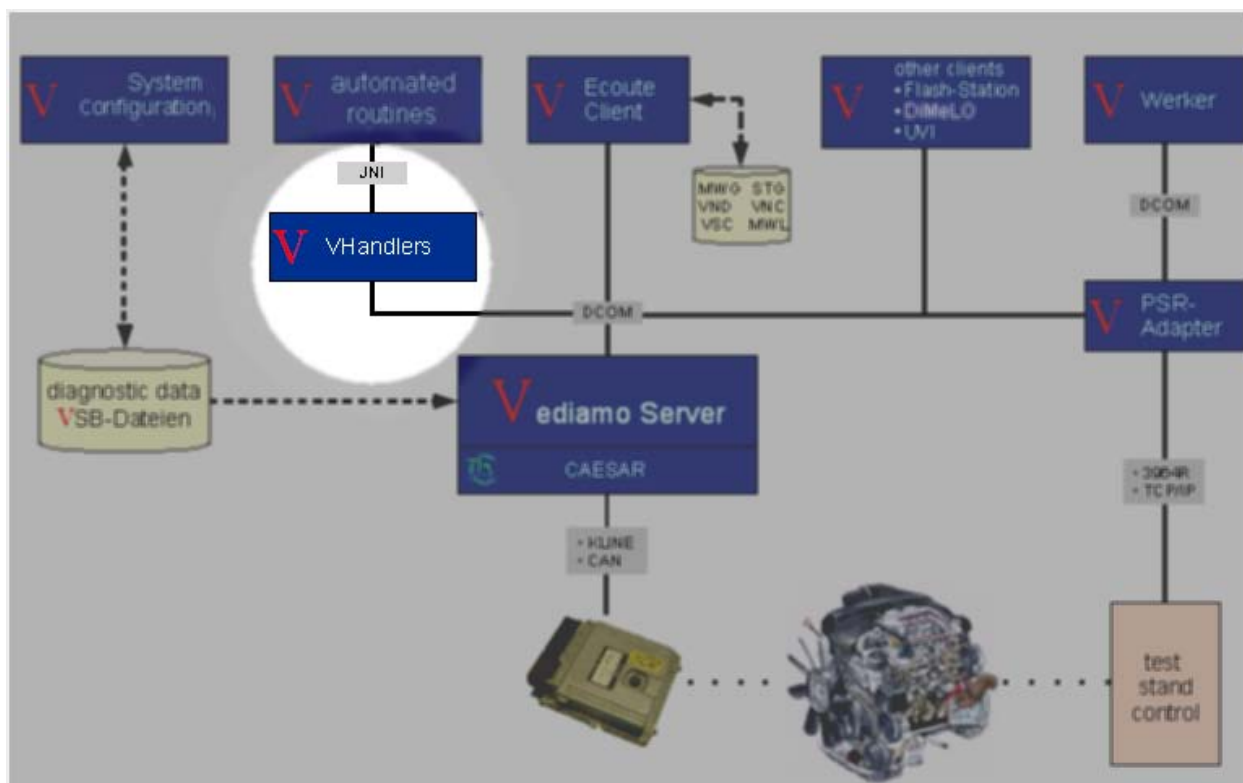
### Selection Window (System Window)

Key	Action
Right arrow	Expand entry
Left arrow	Collapse entry
Arrow up	Move up in selection window
Arrow down	Move down in selection window
Enter	Start action: ECU: Establish or end contact Service, Java routine, etc.: Execute Folder: Expand / collapse

### Graphic Measurement Window

Key	Action
Enter	Start / Stop recording values
Blank (Space key)	Start / Pause cyclic reading
Del	Erase recorded values from memory
Cursor keys	Select value point to be displayed (if any measurement values have been read)

## 2.5 Java Handler Functions



In Vediamo, it is possible to execute Java routines during a diagnostic session. Java routines are Java programs which the user can write/edit himself and from within Vediamo diagnostic server functions can be called. Therefore Vediamo provides an interface realized by a number of Java classes.

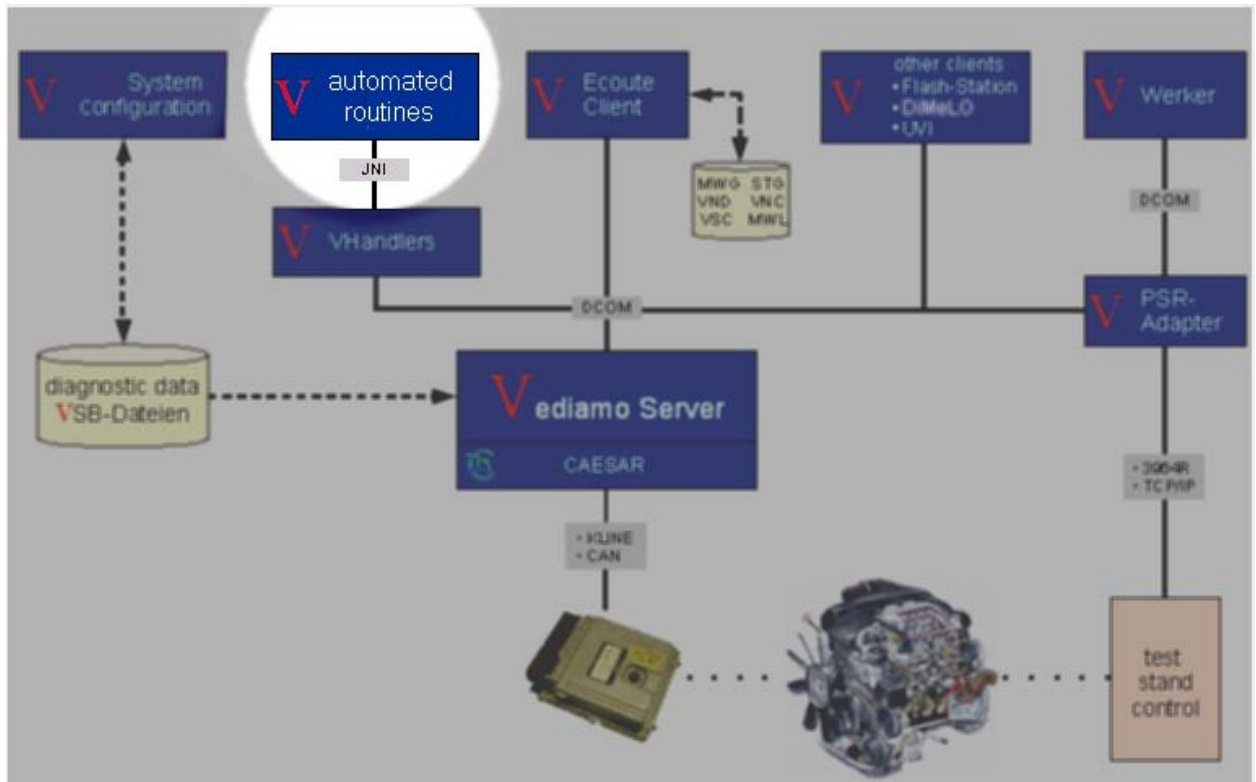
### 2.5.1 Vediamo Java Interface

The [VHandlers Java Interface Documentation \(click for more...\)](#) describes all provided Java classes for accessing the functionality of the DiagServer. These classes can be used in own applications, developed in Java.

Depending on the Java interpreter used, there can be conflicts in connection with Vediamo's own Java handler classes: Some Java development environments/interpreters require the Vediamo's own Java handler classes in a named package, e.g., "JVHandlers". The Java handler classes in Vediamo are therefore provided in two versions, once in an unnamed package, and additionally in a package named "JVHandlers". The function of both versions is absolutely identical. The `vHandlers.dll` module supports the unnamed package as well as the JVHandlers package at the same time. The handler classes in the

"unnamed" package and the additional handler classes in the "JVHandlers" package are installed in separate archives in the ..\Java directory.

## 2.6 Java Programs (Java Routines)



### 2.6.1 Introduction

A programming capability is required to increase the effectiveness of frequently repeated functions and actions. In earlier diagnostic systems (MODI, Isoscan) functional chains programmed in an own Pascal dialect proved reliable. Many other programs have the capability to record and play back macros, which resemble an own programming language in extreme cases (e.g., Visual Basic in MS office applications).

In Vediamo, it is possible to program customized Java routines. The advantages of Java are obvious:

- Java is a standardized and widespread programming language with sufficient documentation in all languages. It is not necessary to learn a Vediamo-specific language.
- Java Runtime is available at no cost for all major operating systems. No own interpreter has to be developed for Vediamo nor are extensive testing efforts

necessary, since a Java interpreter from Sun Microsystems already implemented by millions is integrated directly in Vediamo.

- Java can not only address the Vediamo system functions, but all available libraries as well. It is therefore possible to program stand-alone applications of any given complexity, with windows, menus, graphics, etc., and to profit from the simply accessed diagnostic functions of the Vediamo DiagServer.

## 2.6.2 Executing a Java routine from Ecoute or Another Client

Java routines were originally intended as an extension of the ECU parameterization, as a type of additional DiagJobs. This allows, e.g., a sequence of specific diagnostic services to be executed each time contact is established, or a specific action (actuator activation, variant coding, ...) to be executed when starting up. For this reason, Java routines can be added to the system description using the [system configuration](#). These are visible and accessible in the Ecoute selection window after the system is loaded. The routine sequences are usually rather simple; they do not require their own interface and can assume that a certain ECU is available or already contacted when they start.

These routines can be started in [Ecoute](#) by selecting and acknowledging (double-click with mouse, <RETURN> on the keyboard) the entry in the selection window. Any outputs from the program are displayed in the Ecoute output window.

If the routines are declared as initialization routines (initialization sequences), they are executed at the correct instance in time without any assistance by the user.

Routines listed in the system description can be started in the [PSR client](#) using the [command](#) "600". To communicate with the test bench controller, the [Vediamo-Java interface](#) has a class JVPSR which can send messages to the PSR.

## 2.6.3 Java Program as Standalone Client

The power of Java and available libraries allows complete applications to be generated. These applications do not need to be (and usually aren't) linked to a specific ECU. They can execute, select systems, contact ECUs, receive and process data and much more independently of all other Vediamo clients.

Such Java clients are usually generated as JAR files. They can be started in the following manner:

### General:

```
javaw -classpath <JavaKlassenPfad_1>;<JavaKlassenPfad_2>;...; <routine name>
```

### Example:

```
"C:\Program Files\Vediamo\JRE\bin\javaw" -classpath  
"C:\ProgramData\Vediamo\VediamoDaten\AllgemeineScripte";"C:\Programme\V  
ediamo\Java"; Classname.jar
```



## 2.6.4 Example: Program, Compile and Execute a Simple Routine

Please use the [example file](#) from the attachment for your first programming attempts with the Vediamo-Java interface. You need the free Java Development kit (the current version is J2SE 5.0) from [Sun Microsystems](#) to write and compile Java programs. Alternatively, you can use comprehensive development environments such as Borland's JBuilder. This example is oriented primarily towards Java beginners, and it is assumed that J2SE or an earlier version is installed.

### Here is how you arrive at your first Java client for Vediamo:

- Store the contents of the example file as pure text in a file named `Example.java`. Make certain that use of uppercase and lowercase letters is consistent: The name of the Java file must be identical to the name of the class it contains ("Example" in this case).
- In the definition section, change the qualifiers (IDs) of the ECU and services that you wish to communicate with.  
Use Ecoute to make contact with your ECU first and note the qualifiers of the ECU and the services you want to address. The display in Ecoute (*Extras / Options / General / Display*) must be set to **Qualifier** (and not **Name**).
- Compile the source file to binary code.  
The command line for this is:

```
javac -classpath <Path to the Vediamo handler classes>JVHandlersPackage.jar  
<Path to Java source code> <Source code Name>.java
```

In our example, if your file is in `C:\JavaTest\Quellen`:

```
javac -classpath "C:\Programme\Vediamo\Java\JVHandlersPackage.jar"  
C:\JavaTest\Quellen\Example.java
```

The system variable `PATH` must contain the path to the Java programs (to the Java compiler `javac.exe` in this case). Alternatively, enter the complete path name in the command line:

```
C:\Program Files\Java\jdk1.7.0_25\bin\javac -classpath  
"C:\Programme\Vediamo\Java\JVHandlersPackage.jar"  
C:\JavaTest\Quellen\Example.java
```

The compiled file has the same name and the extension `.class`: `Example.class`. For practical purposes, this file should be located in the subdirectory of the system for which the routine is valid. If the system file is named, e.g., `CR4test.VSB`, copy the `class` file to the directory `...\VediamoDaten\CR4test`.

- Tie the routine into the correct system (which contains the ECU to be addressed). Use the [system configuration](#) to do this. Open the system, find the item *routines* and click on it with the right mouse key. You can then find and select the *class* file that was just generated. You can also add a description if desired, or leave the description field empty.
- Start Ecoute
- Load the named system
- Find the new routine in the system window and execute it by double-clicking it.

You will see in the "routine manager" window that the routine is active. The window closes automatically when the routine has ended. The output from calling `client.Trace()` results in text output in the Ecoute status window. If multiple clients are active, the outputs occur in the client which started the routine

- Try other functions as well

The description of the [Vediamo-Java interface](#) contains a complete list of all classes and their methods (class functions). The Sun Microsystems [Java homepage](#) contains a complete description of the Java programming language.

## 2.6.5 Particulars

### Aborting Java Routines

When a Java routine program is started from Ecoute, either directly or using a standard object, this Java routine can be aborted using the *Active routines* dialog. The dialog is displayed when at least one Java routine is executing and the option *routine Manager* is activated. All currently executing Java routines started by the Ecoute client are displayed in the dialog. The starting time of each Java routine is also displayed. To abort a Java routine, select the corresponding line in the dialog and then click on the button *Abort routine*. Initialization routines and Java routines which run as preconditions run in the foreground of the Ecoute client and cannot be aborted.

Important:

An executing Java routine can only be aborted by the Vediamo system if the routine regularly calls Vediamo handler functions. This fact should be taken into account already when writing a Java routine.

### Special features in the use of Java routines in .jar files:

Java routines can be run as a .class, as well as a .jar file. When running .jar files, a special feature must be noted: If communication is desired between the Java routine and the client that started the routine, e.g. via the `JVClient.Information()` function, the starting

client passes a unique identifier as the last command line parameter of the Java routine... Communication is only enabled if this command line parameter is passed to the function `JVUtil.SetScriptIdentifier()` (preferably immediately at the start of the routine in its `main()` method). Otherwise, the client can not be identified as the desired communication target. When you run a `.class` file, the VHandlers framework does this work automatically, when using a `.jar` file, your own Java program code must do it.

## 2.6.6 Configuration (INI Parameters)

The following values must be correctly entered in the configuration file [vediamo.ini](#) for Java routines to execute properly:

```
[ INTERPRETER ]
ClassPath - Paths and JAR files with the Vediamo-Java classes used
VHandlersLib - Path to the provided VHandlers.DLL
JavaInterpreter - Path to the Java interpreter
```

These entries are made by the Vediamo setup and only need to be modified by the user in exceptional cases.

In addition, the entry

```
UseJVHandlersPackage=0
```

controls whether you want to use packages in customized Java programs (e.g., this is obligatory in new versions of JBuilder).


## 2.7 BlackBox

### 2.7.1 Introduction

Even if they are not planned - unfortunately, crashes do happen. It is important to determine the cause and to avoid it in the future. As in air travel, this is done with the help of the BlackBox.

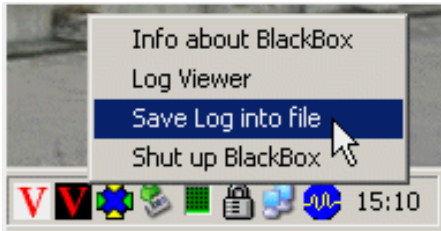
In this case it is a program which monitors the functions of the Vediamo modules (servers and clients), by receiving entries from the applications on their current program state. If a critical event occurs (server or client crash, loss of contact), the cached entries are stored in a log file.

### 2.7.2 Structure and Function

The BlackBox can be cativated or deactivated by INI parameter "Run". After installation, it is by default off. BlackBox runs unnoticed in the background. Its menus can be reached using the  icon in the taskbar. It has contact to all Vediamo applications (servers and clients) by way of a [DCOM](#) interface. These pass their log entries to the BlackBox, which

stores them in a ring buffer. Upon request from an application or the user (via taskbar menu), the last available entries (up to 5000 lines, although this number can be changed by INI parameter) are stored in a file.

### 2.7.3 BlackBox Functions

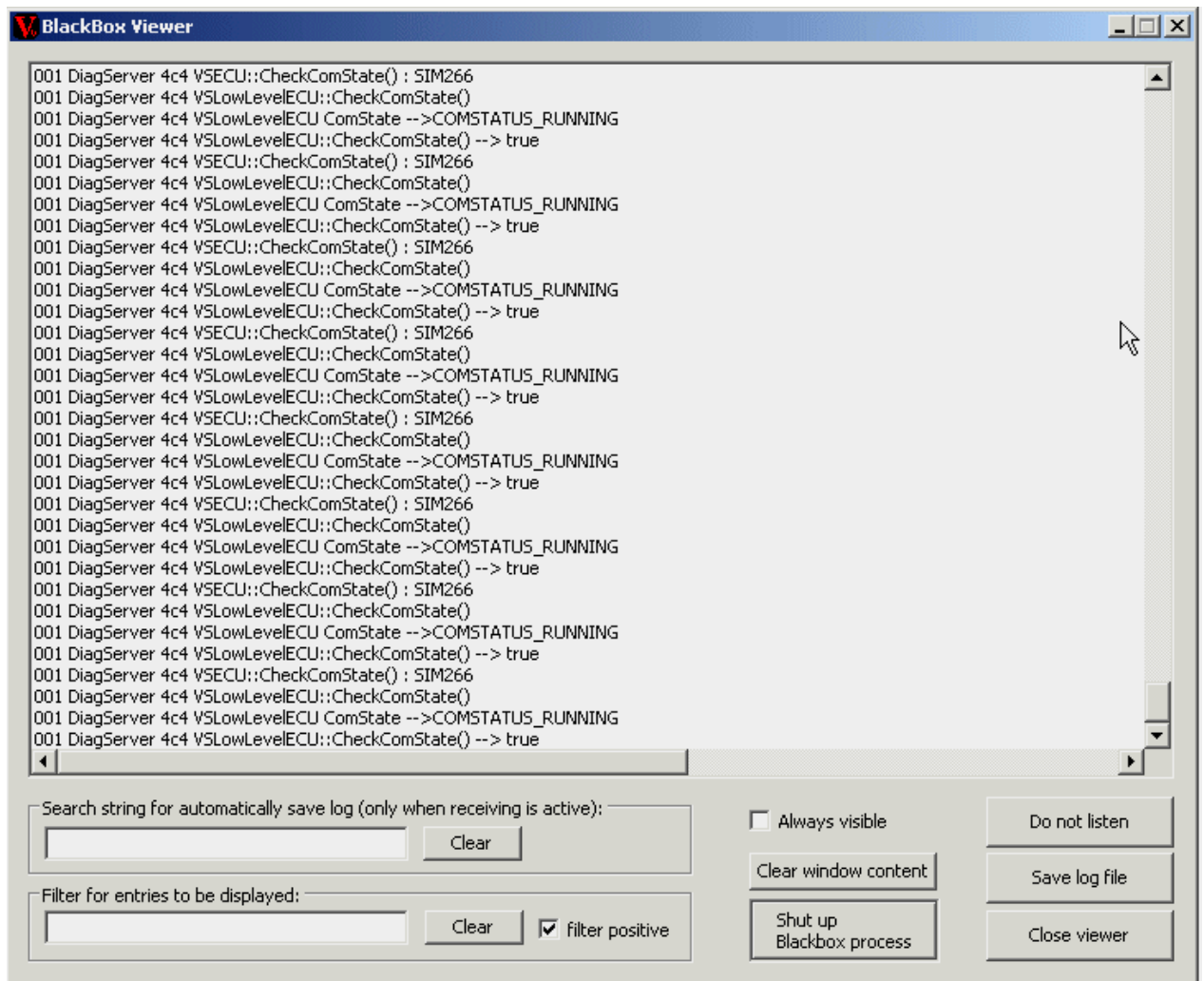


The following functions are accessible using the taskbar menu:

- *Info about BlackBox*. The info window displayed contains information on the program version and configuration file.
- *Log Viewer* This starts a program which continuously displays the log entries.
- *Save Log into File*. If an irregular event occurs, the user can save the logs in order to make them available to the developers for analysis.
- *Shut up BlackBox*. In exceptional cases, it could happen that BlackBox does not end by itself (e.g., after an application has crashed). This menu entry ends BlackBox and returns to a defined initial state.

### 2.7.4 BlackBoxViewer: Log Display at Runtime

The menu entry "Log Window" opens the following display:



Most of this window's control elements are self-explanatory. The search string masks deserve special mention.

- Search string for automatically saving log. Enter a text string here. As soon as this string is found in a log entry, all available entries up to that point are stored in the log file.

Please note:

The search distinguishes between uppercase and lowercase.

- Filter for entries to be displayed: Here you can limit the display to a choice of the information. If the option *filter positive* is active, only those entries which contain the search string are shown. The filter affects only the display in the window, the content of the potentially saved logfile is not impacted.

Note:

Displaying during runtime noticeably burdens the CPU. Under certain circumstances, this can result in a different sequence being displayed than if the BlackBoxViewer was closed. For time-critical sequences therefore, it is recommended to repeat the situation without the log window and to save the log using the taskbar icon.

## 2.7.5 Linking to Other Applications

If you develop customized programs which run together with Vediamo, BlackBox can be applied to them as well. To receive the description of the [DCOM](#) interface, please contact the [Vediamo team](#).

## 2.7.6 Configuration (INI Parameters)

This program has three parameters in `vediamo.ini`.

- Run - BlackBox is active (1) or inactive (0)
- LineNum - the count of entries in the log file.
- It specifies the path and the string the logfile names start with. The full name is constructed from this entry, supplemented by date and time.

Example:

```
[BLACKBOX]
```

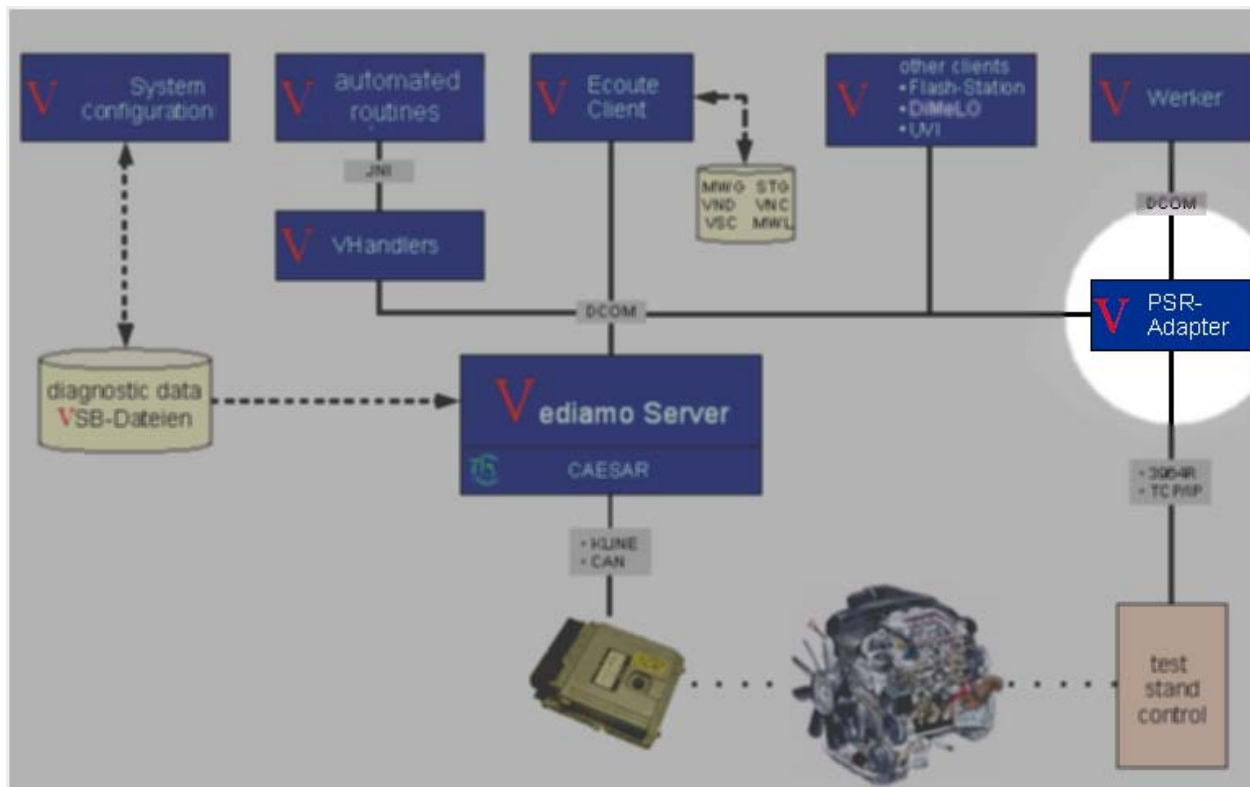
```
Run=1
```

```
LineNum=5000
```

```
LOGFILE=D:\Data\Log\BlackBox
```

In this case a file can be named, e.g., **BlackBox\_03-11-2005\_17-23.log**. It contains up to 5000 lines of log info.

## 2.8 PSR Adapter




### 2.8.1 Introduction

In the following text the abbreviation PSR means **test bench controller** (= Prüfstandrechner, PSR).

The PSR client is a utility program which gives the test bench controller access to the DiagServers diagnostic functions. Its purpose is the adaptation of the complex Vediamo interface to the communication with the test bench controller.

The PSR client is started by activating the *Connect to Server* command in the worker client. The PSR client opens the interface to the PSR and activates the DiagServer. When the worker client is ended (if more than one was active, then when the last worker client is ended) the PSR client is also ended and thus also the DiagServer (if no other clients such as Ecoute or Java programs are active).

If no interface is needed, the PSR client can also be started directly by the test bench software using DCOM calls.

In this case, the PSR adapter will be stopped by closing the test bench software. Additionally to this standard way, PSR client can be also ended by a command (702) from the PSR or from the taskbar menu (right mouse key click on the  task icon).

## 2.8.2 Communication between Vediamo and PSR

The communication between PSR and the Vediamo DiagServer occurs in individual messages through the PSR client. These message blocks can be transmitted in two ways:

- serial with protocol 3964R over the COMx interface
- over a network contact with HDLC over TCP/IP protocol

The contents of the message is identical in both cases.

The test bench controller (PSR) is the master and the PSR client is the slave. This means that messages from the PSR have the meaning of commands which start, stop certain actions or execute them once. The PSR client can receive commands at any time and in variable order. Whether the commands have the desired effect depends on the system state.

The PSR receives various information from the PSR client as a result of performed actions and various events. Some messages from the PSR client can be attributed exactly to one command (e.g., message 601 after command 600), others can occur in changing quantities at various times (e.g., 301 during cyclic measurement reading). There are also messages which are sent to the PSR without being requested. These are messages on system errors as well as on changes in the system state (e.g., lost contact to the ECU).

The following rule applies to all messages:

Every message which should be sent by the PSR adapter is sent until reception by the PSR has been acknowledged.

Exception: During cyclic measurement reading, unsent measurements are cleared from the buffer, so that only the newest values are sent.

### Behavior when Communication is Interrupted

No data is lost if communication with the PSR is interrupted. The PSR client initializes the interface and sends all blocks contained in the buffer as soon as possible once contact is established.

This leads to the PSR first receiving messages from the PSR client after contact is established, e.g., in the case that the PSR was restarted while Vediamo is still in the test run and has messages in the buffer.

For test bench systems where this behavior leads to problems (e.g., older test benches from AT&T), the INI entry `COMTIMEOUT` can be set so that the test run is ended and the



transmission buffer cleared when communication has a longer interrupt. A newly started PSR then finds the PSR client in a defined state.

### Message Block Structure

Length:: 2 Bytes (HI, LO)	Time Stamp: 2 Bytes (HI, LO)	Service Number: 2 Bytes (HI, LO)	Optional Data: 0 to 1018 Bytes Word, String(s) or Combination
Length of block, including length, time stamp and service number	Time elapsed since program start in 1/10 sec	Unique number that starts a certain action in the DiagServer	Dependent on the service number. Words are transmitted in the order (HI, LO). Strings are zero-terminated ASCII strings

Each block from the PSR has the meaning of a command to the Diag Server. Replies from the DiagServer (acknowledgements, data, information) are sent in individual blocks. The transmission attempt is repeated until the recipient (the PSR) acknowledges reception in accordance with protocol. Cyclically transmitted measurements are the exception - they are sent only once.

The command blocks are transmitted to and from the PSR over 3964R / ASYNC or HDLC / TCP/IP .

The following parameters need to be set in the PSR:

- 3964R / ASYNC:
  - Prio=1 (i.e, in case of a simultaneous transmission, the PSR adapter switches to receive and gives the PSR priority)
  - COM interface: 19200,8,e,1
  - Byte delay = 220 ms (time between bytes in the block)
  - Ack delay = 550 ms (delay for an acknowledgement)
  - Retries = 6 (the protocol makes a maximum 6 attempts to transmit each block)
- HDLC / TCP/IP:
  - IP Port = 2049 (other values over 1024 are possible)
  - PSR Client = IP server - the PSR client is the first to open the socket
  - Control computer = IP client - the PSR establishes contact with the running PSR client first.

Note:

The values *Byte Delay* and *Ack Delay* may be exceeded but not under-run on the receiver side. This means that the receiver should wait at least the specified time for the byte. The sender must transmit in less than these times. Increasing the value only increases the assurance of reception for a slow receiver, it does not

affect the runtime since under normal circumstances the bytes or ACK are sent with substantially shorter delay times.

### 2.8.3 The Functions of the PSR Clients

The PSR client provides most of Vediamo's functions which can be relevant in the test facility. Everything necessary beyond that is programmed using Java routines and integrated into the test run.

The individual functions are:

- Load and initialize data from the selected system.
- Establish and end contact with ECUs.
- Read errors with and without environment data. Single or cyclical until the command ends.
- Read all or selected measurements. Single or cyclical.  
Measurement reading is time-optimized. Numbers, and not IDs, are used for referencing to reduce the amount of data. This is especially important for the communication over 3964R, since the data throughput can be lower than between Vediamo and the ECU.
- Execute services including actuators and adjustments (referencing using IDs). Parameters are transmitted (if necessary), result is returned. Also applies to measurements.
- Flashing the ECU.
- Sending and receiving API-1 messages to the ECU
- Execute Java Routines.  
The [Java interface](#) has a [Class JVPSR](#) with functions to send messages to the test bench directly from the Java code.
- Read various information from Vediamo:
  - CAESAR version
  - Identblock
  - Defined measurements (that are filtered positively in the system configuration)
  - Lists of services
  - Details on services
  - ...
- Restart the DiagServer.  
This is necessary, e.g. when CAESAR data is updated and has to be read in again by the server. The PSR client remains active and contact to the PSR remains intact.

Variant coding is not (yet) realizable directly with PSR commands but only with Java routines.

The function principle of the test run is event control. This means that certain events (e.g., a command from the PSR, loss of contact with the ECU) lead to actions which can

be longer or shorter and can change the state of the complete system. The PSR is notified of every action in the PSR client. The sequence of the messages to the PSR can deviate from the order the events (or commands) occurred, because the tasks are performed partial in different parallel tasks.

It has to be considered in the PSR program that neither the order nor the time between the messages needs to remain constant. Messages which relate to the same action are the exception, e.g., the start of a Java routine and the end of a routine can never occur in reversed order.

The following shows an example of a test run sequence.

Sender	Code	Parameter	Explanation
PSR	3	"CR4_1"	Load data of motor identifies in <a href="#">engine table</a> as "CR4_1".
Vediamo	5	"CR4psr"	Data of system "CR4psr" is loaded. The engine table specifies which system description belongs to which motor.
PSR	100		Establish contact to ECU.
Vediamo	101	"CR4"	Contact to ECU "CR4" is established. The system must not be named exactly like the ECU.
Vediamo	112		Initialization phase ended. It is possible that other messages precede this one, e.g., if an initialization routine is executed automatically.
Vediamo	201	"P0115","Temperature sensor defective"	The ECU reports an error. This is automatically read cyclically and transmitted by Vediamo after contact is established. If no errors are present, only the message "End of error list" is displayed.
Vediamo	201	"," (2 empty strings)	End of error list.
Vediamo	201	"P0115", "Temperature sensor defective"	Next error reading cycle. The errors are read cyclically until this is ended by the PSR.
Vediamo	201	","	
PSR	204		Stop reading errors.
PSR	306	2,7	Read measurements 2 and 7 once
Vediamo	301	2,"Off"	The result of measurement 2 is "Off".

Vediamo	301	7,"3250"	The result of measurement 7 is 3250
PSR	500	"ACT_Drosselklappe","75"	Set actuator "QACT_Drosselklappe" to target value 75.
Vediamo	502	3,"ACT_Drosselklappe"	Error: Vediamo cannot activate the actuator. (e.g. access privilege not turned on)
PSR	550	"FN_Zugriffsberechtigung"	Execute service
Vediamo	551	"FN_Zugriffsberechtigung", "Access privilege granted"	Service executed, result text is provided.
PSR	500	"ACT_Drosselklappe","75"	Request actuator adjustment one more time.
Vediamo	501	"ACT_Drosselklappe","75.2"	Actuator adjustment successful, new value is 75.2 (value can vary slightly because a float is converted to byte).
PSR	700		End test
Vediamo	102	"CR4"	Contact to ECU closed.
Vediamo	701		Test run is complete.

To program your own test run, open the [complete list](#) of all messages which can be transmitted between the PSR and Vediamo.

## 2.8.4 Configuration

Like other Vediamo programs, the PSR client also takes its settings from the `Vediamo.ini` file. As with other Vediamo programs, the INI file is searched for in the same directory the program is stored.

Usually, however, all Vediamo programs including the respective DLLs and INI files are installed in the same directory, i.e., there is a common `Vediamo.ini` for all programs.

PSR relevant entries are made in the `[PSR]` section.

Key word	Explanation
COMPORT	Connection used for communication between PSR and PSR adapter. 0 = no communication with PSR. Program inactive. 1 or 2 = 3964R protocol over COM1 or COM2. For higher values, an I/O card with additional ports must be installed. 2049 = HDLC/TCP Communication over IP port 2049 (other values over 1024 possible)
COMTIMEOUT	Time in ms. If the communication with the PSR is interrupted for a longer period of

	time, a test run could be ended (contact with ECU discontinued) and all data deleted from the transmission buffer. 0 means no timeout. The test run is continued once contact is reestablished, no messages are deleted. Default: 0
ENGINETABLE	Engine table: File which defines the motors (designates motor IDs - system description + initialization routine + optional routine parameters). This file is absolutely mandatory.
LOGFILE	File in which the data exchange between PSR and PSR client is logged on byte level. Should only be activated when searching for errors, under normal conditions this entry is empty.

Example:

COMPORT = 1

LOGFILE = "C:\Log\LUCA.LOG"

ENGINETABLE = C:\Vediamo\Systembeschreibungen\Engine.vet

Note:

In most cases, filenames can be entered in the INI file without quotes. Since the LOGFILE entry is used by a program module from a third-party source, it is necessary that the pathname, including the letter of the drive, be listed in quotes.

## 2.8.5 The Engine Table

The engine table specifies which engine is diagnosed with which dataset using which ECU. The file structure is as follows:

- Lines which begin with ";" are comments and are ignored
- Every (other) line starts with the engine ID, followed by the system ID (VSB name), optionally followed by the name of the initialization routine, possibly with parameters. Blanks or TABs are used as separators.
- A blank line or end of file means end of table

For systems which include more than one ECU, the ECU to be used for testing can be selected. This is done by specifying the ECU ID after the system ID, separated by a "/" without blanks or TABs. If no ECU is specified, the first one found is considered selected.

If the system contains multiple ECUs of the same type, their IDs are automatically extended by a number in parentheses (e.g., ME20(1), ME20(2) etc.).

Example:

;This is a comment

;

Engine is named Aklasse, has an Sim4, we use the sim4serie.vsb. A Java routine is executed at the start of the test run

**Aklasse sim4serie Init.class**

```

;
    Gas engine with ME2.0, without Java routine
Benziner me20
;
    For diesel engines we have a system cr2cr3.vsb, in which a CR2 and a CR3 are
    defined. Old and new diesel engines use the same system, but different ECUs and
    are initialized by the Java routine, but with different parameters:
DieselAlt cr2cr3/CR2 InitDiesel.class alt
DieselNeu cr2cr3/CR3 InitDiesel.class neu
;
    a system with two of the same ECUs. This makes sense, e.g., when two engines
    use the same ECU, but different errors/measurement filters are applied
DieselMitSLP Diesel/CR2(1) Varcod.class slp

DieselOhne Diesel/CR2(2) Varcod.class ohne_slp

```

### **When does a system with multiple ECUs make sense?**

The case that simultaneous contact to multiple ECUs is necessary to test an engine is very rare. It can still make sense to use a system file with multiple ECUs. Especially if the only a few different ECU types can be used alternating at the test bench. A system with multiple ECUs requires more RAM at runtime but the loaded ECU data stays in memory permanently. This has the following advantages:

- The starting phase of the test run (other than the first) is substantially shorter, because the data does not have to be loaded (10-20 seconds, depending on the amount of data).
- Memory is not fragmented. This effect can lead to the PC getting continually slower and having to be rebooted after a few hundred test runs.

Use of such a system does not change anything regarding the test program. No changes are necessary for the PSR. Merely the system file and the engine table must be appropriately prepared.

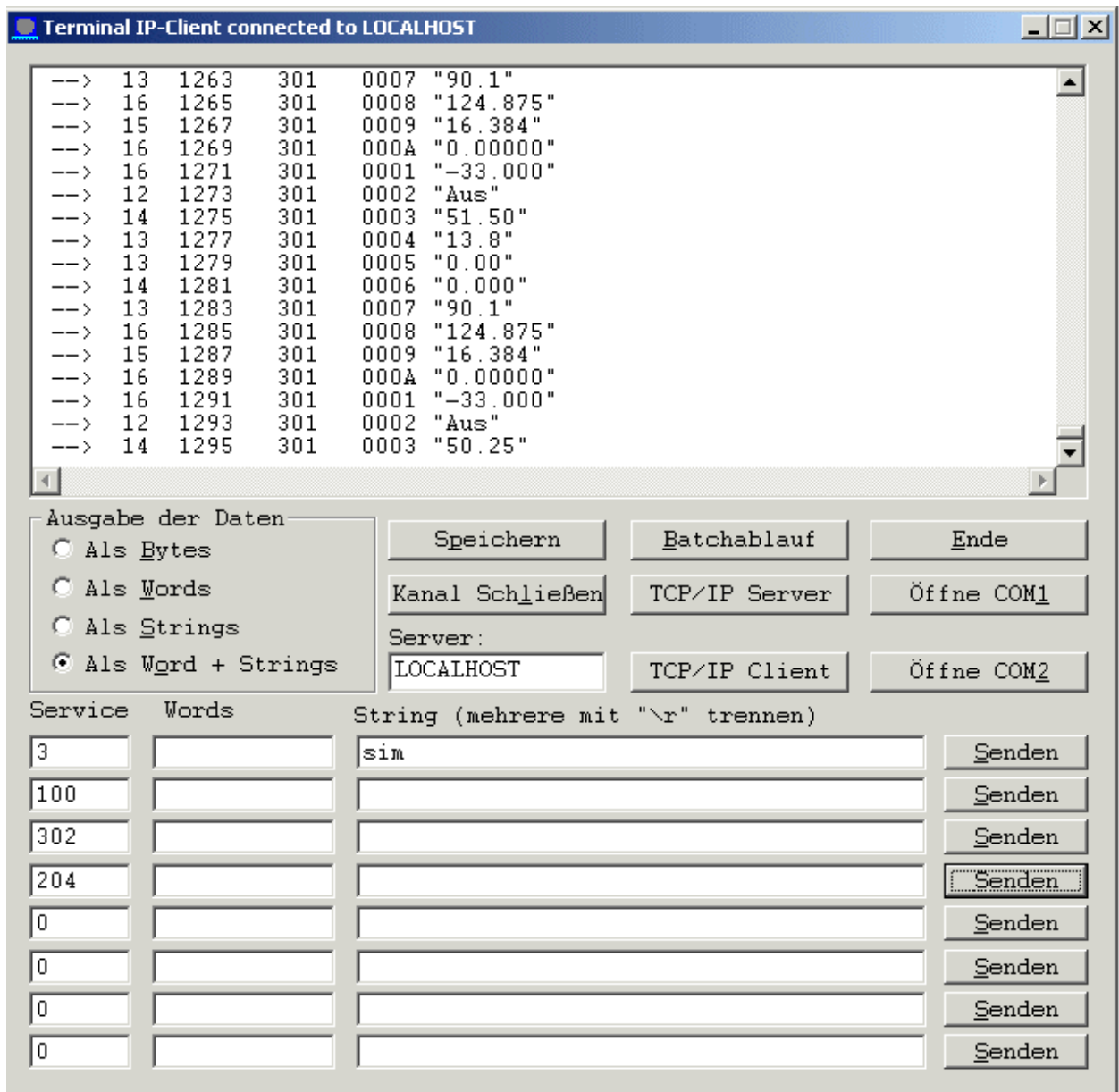
## **2.8.6 Examples - how can I...**

### **Test the functions of the PSRClient without the PSR**

The test program `PSRTerm.exe` can be found in the Vediamo directory. It uses the same communication interface as PSRClient and allows any messages which are usually sent by the PSR to be entered manually and to receive the responses from Vediamo.

Note

This is a test tool that is made available without guarantee and without support.



To work with PSRTerm, first start the PSRClient.

For a serial connection, the COM1 or COM2 interface of the computer on which PSRTerm is running must be connected with the Vediamo computer interface entered in Vediamo.ini by a null modem cable. This cable connects the pins GND, TX and RX of both ports, with TX connected to RX, RX connected to TX, and GND connected to GND. Contact can then be made to PSRClient by clicking on *Open COM1* or *Open COM2*.

For a network connection, both computers have to be in the same network and capable of being connected using the specified port. A successful PING is sufficient, DCOM

configuration, sharing, and releasing are not necessary. PSRClient can then be contacted by entering the name of the Vediamo computer or its IP address under *Server*, and then clicking on *TCP/IP Client* in the PSRTerm. Alternatively, PSRTerm can run on the same computer as PSRClient. In this case the computer name is "localhost". The address of your own computer is 127.0.0.1. If the connection cannot be established despite a successful PING, please check the settings of your firewall (e.g., Windows firewall).

To re-establish an interrupted connection, always click on *Close channel* first and then continue as above.

### **Test the PSR interface without starting Vediamo**

PSRTerm is suited for this task as well. In this case, start PSRTerm first and activate the connection with *TCP/IP Server* (for a network) or *Open COMx* (for a serial connection) before the PSR establishes contact.

Attention!

With a serial 3964R connection, the connection might become blocked as soon as both computers attempt to transmit a block at the same time. This is because both the PSR and PSRTerm have the parameter PRIO set to 1. In normal PSR-Vediamo operation this does not occur because PSRClient has PRIO=0.

### **Speed up measurement reading with DDLID**

Measurements can be read much more efficiently and faster from the ECU with the help of dynamically defined local IDs (DDLID). Another possible advantage is that the values read in this manner are calculated at the same time. DDLID are therefore used automatically for cyclic reading of measurements (command 302) if possible. The commands and the data format are not affected, only the speed of execution changes.

To use DDLID in the PSR client, the following conditions must be met:

- The ECU supports DDLID for the selected measurements
- The code for DDLID is entered correctly in the system file (VSB)
- No other client is using DDLID with the same ECU

### **Shorten the test run's initialization phase**

At the beginning of a test run, data is first loaded and initialized before contact can be established to the ECU. In the process, all measurement services defined in the basic ECU variant are initialized as software objects. After establishing contact and identifying the ECU variant, Vediamo initializes all measurement values defined in the identified variant.

Modern ECUs have up to several thousand measurement values which initialization, depending on PC performance, can take 2 to 3 minutes. However, since only a few of



these measurement values need to be measured in a test run, it makes sense to use the system configuration in the system description (VSB) to filter out all unneeded measurement values. This should be done for the basic variant (shortens initialization phase prior to establishing contact) as well as for the variant used (shortens initialization phase after contact has been established).

Filtering out other services (actuators, adjustments, etc.) does not influence the initialization phase because these services, unlike the measurement values, are only initialized in the PSR adapter when they are required.

As described [above](#), the initialization phase for data (prior to establishing contact) can be shortened substantially by implementing a system description with all required ECUs rather than multiple system descriptions. This can save up to 10 seconds per engine change. The PSR adapter does not reload the data if the required system description is identical to the preceding one.

### **Monitor communication between the PSR adapter and the PSR**

The worker client makes it possible to display and store the exchanged messages. This allows the test process to be monitored.

If there are problems in the communication between test bench and PSR adapter, it can be helpful to log the communication on a low level. Enter the name for the logfile in the INI file.

Example:

```
[PSR]
LOGFILE=W:\LOG\Luca.LOG
```

The next time the PSR adapter runs, all actually transmitted and received bytes are stored in the logfile. The following text box contains an example of a log file.

Example:

```
-----
PSR Comm log file W:\LOG\Luca.LOG
-----

Port: 2049
Date: 3.11.2005
Time: 9:4:19
-----

-1721665.-634[01] hdlc->APPL 0000 00 0d 00 e0 00 03 73 69 6d 32 36 36
00 '...`..sim266.'

-1721665.-544[01] APPL->hdlc 0000 00 11 01 3e 00 32 53 69 6d 75 6c 61
74 69 6f 6e '...>.2Simulation'
```

```
-1721665.-544APPL->hdlc 0010 00 '.'
-1721654.-408[01] APPL->hdlc 0000 00 36 01 ac 00 32 46 45 48 4c 45 52
3a 20 46 65 '.6.,.2ERROR: Er'
-1721654.-408APPL->hdlc 0010 68 6c 65 72 3a 20 55 6e 67 fc 6c 74 69 67
65 72 'ror: Invalid'
-1721654.-408APPL->hdlc 0020 20 53 79 73 74 65 6d 6e 61 6d 65 20 53 49
4d 32 ' system name SIM2'
-1721654.-408APPL->hdlc 0030 36 36 70 73 72 00 '66psr.'
-1721645.-986[01] hdlc->APPL 0000 00 06 01 a4 04 4c '...$.L'
-1721645.-916[01] APPL->hdlc 0000 00 13 02 03 04 4d 43 52 33 20 63 72
33 5f 75 70 '.....MCR3 cr3_up'
-1721645.-916APPL->hdlc 0010 73 72 00 'sr.'

-1721614.-110[01] APPL->hdlc 0000 00 30 03 3e 00 32 46 45 48 4c 45 52
3a 20 46 65 '.0.>.2ERROR: Er'

-1721614.-110APPL->hdlc 0010 68 6c 65 72 3a 20 55 6e 67 fc 6c 74 69 67
65 72 'ror: Invalid'

-1721614.-110APPL->hdlc 0020 20 53 79 73 74 65 6d 6e 61 6d 65 20 6e 61
67 00 ' system name nag.'

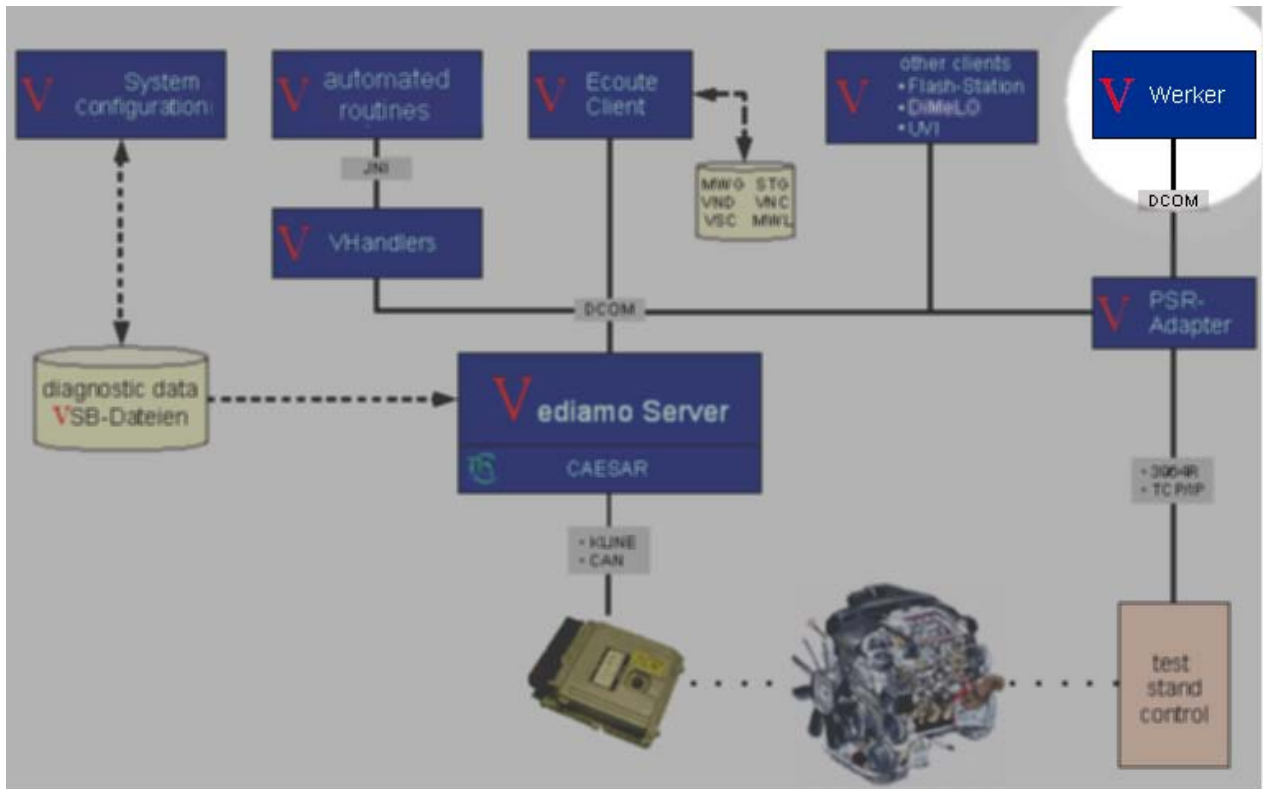
-1721155.-441[01] APPL->hdlc 0000 00 2e 15 2b 00 32 56 50 53 52 43 6c
69 65 6e 74 '...+.2VPSRClient'

-1721155.-441APPL->hdlc 0010 3a 20 56 65 72 62 69 6e 64 75 6e 67 20 77
69 72 ': Connection is'

-1721155.-441APPL->hdlc 0020 64 20 67 65 73 63 68 6c 6f 73 73 65 6e 00
'being closed.'
```

Closing log file: 9:13:30

## 2.9 2.9 Worker Client



### 2.9.1 Introduction

The worker client presents a graphical user interface for initializing the PSR client and for displaying:

- Measurements
- Errors
- Status messages on important status changes and actions

which are delivered by the diagnostic server during an automatic test run (controlled by the [PSR](#)). The application has no controlling influence on the testing. Only the display as well as the saving of log information can be influenced.

The PSR client is initialized through the establishment of the connection to the server. The PSR client is an independent program which forms the interface between diagnostic server and PSR. The PSR program has no user interface (window). The worker client is therefore necessary in order to monitor testing.

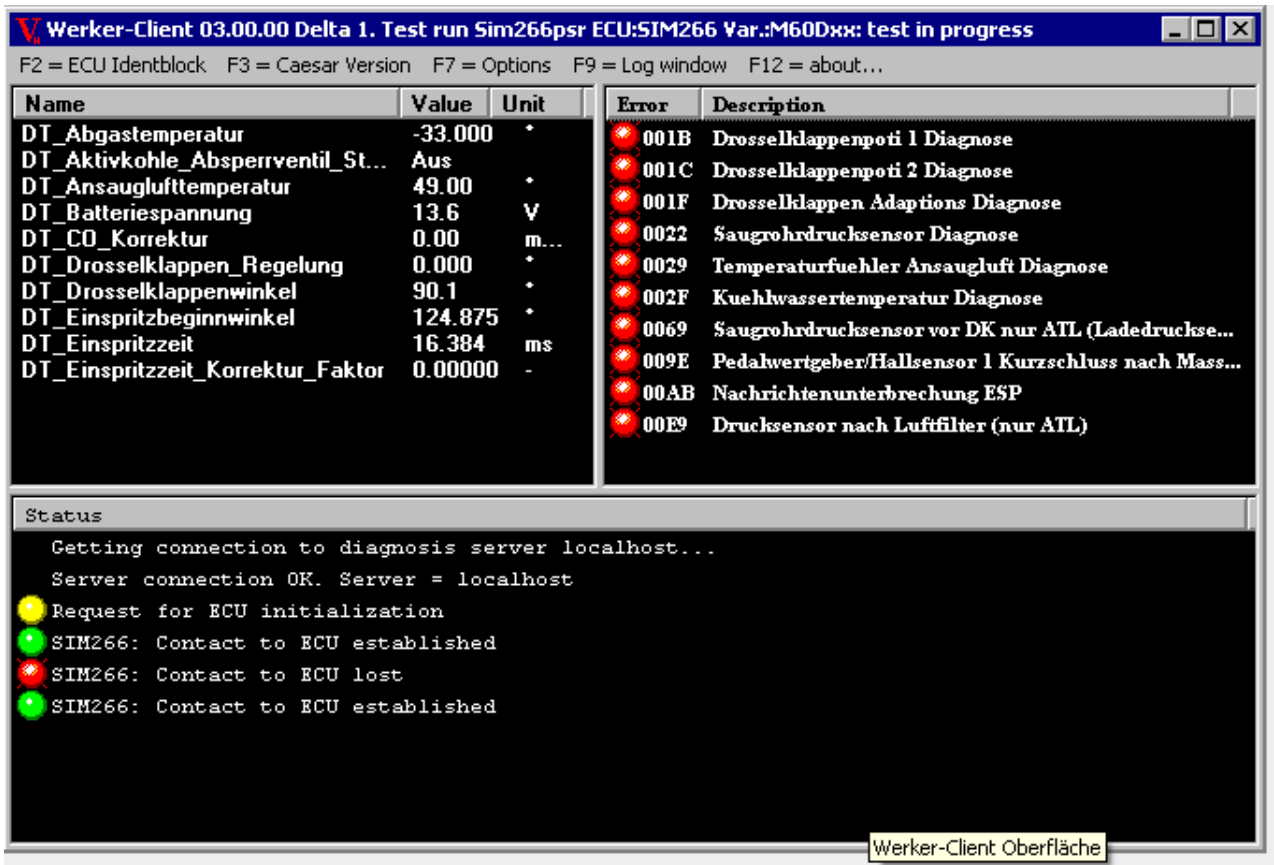
The command *Connect with Server* serves to initialize the PSR client. The PSR cannot establish contact with the server over 3964R or TCP/IP until afterwards. The number of worker clients (which can also be executed on various PCs) is not significant. Closing the

last client with a connection to the server ends the PSR client and - if no other Vediamo client is active - also ends the server.

The worker client user interface supports several languages. The language is specified by the Language entry in the [COMMON] of the Vediamo.ini configuration file.

## 2.9.2 Structure

The user interface consists of a title bar, a menu, and three display windows, as well as a separate log window:



### Measurement Window

All measurements which the PSR requests are displayed in this window (name, value, and units). When the measurement reading cycle is ended the last read value remains in the display, but in square brackets. The displayed values are cleared when new measurements are selected or a new test run starts.

### Error Window

All errors reported during a test run are displayed here (code and text). A lit red light next to an error code means that this error was reported as "current" in the last cycle. A dark red light denotes a "stored" error. The window stores all errors reported during the test run. The contents of the error window are not deleted until a new test run starts.

### **Status Window**

Important actions are displayed here: contact with the ECU (established, ended, and initialized), switching of an actuator, start and end of Java routines, as well as errors and other important events in the diagnostic server.

### **Tile Bar**

The current status of the test run is displayed here (number since program start, name of tested system, test run status, ECU: ID and variant).

### **Log Window**

This window is opened and closed with *F9 = log window*. This window shows the data blocks received and sent by the PSR. Only the last 1000 blocks are displayed, but it is possible also to continually store the data in a file.

## **2.9.3 Function Description**

### **Connection with the PSR adapter**

The worker client can receive data from a PSR adapter running on the same or on a different computer. The server computer can be selected in two ways:

Using command line parameters:

The name or IP address of the computer on which the PSR adapter is running is passed to the worker as the first parameter at the start.

With the F5 key:

The name of the server PC can be entered in the dialog which appears. An empty ID is the same as "localhost", i.e., the server on the same PC is contacted (and started, if necessary).

The ID of the contacted server is stored in the INI file when the program ends. This server is then suggested, when the user presses F5 after the subsequent start.

If a server connection already exists, no connection to a new server can be established. However, multiple clients can be started and connected to different servers.

### **Debug functions**

*F9 = Log window* opens or closes the log window. All blocks received and transmitted between PSR and PSR adapter are shown there. The log can be stored continually in a text file during runtime regardless if the log window is open or not. The content of the log window can be stored also at a later point in time, but only the last 1000 message blocks are available in the latter case.

The appropriate entry in the options menu automatically opens the log window when the program starts. A log file can be specified in the options in which the log is automatically stored.

#### **Important!**

The display in the log window does not always correspond exactly to the data exchange between PSR and Vediamo. It contains blocks which the PSR transmitted and which were correctly received by the Vediamo server, as well as blocks which Vediamo prepared for transmission to the PSR. It is therefore possible, if contact between the PSR and Vediamo is interrupted, that there are already blocks in the log which the PSR did not receive anymore. In addition, the times listed in the log are those from the diagnostic server and not the actual time of transmission.

If there are suspected problems with the data transmission, the data exchange should be saved in a separate file. This file can be specified in the `LOGFILE` entry in the `[PSR]` section of the `Vediamo.ini` file. This logfile, generated by the PSR client, contains an exact record of all bytes actually exchanged between Vediamo and PSR.

*F12 = Info...* displays a window containing the current program version as well as the path of the currently implemented INI file.

## **2.9.4 Examples: How can I...**

### **Display the identblock of the tested ECU**

F2 displays the identblock of the connected ECU, as long as a system is connected and the contact is established to the ECU.

### **Display the CAESAR software version**

F3 displays the version of the implemented CAESAR software (`c32s.dll`).

### **Adapt user interface**

the appearance of the worker client can be conveniently adapted to your needs. The position and size of the windows and subwindows can be changed by dragging with the mouse. In addition, F7 opens a settings window where the font and fontsize can be

changed (see more under [Configuration](#)). Any changes are automatically stored in [Vediamo.ini](#) when the program ends.

## 2.9.5 Command Line Parameters

If you start `Werker_Client.exe` with a command line parameter, the program tries to establish contact to the server immediately. The parameter must be either the computer name or the server IP address.

Example:

`werker_client localhost`

or

`werker_client 127.0.0.1`

## 2.9.6 Configuration (INI Parameter)

As with other Vediamo programs, the current program settings are stored in the `Vediamo.ini` file. The section `[Werker]` is reserved for the worker client.

### Settings

Server

Server name

The name of the PSR adapter is automatically copied into `Vediamo.ini` when connection is established. When the next call to establish connection is made after a subsequent program start, the name of the last contacted server is suggested.

Warnings

Display of warnings

This parameter can only be modified directly in the INI file (with `notepad.exe` or the [INI editor](#)). If it is set, warnings are displayed in the status window which can be ignored in a normal test run, but which can be very helpful for debugging the test bench when taking it into operation.

Logfile

If this entry is set to a valid filename, the specified file is opened automatically when the log window is opened (F9 key) and all log entries are saved continually.

### Configuration of user interface

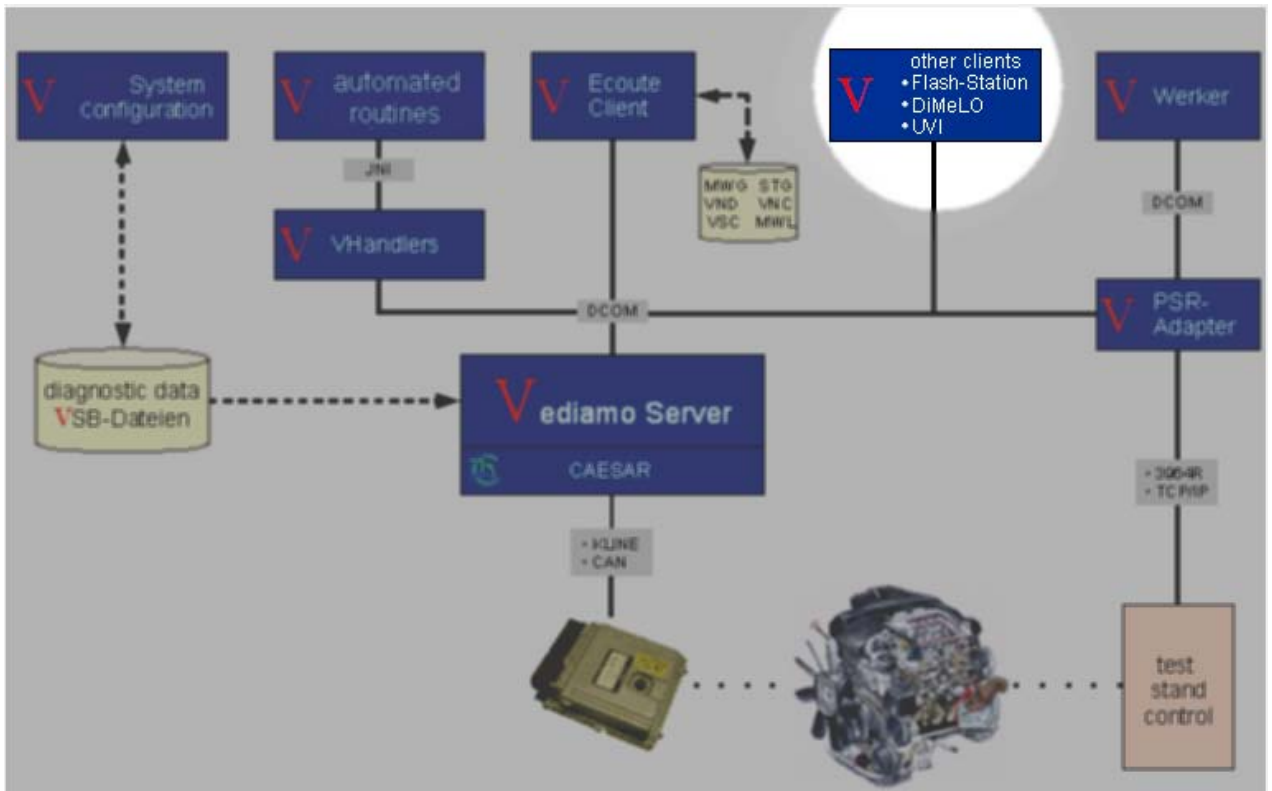
Window position and size can be changed by dragging the mouse. In addition, the following settings can be changed using the options dialog (menu entry *F7 = Options*):

- The fonts of the three subwindows
- Display of the measurements as name or qualifier
- Automatic opening of the log window during program start
- Automatic opening of the log file when the log window is opened

All these settings are automatically saved in the current `Vediamo.ini`. The program will have the same appearance the next time it starts as it did for the previous run.

The entries which apply to the appearance of the worker window (size, position, font) can only be modified directly in the client and not with an editor. Only in case of a "ruined" GUI configuration it might be recommended to delete all the entries in an editor in order to restore the window to its default state. This would apply to the parameter "Size" as well as all entries which contain the string "Font".

## 2.10 Other Clients



Additional Vediamo clients are mentioned here for completeness:

### 2.10.1 Flash Station

Vediamo Flash Station

The Vediamo flash station (VFS) is used for automated mass flashing and for process-pressure flashing.

User intervention in selecting the appropriate flashware is minimized, or selection can be omitted completely, through configuration. The flashing can then be performed even by untrained personnel (e.g., inventory update).

More information can be found in [Flash Station Help](#).



## 2.10.2 DiMeLo

Dimelo - automatized recording of measurement data.  
More information can be found in [Dimelo Help](#).

## 2.10.3 UVI

UVI - Unipas Vediamo Interface  
Unipas uses Vediamo over this interface to program (flash) ECUs and to obtain diagnostic information in XML format.

## 2.10.4 More Clients And Utilities

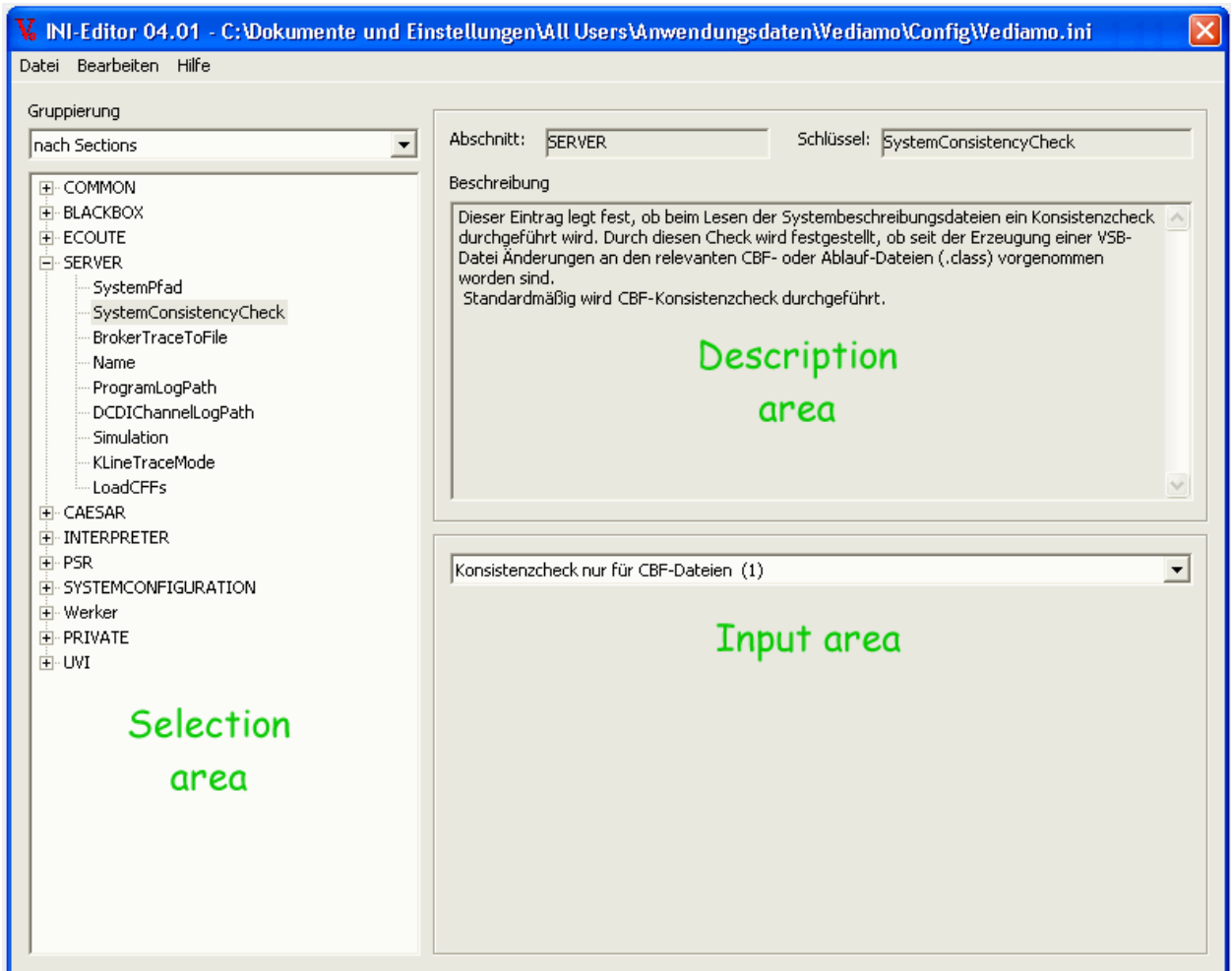
- **PSRChecker**  
This program is used in the acceptance procedure for implementing the Vediamo interface in test bench software. It simulates rare but possible problem scenarios which a test bench program must be capable of dealing with.
- [PSRTerm](#)  
This is a complimentary test program (without any entitlement to support and without liability) to manually simulate the communication with a PSR over TCP/IP as well as serial (3964R protocol).

## 2.11 INI Editor

This application is for setting all parameters used in the Vediamo system. It is activated from the StartCenter by clicking on the *Options* button on the toolbar. There is a detailed description for each parameter, along with legitimate values in some cases. The settings are stored in the [Vediamo.ini](#) file.

Please note:

If you change parameters which impact the function of a module, the change may not take effect until after a restart.



## 2.11.1 Menu

- File
  - Open: Selects an INI file and loads it.
  - Save: Writes all values in the INI file currently being edited.
  - Save as: Selects an INI file and writes all values in it.
- Edit
  - Reset this value: Resets the value being edited back to the original loaded value.
  - Reset all values: Resets all values (which the editor is aware of) back to the original loaded values.
- Help
  - Help: Calls up help.
  - About: Displays a dialog box with information on the INI editor

## 2.11.2 User Interface Areas

The setting to be edited is selected in the *Selection Area*. One or more different categories are available.

After selection, information on the selected setting is displayed in the *Description Area*.

Different input elements for modifying the setting are displayed in the *Input Area*.

## 2.11.3 Input Elements

### Input of boolean values (true/false, yes/no, ...)



A screenshot of a user interface element for a boolean value. It consists of a light gray rectangular box containing a small square checkbox with a checkmark inside, followed by the text "StatusTraceToFile".

To change the value, click on the box or select the ECU and press the space bar.

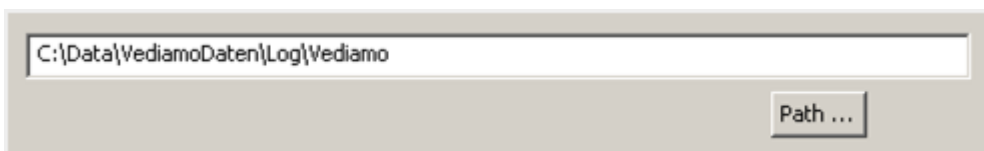
### Input of numbers and strings



A screenshot of a user interface element for a numeric value. It features a light gray rectangular box with a white input field at the top containing the number "1000". Below the input field, the text "Min: 100" is on the left and "Max: 10000" is on the right.

To change the value, click or select the input line and change its contents. The minimum and maximum limits for numbers have to be maintained.

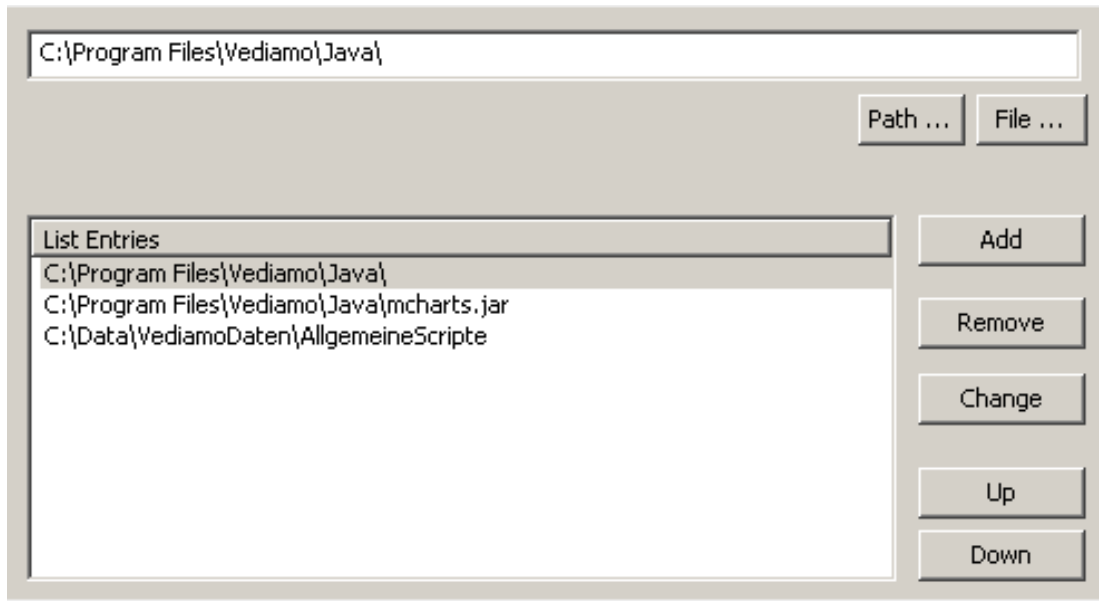
### Input of paths and files



A screenshot of a user interface element for a file path. It shows a light gray rectangular box with a white input field containing the path "C:\Data\VediamoDaten\Log\Vediamo". To the right of the input field is a button labeled "Path ...".

To change the value, click on the "..." button, or select it and press the space bar, then select the path or a file.

### Input of lists



Add:

Enter a value in the input line and press the "Add" button. The value is entered in the list below the selected value.

Remove:

Select a value from the list and press the "Remove" button.

Change:

Select a value from the list. Change the value in the input line. Press the "Change" button.

Up:

Select a value from the list. Press the "Up" button.

Down:

Select a value from the list. Press the "Down" button.

**ATTENTION!**

Only the content of the list is saved in the INI file. The content of the input line is ignored when saving.

**Input of predefined values**



Open the drop-down menu and make the appropriate selection.

## 2.11.4 All INI Parameters

A complete list of all parameters in the `vediamo.ini` file can be found in the attachment.

## 3 How Can I...

- [...Connect a vehicle](#)
- [...Connect an ECU without a Vehicle](#)
- [...Flash an ECU](#)
- [...Restart the Server](#)
- [...Read Measurements from an ECU](#)
- [...Read an ECU ID Block](#)
- [...Read and Clear an ECUs Error Memory](#)
- [...Execute a Quicktest](#)
- [...Perform Variant Coding](#)
- [...Execute a Java Routine \(Java Program\)](#)
- [...Change the Connection Between K-Line and CAN](#)
- [...Open Ecoute in the Same State I Closed It](#)

### 3.1 Connect a Vehicle

To connect a vehicle, you need one of the following alternatives:

- CAESAR Part A + B2 *or* Part Y *or* Part J **in addition to** Part E4 (with OBD connector)
- CAESAR Part C + special cable with OBD connector
- CAESAR Part X + WLAN connection on computer. Part X is equipped with an OBD connector.
- CAESAR Part W (SDConnect)

The OBD connector in each case requires a cable and connector corresponding to the hardware.

Insert the OBD connector into the vehicle's diagnostic socket (in the driver's footwell, pointing down).

If you use a Part E4, please configure `[CAESAR] USE_SIPartEDriver=1` in `Vediamo.ini`.

Please note:

The Part E4 must be connected to the vehicle before you start a Vediamo module, since the Part E4 requires the vehicle's voltage supply. If the Part E4 is not connected to the vehicle, it is not recognized by CAESAR.

If you do not use a Part E4, the setting above may not be used, i.e., `[CAESAR] USE_SIPartEDriver=0`

## 3.2 Connect an ECU without a Vehicle

To connect an ECU without a vehicle, you need one of the following alternatives:

- CAESAR Part A + B2 + F
- CAESAR Part Y + F
- CAESAR Part C + special cable with Part F
- CAESAR Part J + OBD connector + BreakOut-Box
- Part P - eCOM Box + F + USB2LAN Adapter
- CAESAR Part W (SDConnect)

as well as

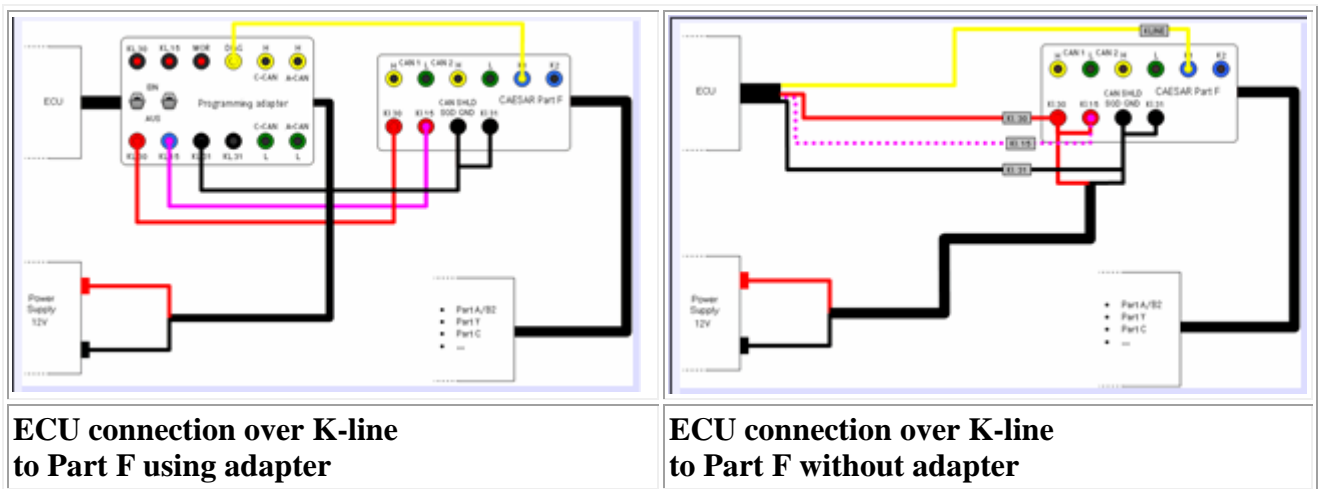
- ECU with connector and cable harness or adapter.  
To acquire a connector with cable harness or adapter, please contact one of the ECU's developers.
- Power supply to power the ECU (12V or 24V)

If you use an adapter, you will also need measurement cables with 4mm banana plugs.

### Connection to K-Line:

The available resources in Ecoute or in the system configuration determine whether the ECU can be addressed over K-line.

Connect the ECU as depicted in the following figures (click on figure for larger image).

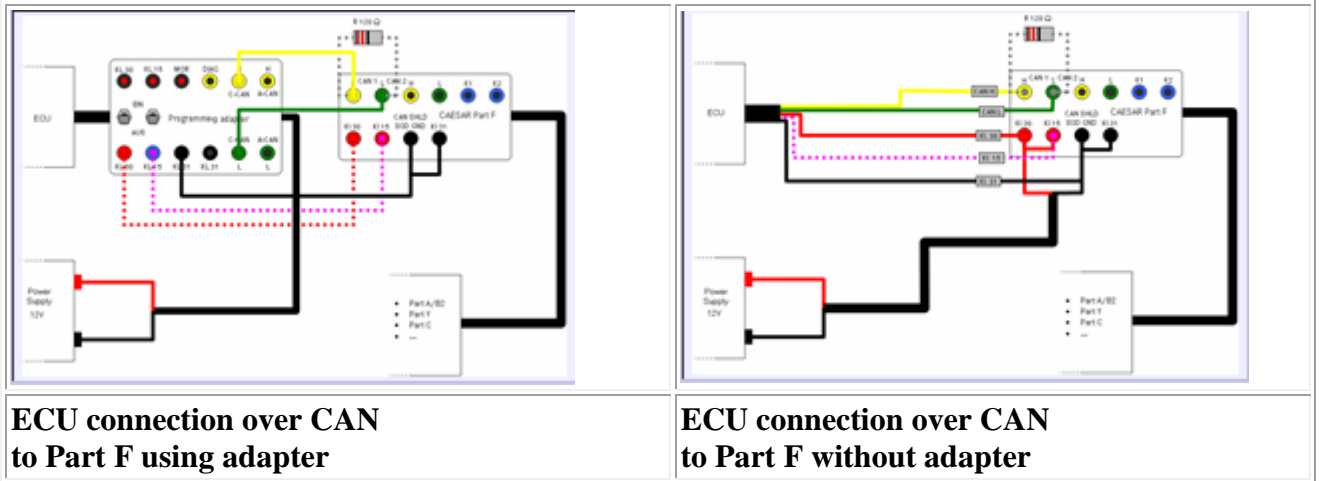


Please note, that a connection over K-line requires the supply voltage for Part F.

### Connection to CAN:

The available resources in Ecoute or in the system configuration determine whether the ECU can be addressed over CAN.

Connect the ECU as depicted in the following figures (click on figure for larger image).



Please note:

- A 120Ω termination may be necessary for the CAN, depending on whether the ECU has a terminating resistor installed or not.
- CAN SHLD (CAN shield/ground) is not identical to CAN Low. These contacts must not be connected.

### 3.3 Flash an ECU

**Starting point: .BIN-, .HEX- or .S19 files**

Vediamo can only flash CAESAR Flash Files (CFFs), i.e. you must first transform the binary files from the manufacturer into CFF format using [DIOGENES](#).

Preferred Method: Pass the files and various technical specifications to the person transcribing the ECU data file and receive the CFFs from them.

If this method cannot be applied (e.g., because of very short development cycles), you can also transform the CFFs yourself. Please address the person transcribing the ECU data file regarding this as well, since a description is beyond the scope of this document. Please note that the ECU name (DIOGENES name) in the CFF must be the same as the ECU name in the CBF.

**Starting point: .CFF files**

Copy the files into the directory specified in the options (`vediamo.ini`) under `[CAESAR]CFFPath`. If you like to sort the CFFs, you can create subdirectories in the specified CFF path and sort the CFFs there.

After you have copied the files, you must [restart the Vediamo server](#) if it is running (see taskbar).

### **Starting point: CFFs are in the CFF path and server is (re) started**

- Start Ecoute.
- Establish contact with the ECU you wish to flash.
- Do not execute any services or Java routines! Access privileges are set by the flash job.
- Open the flash dialog
- Proceed as described for [flashing with Ecoute](#).

Behaviour when loading CFF files:

To reduce server startup time, loading CFF files on server init depends on some `.ini` parameter settings:

In `vediamo.ini` there is a parameter `[Server]LoadCFFs`.

This parameter influences the loading of CFF files on server init and when using the files.

The parameter may be set to 4 values:

0 Load all CFFs: All CFFs located in the CFF-path are loaded on server start (default setting).

1 Do not load any CFF: No CFFs are loaded on server start. (load CFFs via "Flashdata administration")

2 Load CFFs from ECU subdirectory, when necessary: All CFFs located in \ are loaded on system selection.

3 Load CFFs from standard list, when necessary: Loads the flash files listed in the file \\Flashfiles.cfg are loaded on system selection.

See also Ecoute, "Flashdata administration"

## **3.4 Restart the Server**

The server must be restarted under the following circumstances:

- You have placed CAESAR files or Vediamo system descriptions (VSB) in the appropriate directory and want Vediamo to detect these files.

Important:

When restarting, make certain that no automated applications or Java routines are running, these could crash or at least not run properly if the server is restarted.

When Ecoute is running:

Select the item *Extras/ Restart Server* from the menu.



Otherwise:

End all applications which require the server.

Note:

The StartCenter can also require the Server

## **3.5 Read Measurements from an ECU**

### **3.5.1 Read individual measurements**

- Start Ecoute
- Open the system containing the ECU of interest
- Establish contact to the ECU
- Open the tree underneath the ECU
- Open the tree underneath the group *Measurements*
- Double-click on the measurement of interest
- Read the event in the status window

### **3.5.2 Read multiple measurements simultaneously or read measurements cyclically**

- Start Ecoute
- [Create a service group](#)
- Read the measurements by pressing the appropriate buttons

## **3.6 Read an ECU ID Block**

- Start Ecoute
- Open the system containing the ECU of interest
- Establish contact to the ECU
- Select the menu entry *ECU/ID Block* or use the key combination ALT + I
- If the information you require is not shown here, you can add [measurements to the ID block](#) using the [system configuration](#)

Note:

Not every ECU supports all the data which can theoretically be displayed in the ID block. Please also note the difference between ID block reading with 1A 86 and 1A 87.

## **3.7 Read and Clear an ECUs Error Memory**

- Start Ecoute
- Open the system containing the ECU of interest
- Establish contact to the ECU

- Select the appropriate menu entry [Error/...](#), depending on whether you wish to read or clear

Note:

You can set parameters for reading errors and also clear errors in the error window.

### **3.8 Execute a Quicktest**

Important:

Before you execute a quicktest, you need quicktest data. This data can be downloaded from the Vediamo homepage.

- Start Ecoute
- Select the menu entry *Error/Quicktest* or start Ecoute from the StartCenter with the parameter "Quicktest" or start Ecoute with the command line parameter `-k`
- [Configure the quicktest and execute it.](#)

### **3.9 Perform Variant Coding**

- Start Ecoute
- Load the system containing the ECU to be coded
- Establish contact to the ECU
- Select the menu entry [Coding / Variant coding](#)
- If all preconditions are correctly entered in the system description, they are automatically executed when the Varcode window is opened. If this is not the case, you can execute the necessary services(e.g., release, programming mode, etc.) yourself by double-clicking in the selection window.
- Proceed as [described above](#).

### **3.10 Execute a Java Routine (Java Program)**

#### **Storage of Java Routines**

Java routine files can be saved in any directory. To make them easier to find it is recommended to save them in a subdirectory of `\VediamoDaten`.

Java routines corresponding to a specific ECU, e.g. CR4 are best saved in the directory `VediamoDaten\CR4`, and Java routines corresponding to various ECUs should be saved in `VediamoDaten\AllgemeineScripte`. The data path for Java routines must always be specified in `Vediamo.ini` under `[INTERPRETER]ClassPath`, so that the Java interpreter can find the programs. Care should be taken that different versions of the same Java routine (same routine name) do not exist in different directories, since it cannot be guaranteed in that case that the expected routine version is executed.

#### **Execution from Ecoute or PSR:**

In order to be able to execute a Java routine (i.e., make it visible) from a Vediamo application (Ecoute or PSR), the routine name must be entered in the VSB using the system configuration.

The Java routine is displayed in Ecoute in the system window under *routines* and can be started by a double-click.

The routine is started in the PSR adapter upon the command "600, routine name" from the PSR.

### Execution using the command line

The following must be entered in a batch file or in the command line:

#### In general:

```
javaw -classpath <JavaKlassenPfad_1>;<JavaKlassenPfad_2>;...; <routine name>
```

#### Example:

```
"C:\Program Files\Vediamo\JRE\bin\javaw" -classpath  
"C:\ProgramData\Vediamo\VediamoDaten\AllgemeineScripte";"C:\Program  
Files\Vediamo\Java"; ExampleRoutine
```

## 3.11 Change the Connection Between K-Line and CAN

- Connect the ECU as described [above](#)
- Start Ecoute
- Load the system containing the ECU of interest
- Select the menu entry *Properties* from the ECU context menu (right mouse key on ECU)
- Select the desired connection in the subsequent dialog, if necessary, take the connection away from an ECU that is not required at the moment.
- Establish contact to the ECU

If the desired connection is not available, this can have various causes:

- The ECU cannot be communicated with over this connection. That a physical connection exists does not mean that the ECU serves that connection. And if the ECU serves the connection, that does not mean it is parameterized. Keyword [parameterization, CBF](#).
- The protocol for the loaded firmware is not implemented. Keyword [CaesarGo ↔ TLSlave Firmware](#)

## 3.12 Open Ecoute in the Same State I Closed It

- Start Ecoute
- Configure Ecoute the way you want to find it the next time you start (system, window positions, etc.)
- Save a session by selecting the menu entry *System / Save session as*
- Open the Ecoute options by selecting the menu entry *Extras / Options*. Select the *Start* page and activate the option *Load last session*

Alternate possibility:

- If you only want to reload the last system, then select the menu entry *Extras/Options*, Go to the *Start* page and activate the option *Load last system*. Window positions are not saved
- Pass the name of the session file to Ecoute in the command line
- Create a start profile for the [StartCenter](#), with which you pass a saved session to Ecoute in the command line

## 4 Glossary

Term	Explanation
3964R	Communication protocol between PSR and Vediamo for a serial connection
Area	The section of ECU memory to be flashed
CAN	Controller Area Network. Serves for communication between ECUs within a vehicle and between vehicle and tester.
CAESAR	Common Access To Electronic Systems of Automotive Requirements. CAESAR consists of a software and a hardware component.
CBF	CAESAR Binary Format - binary format with ECU description data
CCF	CAESAR Coding File - binary format with variant coding data
CFF	CAESAR Flash File - binary format with flash data. Can contain multiple areas / FlashKeys.
Configuration File	<a href="#">Vediamo.ini</a> (see below)
Diagnostic Pin	Physical K-line pin between ECU and tester (e.g., on CAESAR Part C)
DiagService	Diagnostic service - symbolic communication with ECU, e.g., read measurement
DiagJob	Diagnostic Job - procedures interpreted by CAESAR which can contain, e.g., DiagServices
DIOGENES	Diagnostic data input and data management with SGML
DCOM	Distributed Common Object Model - Microsoft model for programming distributed applications
ECU	Electronic Control Unit
FlashKey	Unique key identifying the contents to be flashed
FlashWare	Software stored permanently in ECU
Gateway	Gateway ECUs allow access to the member ECU connected to it. In many cases, a gateway has a K-line and allows access to a CAN bus to which the member ECUs and the gateway itself are connected.
GBF	GPD Binary Format - binary format, contains a precompiled GPD

GPD	General Protocol Description - general description language for communication protocols
JNI	Java Native Interface - interface for connecting Java code to Win32 code (e.g., in C++ programmed DLLs). Used in <a href="#">VHandlers.dll</a> to make Vediamo classes "visible" for Java programs.
Java	Object oriented programming language from <a href="#">Sun Microsystems</a> . Used in Vediamo for automated routines.
Handler Functions	Also: Built-in Function. Permanently "integrated" Vediamo-Java interface functions ( <a href="#">VHandlers.dll</a> )
ISOSCAN	Application for K-line diagnostics of ECUs
K-Line	Serial connection to ECU
Log File	File containing log information. See logging.
Logging	Also: Tracing. Record of program sequence for debugging (e.g., names of called functions, etc.)
LUCA	<a href="#">Langner</a> Universal Communications API - Software for communication protocols. Used in PSRClient.
Meaning	Contents to be flashed to a specific area
Member	Member ECUs can only be diagnosed by way of a gateway ECU (see above).
MPF	Engine test facility (Motorenprüffeld)
MIL	Malfunction indicator light - ECU error status bit
Name	Spoken name of a service (e.g., "RPM"). Not unique.
OutputRef	Service that gets the set value of another service(e.g., actuator) from an ECU.
Preparation	Input parameter for services
Presentation	Result interpreted from the ECU reply
Precondition	This is what services are called which execution is a precondition for another action in the parameterization ( <a href="#">CBF</a> ) of the ECU. In contrast to this, preconditions can be specified by the user independent of CBF specifications in Vediamo files ( <a href="#">VSB</a> , <a href="#">MWG</a> , <a href="#">STG</a> ) as well and can reference Java routines as well as diagnostic services.

Routine	Also: Java routine, automated test routine, Java program, Java client. A Java program which accesses the diagnostic functionalities of the DiagServer using the VHandlers interface. See the chapter on <a href="#">Java Programs</a> for more information.
PSR	test bench controller (Prüfstandrechner) - controls engine test benches. Obtains diagnostic data from Vediamo using the PSRClient.
PSR Protocol	Protocol for serial as well as TCP/IP connections. Existing protocol between PSR and diagnostic computer. This protocol has been used in the MPF program since the early '90s (serial only).
QDE	Quality data collection computer (Qualitätsdatenerfassungsrechner)
Qualifier	Unique service ID (e.g., "DT_RPM").
Script	Old term for <a href="#">Java routine</a> (known as "functional chain" prior to that). In Vediamo, automated routines are written in Java. See the chapter on <a href="#">Java Programs</a> for more information.
Siemens Protocol 3964R	Serial protocol between PSR and diagnostic server, used over COMx.
Standard Service / Standard Objekt	Certain services are named different in DIOGENES. Therefore they are given a fixed standard name (e.g., Bandend, Unlock, Initialize) in Vediamo to ensure a unique ID.
System	More precisely: ECU system. The combination of one or more ECUs is referred to as a system in Vediamo e.g. a motor with two ECUs is a system.
TL Slave	CAESAR firmware executed in the CAESAR master
Monitoring	Logging of the diagnostic protocol between tester and ECU
Vediamo	Distributed diagnostic application for engines (Verteilte Diagnose Anwendung für Motoren)
Vediamo.ini	Configuration file for the Vediamo modules. All the parameters the Vediamo system requires are set in this file. The entries are sorted by module. Some settings are saved automatically by programs; others can only be edited by the user using the <a href="#">INI editor</a> .
VOM	Vediamo Object Model - object model relating to the engine test facility, for diagnosing ECUs
VND file (vnd)	Vediamo message description file - contains information for manual command input
VSB file (vsb)	Vediamo system description file - contains information on ECU systems. Is created using the <a href="#">system configuration</a> and loaded by the

	<a href="#">DiagServer.</a>
--	-----------------------------



## 5 INI Parameters

<b>COMMON</b>						
<b>Name</b>	<b>Description</b>	<b>Comments</b>				
Language	<p>This entry specifies the language for the application. The prerequisite is that the corresponding versions of the resource files are available. The language is specified by an abbreviation (e.g., EN=English, FR=French, SP=Spanish, etc.). One DLL file per language is created for each application. The default language is German.</p> <p>The translated text from the DiagServer contains the file VCommon_Res_XY.dll, where XY is the language abbreviation. The remaining applications use files whose names are formed from the names of the respective applications: [Application name]_Res_XY.dll, e.g., Werker_Client_Res_EN.dll.</p>	<p>Default value: DE</p> <p>Possible values:</p> <table border="1" style="margin-left: 20px;"> <tr> <td>DE</td> <td>German</td> </tr> <tr> <td>EN</td> <td>English</td> </tr> </table>	DE	German	EN	English
DE	German					
EN	English					
<b>BLACKBOX</b>						
<b>Name</b>	<b>Description</b>	<b>Comments</b>				
LogFile	<p>Path and primary name of the program logfile. All Vediamo program log outputs are buffered by BlackBox and only saved to the file specified here if a critical error occurs or the user issues the explicit command. The current data and time are automatically appended to the filename when the file is saved. The extension ".log" is also added.</p> <p>The default value is (when the entry is empty): ".\BLACKBOX", so the logfile is stored in the directory in which the program file BlackBoxServer.exe is located.</p>	<p>Default value: ...\\Vediamo\\Log\\BlackBox</p>				
LineNum	<p>Number of lines to be stored in the ring buffer. For a bigger amount of running Vediamo applications and for a more complex action, the number should be greater.</p> <p>Every line uses 150 byte of main PC</p>	<p>Default setting: 5000</p> <p>Minimum value: 500</p>				

	memory. Default value: 5000, sufficient for normal use. Min value: 500, entering lower values results in reserving 500 lines.									
<b>ECOUTE</b>										
Name	Description	Comments								
StatusTraceToFile	This entry specifies whether the content of the Ecoute output window should also be saved in a file. The output is saved in "<[ECOUTE]ProgramLogPath>\Status.log".	Default setting: 0								
ProgramLogPath	This entry specifies in which directory to save the Ecoute status logfiles (the output in the status window) should be saved.	Default setting: ...\Vediamo\Log\C lient								
TraceFileMaxSize	This entry specifies the maximum size (in bytes) of the log files. The default setting is 524288 (1/2 MB). The value 0 means the size is not limited.	Default setting: 524288 Minimum value: 0								
DCDIChannelLogPath	This entry specifies in which directory to save the Ecoute ECU related logfiles.	Default setting: ...\Vediamo\Log\E coute								
KLineTraceMode	This entry specifies whether the ECU communication should be displayed in the trace window blockwise, detailed, or not at all. The output is also saved in "<[ECOUTE]DCDIChannelLogPath>\<system>\<ECU>Trace.log". The default setting is not to display.	Default setting: 0 Possible values: <table border="1" style="border-collapse: collapse; width: 100%;"> <tr> <td style="text-align: center;">0</td> <td>Do not display</td> </tr> <tr> <td style="text-align: center;">1</td> <td>Display blocks</td> </tr> <tr> <td style="text-align: center;">2</td> <td>Display detailed (bytes &amp; timing)</td> </tr> <tr> <td style="text-align: center;">3</td> <td>Display blocks and detailed</td> </tr> </table>	0	Do not display	1	Display blocks	2	Display detailed (bytes & timing)	3	Display blocks and detailed
0	Do not display									
1	Display blocks									
2	Display detailed (bytes & timing)									
3	Display blocks and detailed									
ShortTestDataDir	This entry specifies the quickest directory. The data for the quickest must be provided in this directory in subdirectories corresponding to the different models. The default setting is "\Data\VediamoKurztestDaten".	Default setting: ...\Vediamo\Vedia moShorttestData								
LoadLastSession	This entry specifies whether the last saved session file should be loaded when Ecoute starts.	Default setting: 0								

VSPATH	This entry specifies the path and the filename of the last saved session file.									
AutoInitOnSystemSelection	This entry specifies whether, subsequent to a system change, the attempt should be made to establish contact automatically with all the ECUs of the new system.	Default setting: 0								
AutoInitOnContactLost	This entry specifies whether, subsequent to a loss of contact, cyclical attempts should be made to re-establish contact with the system's ECUs. The default setting is no.	Default setting: 0								
AutoInitOnFailedUserInit	This entry specifies whether, if the user failed to establish contact, cyclical attempts should be made to establish contact. The default setting is no.	Default setting: 0								
ExecuteInitSequence	This entry specifies whether the initialization routine should be executed. The default setting is not to execute the routine.	Default setting: 0 Possible values: <table border="1"> <tr> <td>0</td> <td>Initialization routine is not executed</td> </tr> <tr> <td>1</td> <td>Initialization routine is executed after system selection</td> </tr> <tr> <td>2</td> <td>Initialization routine is executed after contact is established</td> </tr> <tr> <td>3</td> <td>Initialization routine is executed after system selection and contact is established</td> </tr> </table>	0	Initialization routine is not executed	1	Initialization routine is executed after system selection	2	Initialization routine is executed after contact is established	3	Initialization routine is executed after system selection and contact is established
0	Initialization routine is not executed									
1	Initialization routine is executed after system selection									
2	Initialization routine is executed after contact is established									
3	Initialization routine is executed after system selection and contact is established									
SnapshotFormat	Specifies the file format (HTML or text) for saving window contents. This does not apply to cyclical saving of measurements with cycle times > 0 (always CSV). The default setting is HTML format	Default setting: html Possible values: <table border="1"> <tr> <td>text</td> <td>Text format</td> </tr> <tr> <td>html</td> <td>HTML format</td> </tr> </table>	text	Text format	html	HTML format				
text	Text format									
html	HTML format									
VarCodStringFormat	This entry specifies the coding string format. The default setting is decimal format.	Default setting: Decimal Possible values: <table border="1"> <tr> <td>Decimal</td> <td>Decimal</td> </tr> <tr> <td>Hexadecim</td> <td>Hexadecim</td> </tr> </table>	Decimal	Decimal	Hexadecim	Hexadecim				
Decimal	Decimal									
Hexadecim	Hexadecim									

		al	al				
TreeStyle	<p>The system services and routines are displayed in the system window as a tree. The branches of the tree are sorted in the categories specified in Vediamo such as measurements, actuators, functions, etc. Alternatively, the services can be sorted by the categories parameterized in DIOGENES.</p> <p>In this case, the services in the system window are unfiltered (i.e., all parameterized services are displayed, not only those specified in the VSB file). The default setting is as in the Vediamo system description.</p>	<p>Default setting: VSB</p> <p>Possible values:</p> <table border="1"> <tr> <td>VSB</td> <td>As in Vediamo system description</td> </tr> <tr> <td>DIO</td> <td>As in DIOGENES</td> </tr> </table>	VSB	As in Vediamo system description	DIO	As in DIOGENES	
VSB	As in Vediamo system description						
DIO	As in DIOGENES						
UseFilters	<p>This entry specifies whether the filters specified in the VSB are applied to the diagnostic services.</p> <p>The default setting is yes.</p>	<p>Default setting: 1</p>					
GlobalFilter	<p>Filters certain services according to the prefix. Services with the prefixes specified here are shown as filtered if a system description is created. The filtering can be inactivated as usual in the system configuration. Services with the specified prefixes are not displayed in Ecoute. The exception is if the filtering has been inactivated using the configuration tool.</p> <p>If Ecoute is being used without a .vsb (i.e., with a system description based on DIOGENES parameterization (default .vsb)), the prefixes specified here also apply.</p> <p>Example : GlobalFilter = DNU_, WVC_, SES_, NR_, RVC_, SRC_</p>	<p>Default setting: DNU_, WVC_, SES_, NR_, RVC_, SRC_</p>					
DisplayQualifiers	<p>This entry specifies whether names or qualifiers are shown in the tree.</p>	<p>Default setting: 1</p> <p>Possible values:</p> <table border="1"> <tr> <td>0</td> <td>Names</td> </tr> <tr> <td>1</td> <td>Qualifiers</td> </tr> </table>	0	Names	1	Qualifiers	
0	Names						
1	Qualifiers						
GraphMaxPoint	<p>This entry specifies the maximum number</p>	<p>Default setting:</p>					

s	of measurement points a measurement curve may have. The default setting is 1000.	1000 Minimum value: 100 Maximum value: 10000
ShowDefaultSystems	This entry specifies whether the systems from the DIOGENES parameterization (CBFs) should be included in the system selection display. The default setting is no.	Default setting: 0
ShowVSBSystems	This entry specifies whether the systems from the Vediamo system description should be included in the system selection display. The default setting is yes.	Default setting: 1
ShowVariantDetectionDialog	This entry specifies whether the dialog for automatic variant recognition should be displayed if only the basic variant of an ECU was identified when contact was established. The default setting is yes.	Default setting: 1
EcuExitInitAfterFlashing	This entry specifies whether contact to the ECU should be automatically ended and re-established (ECU reset) if no shut down cycle is performed after flashing. The default setting is yes.	Default setting: 1
<b>SERVER</b>		
<b>Name</b>	<b>Description</b>	<b>Comments</b>
SystemPfad	This entry specifies in which directory the diagnostic server can find the system description files it should use. In addition, Vediamo scripts are looked for in this path (see ClassPath in the [INTERPRETER] section). Valid entries are normal path specifications with drive and directory. The default setting is ".\" (current director).	Default setting: .\
SystemConsistencyCheck	This entry specifies whether a consistency check is performed when the system description files are read. This check determines if the relevant CBF or script files have been changed after a VSB file was created. By default, no consistency check is	Default setting: 0 Possible values (the values may be combined using the logical AND operator):

	made.	<table border="1"> <tr> <td>0</td> <td>Do not perform consistency check</td> </tr> <tr> <td>1</td> <td>Consistency check for CBF files</td> </tr> <tr> <td>2</td> <td>Consistency check for Java routines</td> </tr> <tr> <td>3</td> <td>Consistency check for GBF files (only alog entry is created on error)</td> </tr> </table>	0	Do not perform consistency check	1	Consistency check for CBF files	2	Consistency check for Java routines	3	Consistency check for GBF files (only alog entry is created on error)
0	Do not perform consistency check									
1	Consistency check for CBF files									
2	Consistency check for Java routines									
3	Consistency check for GBF files (only alog entry is created on error)									
BrokerTraceToFile	This entry specifies whether or not commands executed by the broker level between the diagnostic server and its clients are logged in the trace file. The standard setting is no.	Default setting: 0								
Name	<p>This entry specifies the name of the computer on which the diagnostic server process is running and which Ecoute should communicate with. If no name is specified or the name is an empty string (Name=), it is assumed that the diagnostic server process is running on the same computer. The name can be entered in UNC notation (e.g., \\Server), DNS notation (e.g., Server.com) or as an IP address (e.g., 123.45.67.89). If the computer is in the same domain, only the name must be entered (e.g., Name = August).</p> <p>Note that the server and CAESAR settings of this INI file do not apply to diagnostic server processes which are not running on the same computer as the client (Ecoute/PSRClient/...).</p>									
ProgramLogPath	This entry specifies in which directory the CAESAR logfiles are stored (see [CAESAR] Debuglevel).	Default setting: ...\\Vediamo\Log\\Server								
DCDIChannelLogPath	This entry specifies in which directory the ECU-related server log files should be saved.	Default setting: ...\\Vediamo\Log\\Server								
Simulation	The Vediamo server can be operated in	Default setting:								

	<p>three modes: normal operation, simulation and SIM recording. The default setting is normal operation.</p>	<p>0</p> <p>Possible values:</p> <table border="1"> <tr> <td>0</td> <td>normal operation</td> </tr> <tr> <td>1</td> <td>Simulation</td> </tr> <tr> <td>2</td> <td>Record simulation data</td> </tr> </table>	0	normal operation	1	Simulation	2	Record simulation data		
0	normal operation									
1	Simulation									
2	Record simulation data									
KLineTraceMode	<p>This entry specifies whether the ECU communication (K-line and CAN) should be logged blockwise, in detail, or not at all. The output is saved in "<code>&lt;[SERVER]DCDIChannelLogPath&gt;\&lt;System&gt;\&lt;ECU&gt;Trace.log</code>". The default setting is not to log.</p>	<p>Default setting: 0</p> <p>Possible values:</p> <table border="1"> <tr> <td>0</td> <td>Do not log</td> </tr> <tr> <td>1</td> <td>Log blockwise</td> </tr> <tr> <td>2</td> <td>Log in detail (bytes &amp; timing)</td> </tr> <tr> <td>3</td> <td>Log blockwise and in detail</td> </tr> </table>	0	Do not log	1	Log blockwise	2	Log in detail (bytes & timing)	3	Log blockwise and in detail
0	Do not log									
1	Log blockwise									
2	Log in detail (bytes & timing)									
3	Log blockwise and in detail									
LoadCFFs	<p>This entry controls the loading of CFF files on server initialization and when calling the LoadCFFs function.</p>	<p>0 = Load all CFFs 1 = Do not load any CFF 2 = Load CFFs from ECU subdirectory, when necessary 3 = Load CFFs from standard list, when necessary</p>								
OBDDProtocol	<p>One of the protocols defined in protocol.gbf, to be used for OBD communication</p>	KW2C3PE								
OBDBaudrate	<p>The CAN baudrate for the OBD communication</p>	50000								
OBDServiceFile	<p>This file defines all services, PIDs, TIDs, MIDs etc. It contains also a reference to a file defining all error codes (DTCs). Both files must be in the same directory.</p>	.\OBD\OBD2.xml								
OBD_P2_MAX	<p>Normally the P2 defined in the protocol is sufficient. When timing problems occur (possible with eCOM), a higher value might make the communication more stable.</p>	250								
OBD_REQREP_COUNT	<p>Normally every message is sent once. When timing problems occur (possible with eCOM), a higher value might make the communication more stable.</p>	1								

<b>CAESAR</b>		
<b>Name</b>	<b>Description</b>	<b>Comments</b>
LANGUAGE	This entry specifies the language for the text generated by CAESAR (error messages, names of services). The prerequisite is that the appropriate language files (*.CTF) are available. The language is denoted by an abbreviation (e.g., EN=English, FR=French, SP=Spanish etc.).	
CBFPFAD	This entry specifies in which directory CAESAR can find the CBF files to use. Valid entries are normal path specifications with drive and directory. The default setting is ".\" (current directory).	Default setting: .\
GBFPFAD	This entry specifies in which directory CAESAR can find the GBF files to use. Valid entries are normal path specifications with drive and directory. The default setting is ".\" (current directory).	Default setting: .\
DRIVERPFAD	This entry specifies in which directory CAESAR can find the driver files to use. Valid entries are normal path specifications with drive and directory. The default setting is ".\" (current directory).	Default setting: .\
CFFPFAD	This entry specifies in which directory CAESAR can find the CFF files to use. Valid entries are normal path specifications with drive and directory. The default setting is ".\" (current directory).	Default setting: .\
CTFPFAD	This entry specifies in which directory CAESAR can find the CTF files to use. Valid entries are normal path specifications with drive and directory. The default setting is ".\" (current directory).	Default setting: .\
CCFPFAD	This entry specifies in which directory CAESAR can find the CCF files to use. Valid entries are normal path specifications with drive and directory. The default setting is ".\" (current directory).	Default setting: .\
DEBUGLEVEL	This entry specifies the degree of detail with which the CAESAR actions are logged in the file	Default setting: 1 Possible values:



	"<[SERVER]ProgramLogPath>\CAESAR.log". These files are always generated!	<table border="1"> <tr><td>0</td><td>No output</td></tr> <tr><td>1</td><td>Module related</td></tr> <tr><td>2</td><td>Function related</td></tr> <tr><td>3</td><td>Internal function details</td></tr> <tr><td>4</td><td>Maximum</td></tr> </table>	0	No output	1	Module related	2	Function related	3	Internal function details	4	Maximum
0	No output											
1	Module related											
2	Function related											
3	Internal function details											
4	Maximum											
CHANNELDEBUGLEVEL	<p>This entry specifies the degree of detail with which channel related CAESAR actions are logged. A logfile "&lt;[SERVER]DCDICchannelLogPath&gt;\&lt;System&gt;\&lt;ECU&gt;Kanal.log" exists for every channel.</p> <p>Logging can be turned on and off using Ecoute.</p>	<p>Default setting: 1</p> <p>Possible values:</p> <table border="1"> <tr><td>0</td><td>No output</td></tr> <tr><td>1</td><td>Module related</td></tr> <tr><td>2</td><td>Function related</td></tr> <tr><td>3</td><td>Internal function details</td></tr> <tr><td>4</td><td>Maximum</td></tr> </table>	0	No output	1	Module related	2	Function related	3	Internal function details	4	Maximum
0	No output											
1	Module related											
2	Function related											
3	Internal function details											
4	Maximum											
TraceDebugOutput	<p>This entry specifies whether the log information (DebugOutput and ChannelDebugOutput) generated by CAESAR are displayed in the BlackBoxViewer window.</p> <p>Enhances debugging, e.g., when developing diagnostic jobs... (see also the entries for DEBUGLEVEL and CHANNELDEBUGLEVEL).</p> <p>The default setting is no.</p>	<p>Default setting: 0</p>										
USE_SISerialDriver	<p>This entry specifies whether the driver for CAESAR Part D should be loaded.</p> <p>The default setting is yes.</p>	<p>Default setting: 1</p>										
USE_SIPartCDriver	<p>This entry specifies whether the driver for CAESAR Part C should be loaded.</p> <p>The default setting is no.</p>	<p>Default setting: 0</p>										
USE_SIPCMCIADriver	<p>This entry specifies whether the driver for CAESAR Part A should be loaded.</p> <p>The default setting is no.</p>	<p>Default setting: 0</p>										
USE_SIPartXDriver	<p>This entry specifies whether the driver for CAESAR Part X should be loaded.</p> <p>The default setting is no.</p> <p>In addition, the settings in CAESAR's own</p>	<p>Default setting: 0</p>										

	<p>slave.ini file in the driver directory are used for operating CAESAR Part X. The default setting is no.</p>	
USE_SIPartYDriver	<p>This entry specifies whether the driver for CAESAR Part Y should be loaded. The default setting is no.</p>	Default setting: 0
USE_SIPartJDriver	<p>This entry specifies whether the driver for CAESAR Part J (PassThru Vehicle Communication Interface) should be loaded. The default setting is no.</p>	Default setting: 0
USE_SIPartEDriver	<p>This entry specifies whether the driver for CAESAR Part E should be loaded. The default setting is dependent on the diagnostic server subcomponent selection during installation. The default setting is no.</p>	Default setting: 0
PinMapping	<p>This entry specifies whether PIN mapping should be activated or not when using CAESAR Part E. If the setting is made to use CAESAR Part E in the subcomponent selection during installation of the diagnostic server, then PIN mapping is activated.</p>	Default setting: 0
GPDFlashCaching	<p>Activates the cache mechanism for GPD in the CAESAR slave: If the required GPD is not already in the flash EEPROM, it is written into the flash EEPROM. GPDs stored in the EEPROM are executed from flash memory. When GPDFlashCaching is 0, GPDs are always executed in the CAESAR slave's RAM.</p>	Default setting: 1
BootNewFirmware	<p>As of CAESAR release 2.8, an alternate new firmware is available with which the GPD runs on the CAESAR master. Up to now, it supports KW2000 protocols. Some of the advantages provided by the new firmware: - Number of available CAN channels: 40 / Piggyback - Increased data throughput, e.g., flash processes are accelerated significantly The CAESAR hardware used must be</p>	Default setting: All

	<p>booted with the new firmware in order to activate this feature. By specifying the hardware separated by commas, the specified hardware is booted with the new firmware.</p> <p>The following setting causes the first and fourth hardware to be booted with the new firmware (to be specified as separate numbers &lt; 100 in the list):  <code>BootNewFirmware=0,3</code></p> <p>If the firmware should be applied to all parts, simply enter "All" in the list.</p> <p>If the old firmware should be used, the list must be empty.</p> <p>The default setting is All.</p>	
<p>BootBusSimFirmware</p>	<p>As of version 2.8.0, CAESAR provides functions for bus simulation. This function can be used in Vediamo through Java routines. In order to use the bus simulation, the respective CAESAR hardware must be booted with a special firmware. In order to selectively boot the available CAESAR hardware with the bus simulation firmware, the key <i>BootBusSimFirmware</i> is inserted:</p> <ul style="list-style-type: none"> <li>- <code>BootBusSimFirmware=ALL</code> Boots all CAESAR "cards" recognized when CAESAR (ComConstruct) starts with the bus simulation firmware.</li> <li>- <code>BootBusSimFirmware=0,2</code> To allow selective booting of individual CAESAR "cards", the following rule is applied: If one or more number less than 100 are specified in the key, they are interpreted as consecutive card numbers. In the above example, the first and third CAESAR card recognized during booting are booted with the bus simulation firmware.</li> </ul> <p>The above rule also applies to the key <i>BootNewFirmware</i> which can be used</p>	<p>Default setting:</p>

	to control the booting with the "new" firmware. In case of conflicting specifications, <i>BootNewFirmware</i> has precedence over <i>BootBusSimFirmware</i> , e.g., if <i>BootNewFirmware</i> =0,1 and <i>BootBusSimFirmware</i> =0,1 the first two cards are booted with the "new" firmware.	
DefaultDeviceNumber	This entry specifies the default device number.	Default setting: 0
DefaultDiagPin	This entry specifies the default Diag Pin.	Default setting: 0
MasterSlaveTimeout	When a communication channel is opened, among other things, a parameter is passed to CAESAR specifying how long the communication between CAESAR master and slave may be inactive before CAESAR generates a timeout error This value (in ms) can be modified here. The default setting is 15000 ms.	Default setting: 15000
MonitoringFilterCANIDFile	This entry specifies the file which contains the filter settings for the trace display.	
MonitoringFilterCANIDs	This entry specifies whether the trace display is filtered (see <i>MonitoringFilterCANIDFile</i> ). The default setting is no.	Default setting: 0
UseDriverTypes	With CAESAR release 2.6, it is possible to include multiple GPD references in the parameterization for communication with the ECU. These GPD references are specified by driver-type codes. This entry determines which driver type code application is preferred, if a default system or an older version of a system description is to be loaded. The following values are valid: - NULL - KLINE - CANLS - CANHS - D2B - J1708 - MOST	

	<p>- J1850  - CCD  - SCIENG  - SCITRANS</p> <p>It is possible to specify more than one value. A sequence can be defined in which the use of certain values is preferred to that of others. The values need to be entered in the above format, separated by commas. Unlisted keywords are ignored. The complete entry is read when the diagnostic server starts. If multiple GPD references are included, the specified sequence of the preferred GPD references is maintained when accessing resources and ECU information.</p> <p>Example :  UseDriverTypes =  KLINE,NULL,CANLS,CANHS,D2B</p>					
UseServiceTypes	<p>This entry makes it possible to prefilter the services which are displayed, e.g., when generating a system description. . If "STANDARD" is set, the following non-executable service types are not displayed  DST_SYSTEM,  DST_ENVIRONMENT_DATA,  DST_GLOBAL, DST_NEGRESP,  DST_BINARY_ACTUATOR_INP,  DST_BINARY_ADJUSTMENT_INP.</p>	<p>Default setting:  <b>STANDARD</b></p> <p>Possible values:</p> <table border="1" data-bbox="1092 1056 1430 1446"> <tr> <td data-bbox="1092 1056 1279 1287">STANDARD</td> <td data-bbox="1279 1056 1430 1287">Certain non-executable services are not displayed</td> </tr> <tr> <td data-bbox="1092 1287 1279 1446">ALL</td> <td data-bbox="1279 1287 1430 1446">All services are displayed</td> </tr> </table>	STANDARD	Certain non-executable services are not displayed	ALL	All services are displayed
STANDARD	Certain non-executable services are not displayed					
ALL	All services are displayed					
TesterPresentInfo	<p>Controls the channel-related, enhanced function for updating the communication status. For certain ECUs / protocols, it may be necessary for the Vediamo diagnostic server to send a message to the ECU cyclically, to update the communication status. The function can be configured using this entry.</p> <p>The following can be specified (in the form of a list):</p>					

	<p>- the exact ID of the protocol under which the feature should be activated  - the message to send as "tester present" request, in hex format</p> <p>Individual entries are separated by commas.</p> <p>If a Part J is used for diagnostics on the respective channel, the feature is categorically deactivated for that channel. The current configuration information is always evaluated (taken from the INI file) after a channel is opened.</p> <p>Example - the settings for 2 protocols are affected:  TesterPresentInfo=KW2C3PE,3e01,KW2C2PE,3e01</p>	
--	---	--

**INTERPRETER**

Name	Description	Comments
ClassPath	This entry specifies the search path where the Java Virtual Machine finds the Java classes. Multiple directories can be separated by semicolons. The default setting is the current working directory (.). The path specified under SystemPfad in the [SERVER] section is also searched.	
VHandlersLib	This entry specifies, which library (DLL) should be used for the Vediamo handler functions in scripts. The default setting is VHandlers.dll.	
JavaInterpreter	This entry specifies the Java interpreter. This is javaw.exe for Sun Java Runtime Environment (JRE) without a console window or java.exe for Sun JRE with a console window. Each file can be found in the respective JRE installation directory.	
UseJVHandlers Package	Vediamo's own Java classes are made available in two ways: In an "unnamed package" or in a "JVHandlers package". The default setting is the "unnamed package".	Default setting: 0

	This entry specifies, whether the "JVHandlers package" should be used.	
<b>PSR</b>		
<b>Name</b>	<b>Description</b>	<b>Comments</b>
COMPORT	This entry specifies which interface should be used for communication with the PSR. Valid values are the numbers 0 to n for communication over the serial interface, where n is the number of available serial interfaces. Values over 1024 are interpreted as the port number for TCP/IP communication. The value 2049 is reserved for TCP/IP communication.	Default setting: 1 Minimum value: 0
EngineTable	This entry specifies which file contains the engine table. A sample file is generated and specified when Vediamo is installed.	
PreloadSystem	In this entry the name of a Vediamo system can be entered. If a system is named, its system file (VSB) will be loaded when PSR Client is started. In this case the first engine test run will not have to wait in the beginning for loading the VSB file. Important: the system name is case-sensitive.	
COMTIMEOUT	This entry specifies the timeout in ms for communication with the test bench. If no communication with the test bench takes place for this period of time, the test run and communication with the ECU are ended, and the transmission buffer is cleared. The value 0 means no timeout (default setting).	Default setting: 0 Minimum value: 0
LOGFILE	This entry specifies in which file the byte-level communication (with timestamp) is logged. The default setting is empty and means that no logging takes place. The entry should be empty under normal operation.	
<b>SYSTEMCONFIGURATION</b>		
<b>Name</b>	<b>Description</b>	<b>Comments</b>
ProgramLogPat	This entry specifies in which directory to	Default setting:

h	save the system configuration tool's log file.	...\Vediamo\Log\SystemConfiguration										
LogWin	This entry specifies whether a window in which the application's actions are logged should be opened when the application starts. The default setting is yes.	Default setting: 1										
OutputLevel	This entry specifies the degree of detail with which CAESAR actions are logged. The messages are saved in: "<[SYSTEMCONFIGURATION]ProgramLogPath>\VediamoSysConfLog.txt". The default setting is module-related.	Default setting: 1 Possible values: <table border="1"> <tr><td>0</td><td>No output</td></tr> <tr><td>1</td><td>Module-related</td></tr> <tr><td>2</td><td>Function-related</td></tr> <tr><td>3</td><td>Internal function details</td></tr> <tr><td>4</td><td>Maximum</td></tr> </table>	0	No output	1	Module-related	2	Function-related	3	Internal function details	4	Maximum
0	No output											
1	Module-related											
2	Function-related											
3	Internal function details											
4	Maximum											
LogFileMaxSize	This entry specifies the maximum size (in bytes) of the VediamoSysConfLog.txt logfile in the system configuration directory. The initial setting is 1000000 (ca. 1 MB). The minimum limit is 10000 bytes.	Default setting: 1000000 Minimum value: 10000										
<b>Worker</b>												
<b>Name</b>	<b>Description</b>	<b>Comments</b>										
Server	The name of the server with which the program was connected last. No automatic connection takes place, but this server name is recommended to the user when the F5 key is pressed. It can be acknowledged or overwritten. This entry is set automatically to the name of the server to which a connection is established.											
LogFile	File in which the log window entries are recorded. If this entry is missing or empty, no data is automatically recorded. This entry is updated automatically when the program is ended in accordance with the settings in the options window.											
Warnings	Specifies whether warnings of unusual	Default setting:										



	conditions (which do not necessarily result in an end to operation) should be displayed in the worker clients status window or not. The default setting is no. This parameter can only be modified directly using the <a href="#">INI-Editor</a> , but not using the worker client options window.	0
<b>PRIVATE</b>		
<b>Name</b>	<b>Description</b>	<b>Comments</b>
IncrementalBackup	This entry specifies whether subsequent status logfile should be appended with a continually incrementing number (Status1.log, Status2.log, ...)	Default setting: 0
TraceHandlerFunctions	Supplementary setting for Java function calls. Simplifies debugging when executing Java routines.	Default setting: 0
TraceThreadIDToFile	Supplementary Thread-ID entry in logfiles. Simplifies user program debugging.	Default setting: 0
<b>UVI</b>		
<b>Name</b>	<b>Description</b>	<b>Comments</b>
SYSTEM	Default system loaded by UVI when the initialization specification does not contain a system name in its parameter list.	

## 6 PSR Messages

From the	To the	Explanation	Code	Data	Comments
PSR	DiagServer	New Engine	3	Engine ID (S)	Each engine is assigned a system file, and optionally, an initialization routine.
DiagServer	PSR	Data loaded	5	System qualifier (S)	System ID (of the VSB file)
DiagServer	PSR	Unknown engine type	9	Error message (S)	Cause of error
PSR	DiagServer	Get CAESAR version	10	-	The version of the c32s.dll is transmitted
DiagServer	PSR	CAESAR Version	11	Version (S)	E.g., "02.07.12"
PSR	DiagServer	Get ID block from ECU	20	-	Contact with ECU must be established
DiagServer	PSR	ID block from ECU	21	ID block (S)	Long string with wrapped lines
DiagServer	PSR	ID block error	22	Error text (S)	
DiagServer	PSR	Error message	50	Message as string (S)	Error in DiagServer, e.g. internal program error, wrong configuration, hardware error
PSR	DiagServer	Establish contact with ECU	100	-	The program starts a cyclical activation process. No reply until contact successfully established
DiagServer	PSR	Contact established	101	ECU name (S)	
DiagServer	PSR	Contact ended	102	ECU name (S)	Is sent when contact is ended, not when contact cannot be

					established. Exception: when no system is loaded, 100 is answered with 102 without an ECU name.
DiagServer	PSR	Measurement information	109	Number(W) Qualifier (S) Unit (empty S)  (after command 113)	A list of all measurements is transmitted, numbered consecutively beginning with 1. The last block with number 0 and empty name means end of list. The units are transmitted as an empty string because they cannot be identified by the CAESAR hardware at this point. However, existing PSR programs require a string parameter.
DiagServer	PSR	Error cleared	110	-	
DiagServer	PSR	Error cannot be cleared	111	Error code (W)	1 = Diagnostic message missing 5 = Error cannot be cleared
DiagServer	PSR	Initialization completed	112	-	
PSR	DiagServer	Transmit measurement qualifier list	113	-	List of all measurement qualifiers (Service 109) is requested. <b>Attention!</b> The list of measurements can be appended by additional settings from the current ECU

					variant after contact has been established.
PSR	DiagServer	Protocol information	150	-	Request ECU protocol ID
DiagServer	PSR	Protocol information	151	Protocol name (S)	E.g., "KWP2000E"
PSR	DiagServer	Send list of measurement names	313	-	List of all measurement names (Service 109) is requested. <b>Attention!</b> The list of measurements can be appended by additional settings from the current ECU variant after contact has been established.
DiagServer	PSR	Throttle valve learned	114	-	
DiagServer	PSR	Throttle valve not learned	115	-	
DiagServer	PSR	Error cannot be read	200	Error text (S)	
DiagServer	PSR	Error	201	Error code (S) Name (S) or (see if 235-238) Status (W) Error code (S) Name (S)	Two empty strings are transmitted if no errors exist, or to mark the end of the error list.
PSR	DiagServer	Clear all errors	202	-	
PSR	DiagServer	Start error read	203	-	Server replies cyclically with 201 blocks. Errors are read from the ECU by default after the first initialization of the ECU during the test

					run.
PSR	DiagServer	End error read	204	-	Can also be sent before initialization with command 100, but only after loading the system with command 3. No errors are read after contact is established in this case.
DiagServer	PSR	Error environment data	211	Error code (S) Environment data (S)	The environment data consists of the name, value and possibly the units, separated by spaces.
PSR	DiagServer	Start error with environment data read	213	-	Server replies cyclically with 201 blocks followed by 211 blocks
PSR	DiagServer	Read error once	230	Optional: Error type can be selected with a WORD (0..4) or one of the following strings: POWERTRAIN (=0) CHASSIS (=1) BODY (=2) NETWORK (=3) UNDEFINED (=4) Default: all (=0xFFFF)	Server replies with 201 blocks
PSR	DiagServer	Read error once, unfiltered	231		Server replies with 201 blocks
PSR	DiagServer	Read error once, with environment data	232		Server replies with 201 blocks followed by 211 blocks
PSR	DiagServer	Read error once, with environment data, unfiltered	233		Server replies with 201 blocks followed by 211 blocks
PSR	DiagServer	Read error once, with status byte	235		Server replies as in 230-233, but the 201 reply blocks begin with a WORD value (error status), followed by P-code
PSR	DiagServer	Read error once, unfiltered,	236		

		with status byte			and error text. The 211 blocks (environment data) remain unchanged.
PSR	DiagServer	Read error once, with environment data, with status byte	237		
PSR	DiagServer	Read error once, with environment data, unfiltered, with status byte	238		
DiagServer	PSR	Cannot read measurements	300	Error text (S)	
DiagServer	PSR	Measurement	301	Number (W) Value (S)	
PSR	DiagServer	Start Transmit data	302	-	DiagServer transmits data (measurements) continuously. If service 304 was not transmitted after the test started, all measurements defined in the VSB file are transmitted.
PSR	DiagServer	End Transmit data	303	-	DiagServer ends the continuous transmission of data
PSR	DiagServer	Select measurement value	304	Measurement numbers (list of words)	If a transmission was ongoing, it is cancelled. Measurements must subsequently be requested again with service 302, or read once with 305.
PSR	DiagServer	Read	305	-	Exactly one read

		measurement once			cycle is executed. The selection with 304 or 306 applies.
PSR	DiagServer	Select measurements and read once	306	Measurement numbers (list of words)	Corresponds to 304, 305. The entered values remain selected until the next 304 or 306 command.
PSR	DiagServer	Start flashing ECU with given flash keys	400	Flashkeys (S) optional: ECU-Qualifier (S)	Several flashkeys can be separated by blank. If the flashed ECU is not the main ECU, its qualifier has to be given as 2nd parameter. The ECU must be initialized and the flashkeys must be valid. The program answers with 401, 405 or with 402
DiagServer	PSR	Flashing started	401	Flashkeys (S)	
DiagServer	PSR	Flashing start failed	402	Flashkeys (S) Error text(S)	
DiagServer	PSR	Flashen finished	405		
PSR	DiagServer	Load Flashware (CFF file)	410	File path (S)	A full path is required. Before loading, previous flashware is unloaded. If an empty path is entered, only unloading is done. The answer is 411 or 412
DiagServer	PSR	Flashware loaded	411	Flashkeys (S)	Reporting of the flashkeys found in the flashware

DiagServer	PSR	Flashware cannot be loaded	412	Dateipfad (S) Error text (S)	
PSR	DiagServer	Load flashware and flash ECU	430	File path (S) optional: ECU-Qualifier (S)	The same action as 410 and 400, but only if the flashware contains one flashkey. Otherwise the program answers with 402
PSR	DiagServer	Show flashing progress	440		
DiagServer	PSR	Flashing progress	441	Number (Word) between 0 and 100	If no flashing in progress, 0 is returned
PSR	DiagServer	Show Flashkeys	450	optional: ECU-Qualifier (S)	Show all flashkeys in the currently loaded flashware
DiagServer	PSR	Flashkeys	451	Flashkeys (S)	All flashkeys separated by a blank
PSR	DiagServer	Activate actuator	500	Name (S) Value (S)	DiagServer does not acknowledge; Discrete actuator: Value must be passed as string, e.g., "FN_ELAB_AUS" (Note: case sensitive!). For continuously adjustable actuators, e.g., the value 0.5 is entered as a null-terminated string "0.5" or "0,5" depending on regional system settings.
DiagServer	PSR	Actuator activated	501	Qualifier (S) Result (S)	If OutputRef available, its result is transmitted



DiagServer	PSR	Cannot activate actuator	502	Qualifier (S) Error text (S)	
PSR	DiagServer	Request actuator list	510		Corresponds to 560 with type "act"
PSR	DiagServer	Request actuator info	520	Qualifier (S)	Corresponds to 561
PSR	DiagServer	Execute any service	550	Qualifier (S) Parameters (S)	Parameters are transmitted in a string, separated by "&"
DiagServer	PSR	Service executed	551	Qualifier (S) Result (S)	If OutputRef available, its result is transmitted
DiagServer	PSR	Error while executing service	552	Qualifier (S) Error text (S)	
PSR	DiagServer	Request list of all services (filtered according to VSB file)	560	Optional: Type (S) <i>act, adj, fun, pro, mea, gen.</i> Default is <i>gen</i>	Type abbreviations correspond to: actuator, adjustment, function, procedure, measurement, generic.  Server replies with 563 blocks.
PSR	DiagServer	Request complete list of all services with number of parameters	561	Optional: Type (S) <i>act, adj, fun, pro, mea, gen.</i> Default is <i>gen</i>	Server replies with 563 blocks.
PSR	DiagServer	Request information on service	562	Qualifier (S)	Server replies with 564 blocks.
DiagServer	PSR	List of all services of the specified type	563	Qualifier (S) Number of parameters (S)	For requests with 560: Qualifier only. For requests with 561: Qualifier and number of parameters.

					End of list: Empty string
DiagServer	PSR	Information on service	564	Info text (S)	Multiple blocks per service The last block contains an empty string
DiagServer	PSR	Problems during information on service	565	Error message (S)	
PSR	DiagServer	Send API-1 message	570	Byte array	input payload only. Header and Checksum are calculated by the program.
DiagServer	PSR	ECU answer to API-1 message	571	Byte array	Payload only.
DiagServer	PSR	Sending failed	572	Error text (S)	
PSR	DiagServer	Send API-1 message	580	Byte array as string (S)	Two-digit hex numbers, separated by blank
DiagServer	PSR	ECU answer to API-1 message	581	Byte array as string (S)	
PSR	DiagServer	Start Java routine	600	Name (S), Optional parameter (S)	It is not possible to start multiple routines at once, error message occurs (service 602).
DiagServer	PSR	Java routine is started	601	Name (S)	Output when available routine is started
DiagServer	PSR	Result, or error, from Java routine	602	Result (W)	0 = routine executed correctly 1 = routine not

					available 2 = routine erroneous 3 = routine currently running 4 = routine cannot be started 5 = Other error
PSR	DiagServer	End test run	700	-	Result is "Test run ended"
DiagServer	PSR	Test run ended	701	-	
PSR	DiagServer	End program	702	-	PSR adapter is deactivated. DiagServer keeps running as long as client applications access it. (Use not recommended, only implemented for compatibility reasons)
PSR	DiagServer	Server Reset	710	-	CAESAR, CAESAR files, system descriptions, engine table are reset.
DiagServer	PSR	Server Reset complete	711	-	Reply to 710
PSR	DiagServer	Server initialized again	720	-	CAESAR, CAESAR files, system descriptions, engine table are reloaded.
DiagServer	PSR	Server initialization complete	721	-	Reply to 720
DiagServer	PSR	Service not recognized	800	ServiceID (Word)	
DiagServer	PSR	Format error in service parameter	801	-	Parameter format is wrong (e.g., string ist not null-terminated)

DiagServer	PSR	Format error	803	-	Length of received message is wrong (e.g., smaller than 6 bytes. or length specification does not match length of message)
PSR	DiagServer	Log start/stop	1000	Filename (S)	Empty name: Stop log. Only works with 3964R, not with TCP/IP
PSR	DiagServer	Display engine list	1100		
DiagServer	PSR	Engine list	1101	Engine name (S)	End of list: Empty string
PSR	DiagServer	Display Java routine list	1200		
DiagServer	PSR	Java routine list	1201	routine ID (S)	End of list: Empty string
<b><i>Up to 4999 - reserved for future use</i></b>					
DiagServer	PSR	Upgrade service	5000 - 30000	none, Word or ASCII string	Available for user-defined Java routines. See also Table 6 in Attachment 5.4

## 7 Example: Java Routine

---

```
/**
 * Title: Example Java Routine
 * Description: Example of a test routine that is started from Ecoute
 */

import JVHandlers.JVECU;
import JVHandlers.JVSystem;
import JVHandlers.JVClient;
import JVHandlers.JVErrorService;
import JVHandlers.JVUtil;

public class Example {
    JVClient client = new JVClient();
    JVSystem system = new JVSystem();
    JVErrorService errorS;
    String[] ecus;
    JVECU ecu;
    //***** The following constants must be entered
    explicitly (depending on ECU).
    *****
    final String solleECU="SolleECU",

functionQualifier="FunctionQualifier",procedureQualifier="ProcedureQual
ifier",

adjustmentQualifier="AdjustmentQualifier",adjSetValue="AdjustmentValue"
;
    final String[] measQualifier={"meas_1","meas_2","meas_3"};
    //*****
    *****
    String measResult, funcResult, adjResult, adjGetValue, procResult;

    public Example() {
    // get the ECUs available in the system from the loaded VSB
    ecus = system.GetEcuQualifiers();

    // the program works with the desired ECU
    for (int i = 0; i < ecus.length; i++)
    {
    if (ecus[i].equalsIgnoreCase(solleECU)){
    ecu = new JVECU(ecus[i]);
    break;
    }
    }
    if(ecu==null){
    errorMessage("System does not contain desired ECU " + solleECU +
"! ",true);
    // "true" means that this error ends the program
    // the program would continue to run with "false"
    }
}
```

```

    client.Trace("ECU " + solleECU + " selected"); // output to Ecoute
status window
    client.Trace("Establish contact..."); // establish contact with ECU
solleECU...
    if (!ecu.Init()) {
        errorMessage("Unable to establish contact to "+ solleECU + ".",true);
    }
    client.Trace("Contact established");

// Measurements are read and displayed...
client.Trace("Read measurements...");

for (int i = 0; i < measQualifier.length; i++) {
    measResult = ecu.GetMeasurementValue(measQualifier[i]);
    if (ecu.IsErrorSet()) {
        errorMessage(measQualifier[i]+" read failed! -
"+ecu.GetLastError(),false);
    }
    else {
        client.Trace(measQualifier[i]+": "+measResult);
    }
}

// Execute function...
client.Trace("Function " + functionQualifier + " being executed...");
funcResult = ecu.ExecuteFunction(functionQualifier);
if (ecu.IsErrorSet()) {
    errorMessage("Execution of function " + functionQualifier + " failed!
- "+ecu.GetLastError(),false);
}
else {
    client.Trace(functionQualifier+": "+funcResult);
}

// Execute procedure...
client.Trace("Procedure " + procedureQualifier + " being
executed...");
procResult = ecu.ExecuteProcedure(procedureQualifier);
if (ecu.IsErrorSet()) {
    errorMessage("Execution of procedure " + procedureQualifier + "
failed! - "+ecu.GetLastError(),false);
}
else {
    client.Trace(procedureQualifier+": "+procResult);
}

// Set adjustment...
client.Trace("Adjustment " + adjustmentQualifier + " being set to " +
adjSetValue + " ...");
adjResult = ecu.SetAdjustmentValue(adjustmentQualifier,adjSetValue);
if (!adjResult) {
    errorMessage("Setting of adjustment" + adjustmentQualifier + " failed!

```

```

- "+ecu.GetLastError(),true);
}
else {
client.Trace("Setting of adjustment " + adjustmentQualifier + " was
successful.");
}

// Read adjustment value...
client.Trace("Adjustment " + adjustmentQualifier + " being read...");
adjGetValue = ecu.GetAdjustmentValue(adjustmentQualifier);
if (ecu.IsErrorSet()) {
errorMessage("Reading of adjustment" + adjustmentQualifier + " failed!
- "+ecu.GetLastError(),true);
}
else {
client.Trace("Adjustment " + adjustmentQualifier + " is set to " +
adjGetValue + " .");
}

// Clear all errors in ECU memory...
client.Trace("Clear all errors...");
errorS = new JVErrorService(ecu);
if (!errorS.ClearAllErrors()) {
errorMessage("Clear all errors failed!",false);
}
client.Trace("All errors cleared.");

// disconnect and end Java client
JVUtil.DisconnectFromServer();
System.exit(0);
}

public static void main(String[] args) {
Example Ex_1 = new Example();
}

public void errorMessage(String text, boolean finish) {
client.Beep();
client.Information(text);
client.Trace(text);
if (finish){
JVUtil.DisconnectFromServer();
System.exit(1);
}
}

} // class Example

```