CAN

Automotive Diagnostic Command Set User Manual

December 2009
372139D-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

CVI, National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on `ni.com/legal` for more information about National Instruments trademarks.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries. Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Compliance

Electromagnetic Compatibility Information

This hardware has been tested and found to comply with the applicable regulatory requirements and limits for electromagnetic compatibility (EMC) as indicated in the hardware's Declaration of Conformity (DoC)[1]. These requirements and limits are designed to provide reasonable protection against harmful interference when the hardware is operated in the intended electromagnetic environment. In special cases, for example when either highly sensitive or noisy hardware is being used in close proximity, additional mitigation measures may have to be employed to minimize the potential for electromagnetic interference.

While this hardware is compliant with the applicable regulatory EMC requirements, there is no guarantee that interference will not occur in a particular installation. To minimize the potential for the hardware to cause interference to radio and television reception or to experience unacceptable performance degradation, install and use this hardware in strict accordance with the instructions in the hardware documentation and the DoC[1].

If this hardware does cause interference with licensed radio communications services or other nearby electronics, which can be determined by turning the hardware off and on, you are encouraged to try to correct the interference by one or more of the following measures:

- Reorient the antenna of the receiver (the device suffering interference).
- Relocate the transmitter (the device generating interference) with respect to the receiver.
- Plug the transmitter into a different outlet so that the transmitter and the receiver are on different branch circuits.

Some hardware may require the use of a metal, shielded enclosure (windowless version) to meet the EMC requirements for special EMC environments such as, for marine use or in heavy industrial areas. Refer to the hardware's user documentation and the DoC[1] for product installation requirements.

When the hardware is connected to a test object or to test leads, the system may become more sensitive to disturbances or may cause interference in the local electromagnetic environment.

Operation of this hardware in a residential area is likely to cause harmful interference. Users are required to correct the interference at their own expense or cease operation of the hardware.

Changes or modifications not expressly approved by National Instruments could void the user's right to operate the hardware under the local regulatory rules.

[1] The Declaration of Conformity (DoC) contains important EMC compliance information and instructions for the user or installer. To obtain the DoC for this product, visit `ni.com/certification`, search by model number or product line, and click the appropriate link in the Certification column.

Contents

About This Manual

Chapter 1
Introduction

Chapter 2
Installation and Configuration

Chapter 3
Application Development

Chapter 4
Using the Automotive Diagnostic Command Set

Chapter 5
Automotive Diagnostic Command Set API for LabVIEW

Chapter 6
Automotive Diagnostic Command Set API for C

About This Manual

This manual provides instructions for using the Automotive Diagnostic Command Set. It contains information about installation, configuration, and troubleshooting, and also contains Automotive Diagnostic Command Set function reference for LabVIEW-based and C-based APIs.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *ANSI/ISO Standard 11898-1993, Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*
- *CAN Specification Version 2.0*, 1991, Robert Bosch GmbH., Postfach 106050, D-70049 Stuttgart 1
- *CiA Draft Standard 102, Version 2.0*, CAN Physical Layer for Industrial Applications
- *ISO 14229:1998(E), Road Vehicles, Diagnostic Systems, Diagnostic Services Specification*
- *ISO 14230-1:1999(E), Road Vehicles, Diagnostic Systems, Keyword Protocol 2000, Part 1: Physical Layer*
- I*SO 14230-2:1999(E), Road Vehicles, Diagnostic Systems, Keyword Protocol 2000, Part 2: Data Link Layer*
- *ISO 14230-3:1999(E), Road Vehicles, Diagnostic Systems, Keyword Protocol 2000, Part 3: Application Layer*
- *ISO 15765-1:2004(E), Road Vehicles, Diagnostics on Controller Area Networks (CAN), Part 1: General Information*
- *ISO 15765-2:2004(E), Road Vehicles, Diagnostics on Controller Area Networks (CAN), Part 2: Network Layer Services*
- *ISO 15765-3:2004(E), Road Vehicles, Diagnostics on Controller Area Networks (CAN), Part 3: Implementation of Unified Diagnostic Services (UDS on CAN)*
- *NI-CAN Hardware and Software Manual*

1 Introduction

Diagnostics involve remote execution of routines, or services, on ECUs. To execute a routine, you send a byte string as a request to an ECU, and the ECU usually answers with a response byte string. Several diagnostic protocols such as KWP2000 and UDS standardize the format of the services to be executed, but those standards leave a large amount of room for manufacturer-specific extensions. A newer trend is the emission-related legislated OnBoard Diagnostics (OBD), which is manufacturer independent and standardized in SAE J1979 and ISO 15031-5. This standard adds another set of services that follow the same scheme.

Because diagnostics were traditionally executed on serial communication links, the byte string length is not limited. For newer, CAN-based diagnostics, this involves using a transport protocol that segments the arbitrarily long byte strings into pieces that can be transferred over the CAN bus, and reassembles them on the receiver side. Several transport protocols accomplish this task. The Automotive Diagnostic Command Set implements the ISO TP (standardized in ISO 15765-2) and the manufacturer-specific VW TP 2.0.

Note The Automotive Diagnostic Command Set is designed for CAN-based diagnostics only. Diagnostics on serial lines (K-line and L-line) are not in the scope of the Automotive Diagnostic Command Set.

KWP2000 (Key Word Protocol 2000)

The KWP2000 protocol has become a de facto standard in automotive diagnostic applications. It is standardized as ISO 14230-3. KWP2000 describes the implementation of various diagnostic services you can access through the protocol. You can run KWP2000 on several transport layers such as K-line (serial) or CAN.

Transport Protocol

As KWP2000 uses messages of variable byte lengths, a transport protocol is necessary on layers with only a well defined (short) message length, such as CAN. The transport protocol splits a long KWP2000 message into pieces that can be transferred over the network and reassembles those pieces to recover the original message.

KWP2000 runs on CAN on various transport protocols such as ISO TP (ISO 15765-2), TP 1.6, TP 2.0 (Volkswagen), and SAE J1939-21.

Note For KWP2000, the Automotive Diagnostic Command Set supports only the ISO TP (standardized in ISO 15765-2) and manufacturer-specific VW TP 2.0 transport protocols.

Diagnostic Services

The diagnostic services available in KWP2000 are grouped in functional units and identified by a one-byte code (ServiceId). The standard does not define all codes; for some codes, the standard refers to other SAE or ISO standards, and some are reserved for manufacturer-specific extensions. The Automotive Diagnostic Command Set supports the following services:

- Diagnostic Management
- Data Transmission
- Stored Data Transmission (Diagnostic Trouble Codes)
- Input/Output Control
- Remote Activation of Routine

Note Upload/Download and Extended services are not part of the Automotive Diagnostic Command Set.

Diagnostic Service Format

Diagnostic services have a common message format. Each service defines a Request Message, Positive Response Message, and Negative Response Message.

The Request Message has the ServiceId as first byte, plus additional service-defined parameters. The Positive Response Message has an echo of the ServiceId with bit 6 set as first byte, plus the service-defined response parameters.

The Negative Response Message is usually a three-byte message: it has the Negative Response ServiceId as first byte, an echo of the original ServiceId

as second byte, and a ResponseCode as third byte. The only exception to this format is the negative response to an EscapeCode service; here, the third byte is an echo of the user-defined service code, and the fourth byte is the ResponseCode. The KWP2000 standard partly defines the ResponseCodes, but there is room left for manufacturer-specific extensions. For some of the ResponseCodes, KWP2000 defines an error handling procedure. Because both positive and negative responses have an echo of the requested service, you can always assign the responses to their corresponding request.

Connect/Disconnect

KWP2000 expects a diagnostic session to be started with StartDiagnosticSession and terminated with StopDiagnosticSession. However, StartDiagnosticSession has a DiagnosticMode parameter that determines the diagnostic session type. Depending on this type, the ECU may or may not support other diagnostic services, or operate in a restricted mode where not all ECU functions are available. The DiagnosticMode parameter values are manufacturer specific and not defined in the standard.

For a diagnostic session to remain active, it must execute the TesterPresent service periodically if no other service is executed. If the TesterPresent service is missing for a certain period of time, the diagnostic session is terminated, and the ECU returns to normal operation mode.

GetSeed/Unlock

A GetSeed/Unlock mechanism may protect some diagnostic services. However, the applicable services are left to the manufacturer and not defined by the standard.

You can execute the GetSeed/Unlock mechanism through the SecurityAccess service. This defines several levels of security, but the manufacturer assigns these levels to certain services.

Read/Write Memory

Use the Read/WriteMemoryByAddress services to upload/download data to certain memory addresses on an ECU. The address is a three-byte quantity in KWP2000 and a five-byte quantity (four-byte address and one-byte extension) in the calibration protocols.

The Upload/Download functional unit services are highly manufacturer specific and not well defined in the standard, so they are not a good way to provide a general upload/download mechanism.

Measurements

Use the ReadDataByLocal/CommonIdentifier services to access ECU data in a way similar to a DAQ list. A Local/CommonIdentifier describes a list of ECU quantities that are then transferred from the ECU to the tester. The transfer can be either single value or periodic, with a slow, medium, or fast transfer rate. The transfer rates are manufacturer specific; you can use the SetDataRates service to set them, but this setting is manufacturer specific.

Note The Automotive Diagnostic Command Set supports single-point measurements.

Diagnostic Trouble Codes

A major diagnostic feature is the readout of Diagnostic Trouble Codes (DTCs). KWP2000 defines several services that access DTCs based on their group or status.

Input/Output Control

KWP2000 defines services to modify internal or external ECU signals. One example is redirecting ECU sensor inputs to stimulated signals. The control parameters of these commands are manufacturer specific and not defined in the standard.

Remote Activation of a Routine

These services are similar to the ActionService and DiagService functions of CCP. You can invoke an ECU internal routine identified by a Local/CommonIdentifier or a memory address. Contrary to the CCP case, execution of this routine can be asynchronous; that is, there are separate Start, Stop, and RequestResult services.

The control parameters of these commands are manufacturer specific and not defined in the standard.

External References

For more information about the KWP2000 Standard, refer to the ISO 14230-3 standard.

UDS (Unified Diagnostic Services)

The UDS protocol has become a de facto standard in automotive diagnostic applications. It is standardized as ISO 15765-3. UDS describes the implementation of various diagnostic services you can access through the protocol.

As UDS uses messages of variable byte lengths, a transport protocol is necessary on layers with only a well defined (short) message length, such as CAN. The transport protocol splits a long UDS message into pieces that can be transferred over the network and reassembles those pieces to recover the original message.

UDS runs on CAN on various transport protocols.

Note The Automotive Diagnostic Command Set supports only the ISO TP (standardized in ISO 15765-2) and manufacturer-specific VW TP 2.0 transport protocols.

Diagnostic Services

The diagnostic services available in UDS are grouped in functional units and identified by a one-byte code (ServiceId). Not all codes are defined in the standard; for some codes, the standard refers to other standards, and some are reserved for manufacturer-specific extensions. The Automotive Diagnostic Command Set supports the following services:

- Diagnostic Management
- Data Transmission
- Stored Data Transmission (Diagnostic Trouble Codes)
- Input/Output Control
- Remote Activation of Routine

Diagnostic Service Format

Diagnostic services have a common message format. Each service defines a Request Message, a Positive Response Message, and a Negative Response Message. The general format of the diagnostic services complies with the KWP2000 definition; most of the Service Ids also comply with KWP2000. The Request Message has the ServiceId as first byte, plus additional service-defined parameters. The Positive Response Message has an echo of the ServiceId with bit 6 set as first byte, plus the service-defined response parameters.

Note Some parameters to both the Request and Positive Response Messages are optional. Each service defines these parameters. Also, the standard does not define all parameters.

The Negative Response Message is usually a three-byte message: it has the Negative Response ServiceId (0x7F) as first byte, an echo of the original ServiceId as second byte, and a ResponseCode as third byte. The UDS standard partly defines the ResponseCodes, but there is room left for manufacturer-specific extensions. For some of the ResponseCodes, UDS defines an error handling procedure.

Because both positive and negative responses have an echo of the requested service, you always can assign the responses to their corresponding request.

External References

For more information about the UDS Standard, refer to the ISO 15765-3 standard.

OBD (On-Board Diagnostic)

On-Board Diagnostic (OBD) systems are present in most cars and light trucks on the road today. On-Board Diagnostics refer to the vehicle's self-diagnostic and reporting capability, which the vehicle owner or a repair technician can use to query status information for various vehicle subsystems.

The amount of diagnostic information available via OBD has increased since the introduction of on-board vehicle computers in the early 1980s. Modern OBD implementations use a CAN communication port to provide real-time data and a standardized series of diagnostic trouble codes (DTCs), which identify and remedy malfunctions within the vehicle. In the 1970s and early 1980s, manufacturers began using electronic means to control engine functions and diagnose engine problems. This was primarily to meet EPA emission standards. Through the years, on-board diagnostic systems have become more sophisticated. OBD-II, a new standard introduced in the mid 1990s, provides almost complete engine control and also monitors parts of the chassis, body, and accessory devices, as well as the car's diagnostic control network.

The On-Board Diagnostic (OBD) standard defines a minimum set of diagnostic information for passenger cars and light and medium-duty trucks, which must be exchanged with any off-board test equipment.

2

Installation and Configuration

This chapter explains how to install and configure the Automotive Diagnostic Command Set.

Installation

This section discusses the Automotive Diagnostic Command Set installation for Microsoft Windows.

Note You need administrator rights to install the Automotive Diagnostic Command Set on your computer.

Follow these steps to install the Automotive Diagnostic Command Set software:

1. Insert the Automotive Diagnostic Command Set CD into the CD-ROM drive.
2. Open Windows Explorer.
3. Access the CD-ROM drive.
4. Double-click on `autorun.exe` to launch the software interface.
5. Start the installation. The installation program guides you through the rest of the installation process.
6. If you have not already installed NI-CAN, the Automotive Diagnostic Command Set installer automatically installs the NI-CAN driver on your computer.

   Within the **Devices & Interfaces** branch of the MAX Configuration tree, NI CAN hardware is listed along with other hardware in the local computer system. If the CAN hardware is not listed here, MAX is not configured to search for new devices on startup. To search for the new hardware, press <F5>. To verify installation of the CAN hardware, right-click the CAN device, then select **Self-test**. If the self-test passes, the card icon shows a checkmark. If the self-test fails, the card icon shows an X mark, and the **Test Status** in the right pane describes the problem.

Refer to Appendix A, *Troubleshooting and Common Questions*, of the *NI-CAN User Manual* for information about resolving hardware installation problems.

If you are using the Automotive Diagnostic Command Set on an NI-XNET device, install the NI-XNET driver 1.0 or higher, NI-CAN 2.7 or higher, and the NI-CAN Compatibility Library on your computer.

The MAX Configuration tree **Devices and Interfaces** branch lists NI-XNET hardware (along with other local computer system hardware). If the NI-XNET hardware is not listed there, MAX is not configured to search for new devices on startup. To search for the new hardware, press <F5>. To verify CAN hardware installation, right-click the CAN device and select **Self-Test**. If the self-test passes, the card icon shows a checkmark. If the self-test fails, the card icon shows an X mark, and the **Test Status** in the right pane describes the problem. Refer to Chapter 6, *Troubleshooting and Common Questions,* of the *NI-XNET User Manual* for information about resolving hardware installation problems. The NI-XNET CAN hardware interfaces are listed under the device name. To change the interface name, select a new one from the **Interface Name** box in the middle pane.

When installation is complete, you can access the Automotive Diagnostic Command Set functions in your application development environment.

LabVIEW Real-Time (RT) Configuration

LabVIEW Real-Time (RT) combines easy-to-use LabVIEW programming with the power of real-time systems. When you use a National Instruments PXI controller as a LabVIEW RT system, you can install a PXI CAN card and use the NI-CAN or NI-XNET APIs to develop real-time applications. As with any NI software library for LabVIEW RT, you must install the Automotive Diagnostic Command Set software to the LabVIEW RT target using the Remote Systems branch in MAX. For more information, refer to the LabVIEW RT documentation.

After you install the PXI CAN cards and download the Automotive Diagnostic Command Set software to the LabVIEW RT system, you must verify the installation.

Hardware and Software Requirements

You can use the Automotive Diagnostic Command Set on the following hardware:

- National Instruments NI-CAN hardware Series 1 or 2 with the NI-CAN driver software version 2.3 or later installed.
- National Instruments NI-XNET hardware with the NI-XNET driver software version 1.0 or later installed.
- National Instruments CompactRIO or R Series Multifunction RIO hardware and the NI 9853 or NI 9852 CompactRIO CAN modules.

Note You can use the Automotive Diagnostic Command Set with LabVIEW 2009 or newer on CompactRIO systems or National Instruments R Series Multifunction RIO hardware.

3

Application Development

This chapter explains how to develop an application using the Automotive Diagnostic Command Set API.

Choosing the Programming Language

The programming language you use for application development determines how to access the Automotive Diagnostic Command Set APIs.

LabVIEW

Automotive Diagnostic Command Set functions and controls are in the LabVIEW palettes. In LabVIEW, the Automotive Diagnostic Command Set palette is in the top-level NI Measurements palette.

Chapter 5, *Automotive Diagnostic Command Set API for LabVIEW*, describes each LabVIEW VI for the Automotive Diagnostic Command Set API.

To access the VI reference from within LabVIEW, press <Ctrl-H> to open the Help window, click the appropriate Automotive Diagnostic Command Set VI, and follow the link. The Automotive Diagnostic Command Set software includes a full set of LabVIEW examples. These examples teach programming basics as well as advanced topics. The example help describes each example and includes a link you can use to open the VI.

LabWindows/CVI

Within LabWindows™/CVI™, the Automotive Diagnostic Command Set function panel is in **Libraries»Automotive Diagnostic Command Set**. As with other LabWindows/CVI function panels, the Automotive Diagnostic Command Set function panel provides help for each function and the ability to generate code. Chapter 6, *Automotive Diagnostic Command Set API for C*, describes each Automotive Diagnostic Command Set API function. You can access the reference for each function directly from within the function panel. The Automotive Diagnostic Command Set API header file is `nidiagcs.h`. The Automotive Diagnostic Command Set API library is `nidiagcs.lib`. The toolkit software includes a full set of

LabWindows/CVI examples. The examples are in the LabWindows/CVI `\samples\Automotive Diagnostic Command Set` directory. Each example includes a complete LabWindows/CVI project (`.prj` file). The example description is in comments at the top of the `.c` file.

Visual C++ 6

The Automotive Diagnostic Command Set software supports Microsoft Visual C/C++ 6.

The header file for Visual C/C++ 6 is in the `Program Files\National Instruments\Shared\ExternalCompilerSupport\C\include` folder. To use the Automotive Diagnostic Command Set API, include the `nidiagcs.h` header file in the code, then link with the `nidiagcs.lib` library file. The library file is in the `Program Files\National Instruments\Shared\ExternalCompilerSupport\C\lib32\msvc` folder.

For C applications (files with a `.c` extension), include the header file by adding a `#include` to the beginning of the code, as follows:

```
#include "nidiagcs.h"
```

For C++ applications (files with a `.cpp` extension), define `__cplusplus` before including the header, as follows:

```
#define __cplusplus
#include "nidiagcs.h"
```

The `__cplusplus` define enables the transition from C++ to the C language functions.

Chapter 6, *Automotive Diagnostic Command Set API for C*, describes each function.

On Windows Vista (with Standard User Account), the typical path to the C examples folder is `\Users\Public\Documents\National Instruments\Automotive Diagnostic Command Set\Examples\MS Visual C`.

On Windows XP/2000, the typical path to the C examples folder is `\Documents and Settings\All Users\Documents\National Instruments\Automotive Diagnostic Command Set\Examples\MS Visual C`.

Each example is in a separate folder. The example description is in comments at the top of the `.c` file. At the command prompt, after setting MSVC environment variables (such as with MS `vcvars32.bat`), you can build each example using a command such as:

```
cl /I<HDir> GetDTCs.c <LibDir>\nidiagcs.lib
```

<HDir> is the folder where `nidiagcs.h` can be found.

<LibDir> is the folder where `nidiagcs.lib` can be found.

Other Programming Languages

The Automotive Diagnostic Command Set software does not provide formal support for programming languages other than those described in the preceding sections. If the programming language includes a mechanism to call a Dynamic Link Library (DLL), you can create code to call Automotive Diagnostic Command Set functions. All functions for the Automotive Diagnostic Command Set API are in `nidiagcs.dll`. If the programming language supports the Microsoft Win32 APIs, you can load pointers to Automotive Diagnostic Command Set functions in the application. The following section describes how to use the Win32 functions for C/C++ environments other than Visual C/C++ 6. For more detailed information, refer to Microsoft documentation.

The following C language code fragment shows how to call Win32 `LoadLibrary` to load the Automotive Diagnostic Command Set API DLL:

```
#include <windows.h>
#include "nidiagcs.h"
HINSTANCE NiDiagCSLib = NULL;
NiMcLib = LoadLibrary("nidiagcs.dll");
```

Next, the application must call the Win32 `GetProcAddress` function to obtain a pointer to each Automotive Diagnostic Command Set function the application uses. For each function, you must declare a pointer variable using the prototype of the function. For the Automotive Diagnostic Command Set function prototypes, refer to Chapter 6, *Automotive Diagnostic Command Set API for C*. Before exiting the application, you must unload the Automotive Diagnostic Command Set DLL as follows:

```
FreeLibrary (NiDiagCSLib);
```

Application Development on CompactRIO or R Series

To run a project on an FPGA target, you need an FPGA bitfile (`.lvbitx`). The FPGA bitfile is downloaded to the FPGA target on the execution host. A bitfile is a compiled version of an FPGA VI. FPGA VIs, and thus bitfiles, define the CAN, analog, digital, and pulse width modulation (PWM) inputs and outputs of an FPGA target. The Automotive Diagnostic Command Set does not include FPGA bitfiles for any FPGA target. Refer to the LabVIEW FPGA Module documentation for more information about creating FPGA VIs and bitfiles for an FPGA target.

The default FPGA VI is sufficient for a basic Automotive Diagnostic Command Set application. However, in some situations you may need to modify the existing FPGA code to create a custom bitfile. For example, to use additional I/O on the FPGA target, you must add these I/O to the FPGA VI. You must install the LabVIEW FPGA Module to create these files.

Modify the FPGA VI according to the following guidelines:

- Do not modify, remove, or rename any block diagram controls and indicators named __CAN Rx Data, __CAN Rx Ready, __CAN Tx Data Frame, __CAN Tx Ready, __CAN Bit Timing, __CAN_FPGA Is Running, __CAN Start, __FIFO Full, or __CAN FIFO Empty.
- Do not modify the CAN read and write code except to filter CAN IDs on the receiving side to minimize the amount of CAN data transfers to the host.
- As you create controls or indicators, ensure that each control name is unique within the VI.

Refer to the LabVIEW FPGA Module documentation for more information about creating FPGA VIs and bitfiles for an FPGA target.

Transferring Data between the FPGA and Host Computer

While you are creating or modifying the FPGA VI, it is important to know how the Automotive Diagnostic Command Set transfers CAN data to and from the host computer. The Automotive Diagnostic Command Set transfers data via unique named controls and indicators between the host computer and FPGA target.

Customizing the CAN Bridge (FPGA) VI

All examples for CompactRIO or R Series targets are based on the CAN Bridge (FPGA) VI. This VI ensures that the CAN frames are received from the CAN module and transferred to the Automotive Diagnostic Command Set on the host. After the Automotive Diagnostic Command Set has processed the received frame information, a CAN frame response may be transferred to the CAN Bridge (FPGA) VI. Therefore, changing the code of the CAN receive or transmit part of the CAN Bridge (FPGA) VI may result in communication problems. Be careful when changing this code.

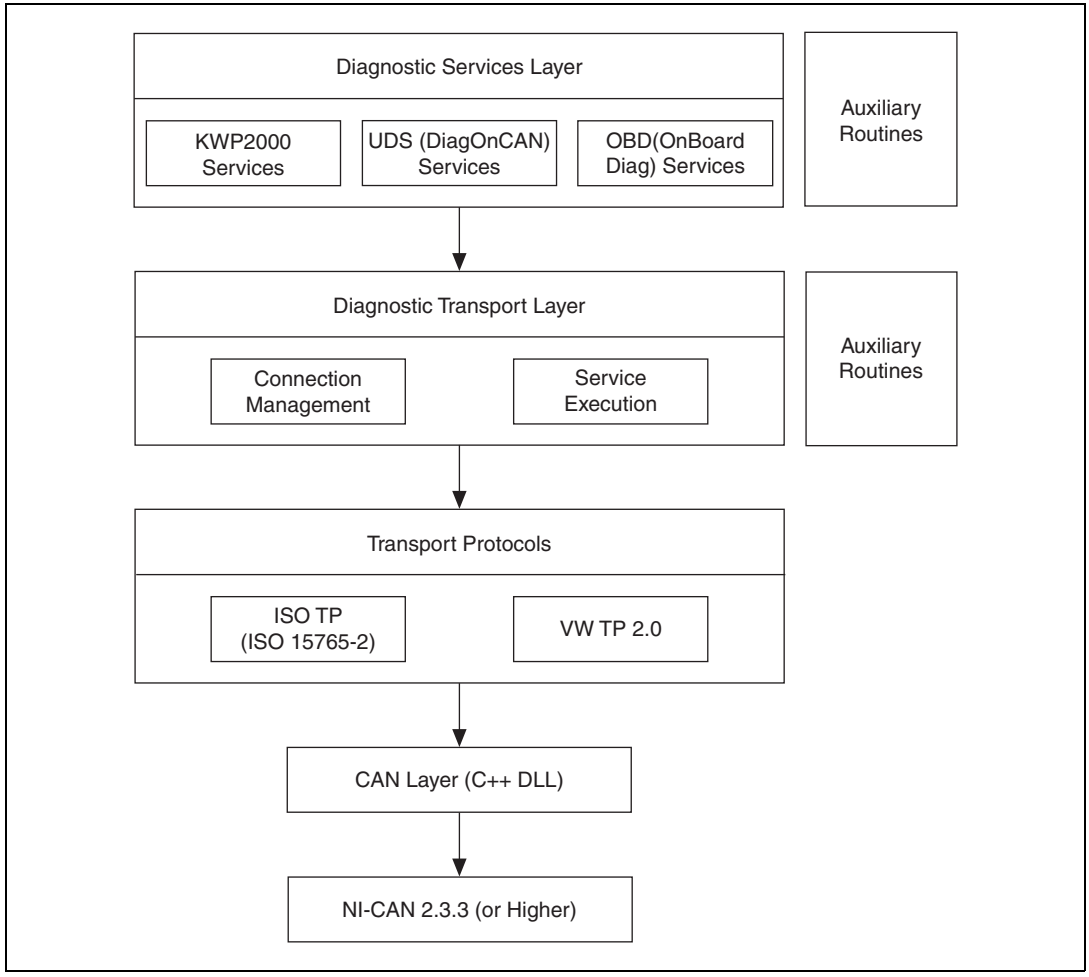
Debugging an Application

To debug your diagnostic application, use the LabVIEW example **Diagnostic Monitor.vi**. This example monitors the CAN traffic the diagnostic protocols generate on the level of individual CAN messages. It works with all other Automotive Diagnostic Command Set examples and diagnostic applications using the Automotive Diagnostic Command Set. To launch this tool, open the LabVIEW Example Finder and search for **Diagnostic Monitor.vi** under **Hardware Input and Output/CAN/ Automotive Diagnostic Command Set/Diagnostic Monitor**.

4

Using the Automotive Diagnostic Command Set

Structure of the Automotive Diagnostic Command Set



The Automotive Diagnostic Command Set is structured into three layers of functionality:

- The top layer implements three sets of diagnostic services for the diagnostic protocols KWP2000, UDS (DiagOnCAN), and OBD (On-Board Diagnostics).
- The second layer implements general routines involving opening and closing diagnostic communication connections, connecting and disconnecting to/from an ECU, and executing a diagnostic service on byte level. The latter routine is the one the top layer uses heavily.
- The third layer implements the transport protocols needed for diagnostic communication to an ECU. The second layer uses these routines to communicate to an ECU.

All three top layers are fully implemented in LabVIEW.

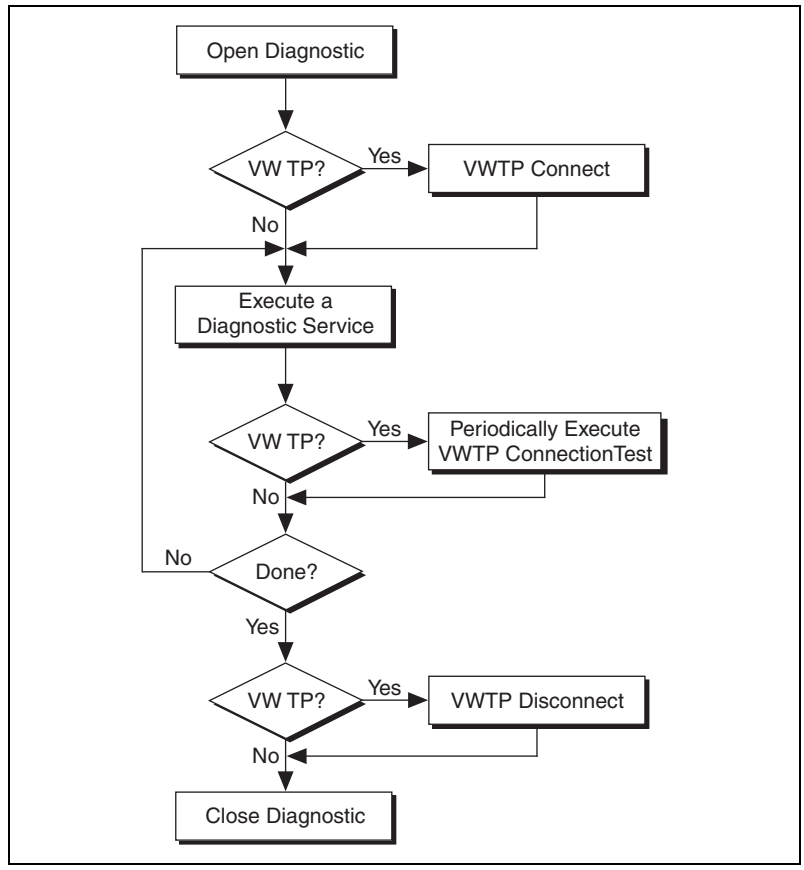
The transport protocols then execute CAN Read/Write operations through a specialized DLL for streamlining the CAN data flow, especially in higher busload situations.

Automotive Diagnostic Command Set API Structure

The top two layer routines are available as API functions. Each diagnostic service for KWP2000, UDS, and OBD is available as one routine. Also available on the top level are auxiliary routines for converting scaled physical data values to and from their binary representations used in the diagnostic services.

On the second layer are more general routines for opening and closing diagnostic communication channels and executing a diagnostic service. Auxiliary routines create the diagnostic CAN identifiers from the logical ECU address.

General Programming Model



First, you must open a diagnostic communication link. This involves initializing the CAN port and defining communication parameters such as the baud rate and CAN identifiers on which the diagnostic communication takes place. No actual communication to the ECU takes place at this stage.

For the VW TP 2.0, you then must establish a communication channel to the ECU using the VWTP Connect routine. The communication channel properties are negotiated between the host and ECU.

After these steps, the diagnostic communication is established, and you can execute diagnostic services of your choice. Note that for the VW TP 2.0, you must execute the VWTP ConnectionTest routine periodically (once per second) to keep the communication channel open.

When you finish your diagnostic services, you must close the diagnostic communication link. This finally closes the CAN port. For the VW TP 2.0, you should disconnect the communication channel established before closing.

Available Diagnostic Services

The standards on automotive diagnostic define many different services for many purposes. Unfortunately, most services leave a large amount of room for manufacturer-specific variants and extensions. National Instruments implemented the most used variants while trying not to overload them with optional parameters.

However, all services are implemented in LabVIEW and open to the user. If you are missing a service or variant of an existing service, you can easily add or modify it on your own.

In the C API, you can also implement your own diagnostic services using the ndDiagnosticService routine. However, the templates from the existing services are not available.

Tweaking the Transport Protocol

A set of global constants controls transport protocol behavior. These constants default to maximum performance. To check the properties of an implementation of a transport protocol in an ECU, for example, you may want to change the constants to nonstandard values using the Get/Set Property routines.

The transport protocols also are fully implemented in LabVIEW and open to the user. In LabVIEW, you can even modify the protocol behavior (for example, you can send undefined responses to check the behavior of an implementation).

However, be sure to save the original routine versions to restore the original behavior.

In the C API, changing the global constants is the only way to modify the transport protocol.

5

Automotive Diagnostic Command Set API for LabVIEW

This chapter lists the LabVIEW VIs for the Automotive Diagnostic Command Set API and describes the format, purpose, and parameters for each VI. The VIs are listed alphabetically in four categories: general functions, KWP2000 services, UDS (DiagOnCAN) services, and OBD (On-Board Diagnostics) services.

Section Headings

The following are section headings found in the Automotive Diagnostic Command Set API for LabVIEW VIs.

Purpose

Each VI description briefly describes the VI purpose.

Format

The format section describes the VI format.

Input and Output

The input and output sections list the VI parameters.

Description

The description section gives details about the VI purpose and effect.

List of VIs

The following table is an alphabetical list of the Automotive Diagnostic Command Set VIs.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW

Function	Purpose
ClearDiagnosticInformation.vi	Executes the ClearDiagnosticInformation service and clears selected Diagnostic Trouble Codes (DTCs).
Close Diagnostic.vi	Closes a diagnostic session.
ControlDTCSetting.vi	Executes the ControlDTCSetting service and modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).
Convert from Phys.vi	Converts a physical data value into a binary representation using a type descriptor.
Convert to Phys.vi	Converts a binary representation of a value into its physical value using a type descriptor.
Create Extended CAN IDs.vi	Creates diagnostic CAN IDs according to ISO 15765-2.
Diag Get Property.vi	Gets a diagnostic global internal parameter.
Diag Set Property.vi	Sets a diagnostic global internal parameter.
Diagnostic Service.vi	Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this VI.
DisableNormalMessageTransmission.vi	Executes the DisableNormalMessageTransmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).
DTC to String.vi	Returns a string representation (such as *P1234*) for a 2-byte Diagnostic Trouble Code (DTC).
ECUReset.vi	Executes the ECUReset service and resets the ECU.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
EnableNormalMessageTransmission.vi	Executes the EnableNormalMessageTransmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).
InputOutputControlByLocalIdentifier.vi	Executes the InputOutputControlByLocalIdentifier service. Modifies the ECU I/O port behavior.
OBD Clear Emission Related Diagnostic Information.vi	Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related Diagnostic Trouble Codes (DTCs) in the ECU.
OBD Open.vi	Opens an OBD-II diagnostic session on a CAN port.
OBD Request Control Of On-Board Device.vi	Executes the OBD Request Control Of On-Board Device service. Use this VI to modify ECU I/O port behavior.
OBD Request Current Powertrain Diagnostic Data.vi	Executes the OBD Request Current Powertrain Diagnostic Data service. Reads a data record from the ECU.
OBD Request Emission Related DTCs.vi	Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).
OBD Request Emission Related DTCs During Current Drive Cycle.vi	Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.
OBD Request On-Board Monitoring Test Results.vi	Executes the OBD Request On-Board Monitoring Test Results service. Reads a test data record from the ECU.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
OBD Request Permanent Fault Codes.vi	Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.
OBD Request Powertrain Freeze Frame Data.vi	Executes the OBD Request Powertrain Freeze Frame Data service. Reads a data record from the ECU that has been stored while a Diagnostic Trouble Code occurred.
OBD Request Supported PIDs.vi	Executes the OBD Request Current Powertrain Diagnostic Data service to retrieve the valid PID values for this service.
OBD Request Vehicle Information.vi	Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.
Open Diagnostic.vi	Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.
ReadDataByLocalIdentifier.vi	Executes the ReadDataByLocalIdentifier service. Reads a data record from the ECU.
ReadDTCByStatus.vi	Executes the ReadDiagnosticTroubleCodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).
ReadECUIdentification.vi	Executes the ReadECUIdentification service. Returns ECU identification data from the ECU.
ReadMemoryByAddress.vi	Executes the ReadMemoryByAddress service. Reads data from the ECU memory.
ReadStatusOfDTC.vi	Executes the ReadStatusOfDiagnosticTroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
RequestRoutineResultsByLocalIdentifier.vi	Executes the RequestRoutineResultsByLocalIdentifier service. Returns results from a routine on the ECU.
RequestSeed.vi	Executes the SecurityAccess service to retrieve a seed from the ECU.
SendKey.vi	Executes the SecurityAccess service to send a key to the ECU.
StartDiagnosticSession.vi	Executes the StartDiagnosticSession service. Sets up the ECU in a specific diagnostic mode.
StartRoutineByLocalIdentifier.vi	Executes the StartRoutineByLocalIdentifier service. Executes a routine on the ECU.
StopDiagnosticSession.vi	Executes the StopDiagnosticSession service. Brings the ECU back in normal mode.
StopRoutineByLocalIdentifier.vi	Executes the StopRoutineByLocalIdentifier service. Stops a routine on the ECU.
TesterPresent.vi	Executes the TesterPresent service. Keeps the ECU in diagnostic mode.
UDS ClearDiagnosticInformation.vi	Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).
UDS CommunicationControl.vi	Executes the UDS CommunicationControl service. Use this VI to switch on or off transmission and/or reception of the normal communication messages (usually CAN messages).
UDS ControlDTCSetting.vi	Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) generation behavior.
UDS DiagnosticSessionControl.vi	Executes the UDS DiagnosticSessionControl service. Sets up the ECU in a specific diagnostic mode.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
UDS ECUReset.vi	Executes the UDS ECUReset service. Resets the ECU.
UDS InputOutputControlByIdentifier.vi	Executes the UDS InputOutputControlByIdentifier service. Use this VI to modify ECU I/O port behavior.
UDS ReadDataByIdentifier.vi	Executes the UDS ReadDataByIdentifier service. Reads a data record from the ECU.
UDS ReadMemoryByAddress.vi	Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.
UDS ReportDTCBySeverityMaskRecord.vi	Executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
UDS ReportDTCByStatusMask.vi	Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
UDS ReportSeverityInformationOfDTC.vi	Executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
UDS ReportSupportedDTCs.vi	Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).
UDS RequestDownload.vi	Initiates a download of data to the ECU.
UDS RequestSeed.vi	Executes the UDS SecurityAccess service to retrieve a seed from the ECU.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
UDS RequestTransferExit.vi	Terminates a download/upload process.
UDS RequestUpload.vi	Initiates an upload of data from the ECU.
UDS RoutineControl.vi	Executes the UDS RoutineControl service. Executes a routine on the ECU.
UDS SendKey.vi	Executes the SecurityAccess service to send a key to the ECU.
UDS TesterPresent.vi	Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.
UDS TransferData.vi	Transfers data to/from the ECU in a download/upload process.
UDS WriteDataByIdentifier.vi	Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.
UDS WriteMemoryByAddress.vi	Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.
VWTP Connect.vi	Establishes a connection channel to an ECU using the VW TP 2.0.
VWTP Connection Test.vi	Maintains a connection channel to an ECU using the VW TP 2.0.
VWTP Disconnect.vi	Terminates a connection channel to an ECU using the VW TP 2.0.
WriteDataByLocalIdentifier.vi	Executes the WriteDataByLocalIdentifier service. Writes a data record to the ECU.
WriteMemoryByAddress.vi	Executes the WriteMemoryByAddress service. Writes data to the ECU memory.

General Functions

Close Diagnostic.vi

Purpose

Closes a diagnostic session.

Format

Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

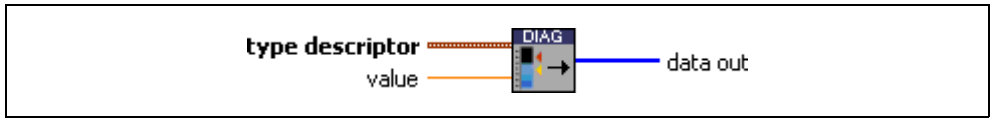
The diagnostic session specified by **diag ref in** is closed, and you can no longer use it for communication to an ECU. Note that this command does not communicate the closing to the ECU before terminating; if this is necessary, you must manually do so (for example, by calling **StopDiagnosticSession.vi**) before calling **Close Diagnostic.vi**.

Convert from Phys.vi

Purpose

Converts a physical data value into a binary representation using a type descriptor.

Format



Input

type descriptor is a cluster that specifies the conversion of the physical value to its binary representation:

Start Byte gives the start byte of the binary representation. For **Convert from Phys.vi**, this value is ignored and always assumed to be 0.

Byte Length is the binary representation byte length.

Byte Order is the byte ordering of the data in the binary representation:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

Data Type is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths of 4 or 8 are allowed.

Scale Factor defines the physical value scaling:

Phys = (Scale Factor) * (binary representation) + (Scale Offset)

Scale Offset (refer to **Scale Factor**)

value is the physical value to be converted.

Output

data out is the binary representation of the physical value. If you build a record of multiple values, you can concatenate the outputs of several instances of **Convert from Phys.vi**.

Description

Data input to diagnostic services (for example, **WriteDataByLocalIdentifier.vi**) is usually a byte stream of binary data. If you have a description of the data input (for example, *byte 3 and 4 are engine RPM scaled as .25 * × RPM in Motorola representation*), you can use **Convert from Phys.vi** to convert the physical value to the byte stream by filling an appropriate type descriptor cluster.

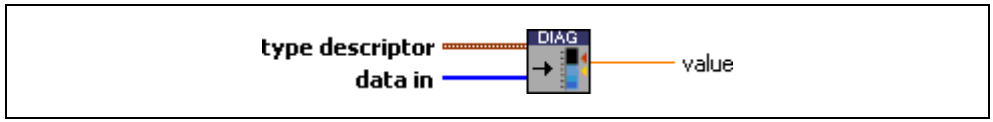
Convert from Phys.vi converts only the portion specified by one type descriptor to a binary representation. If your data input consists of several values, you can use **Convert from Phys.vi** multiple times and concatenate their outputs.

Convert to Phys.vi

Purpose

Converts a binary representation of a value into its physical value using a type descriptor.

Format



Input

type descriptor is a cluster that specifies the conversion of the binary representation to its physical value:

Start Byte gives the binary representation start byte in the **data in** record.

Byte Length is the binary representation byte length.

Byte Order is the byte ordering of the data in the binary representation:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

Data Type is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths of 4 or 8 are allowed.

Scale Factor defines the physical value scaling:

Phys = (Scale Factor) * (binary representation) + (Scale Offset)

Scale Offset (refer to **Scale Factor**)

data in is the data record from which physical values are to be extracted.

Output

value is the physical value extracted from the record.

Description

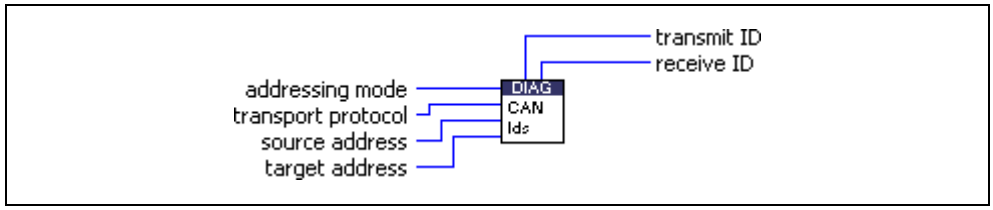
Data output from diagnostic services (for example, **ReadDataByLocalIdentifier.vi**) is usually a byte stream of binary data. If you have a description of the data output (for example, *byte 3 and 4 are engine RPM scaled as .25 * × RPM in Motorola representation*), you can use **Convert to Phys.vi** to extract the physical value from the byte stream by filling an appropriate type descriptor cluster.

Create Extended CAN IDs.vi

Purpose

Creates diagnostic CAN IDs according to ISO 15765-2.

Format



Input

addressing mode specifies whether the ECU is physically or functionally addressed.

transport protocol specifies whether normal or mixed mode addressing is used.

source address is the logical address of the host (diagnostic tester).

target address is the ECU logical address.

Output

transmit ID is the generated CAN identifier for sending diagnostic request messages from the host to the ECU.

receive ID is the generated CAN identifier for sending diagnostic response messages from the ECU to the host.

Description

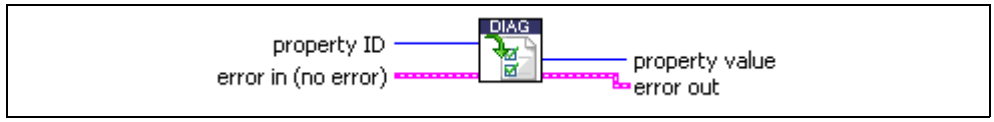
ISO 15765-2 specifies a method (extended/29 bit) of creating CAN identifiers for diagnostic applications given the addressing mode (physical/functional), the transport protocol (normal/mixed), and the 8-bit source and target addresses. This VI implements the construction of these CAN identifiers. You can use them directly in **Open Diagnostic.vi**.

Diag Get Property.vi

Purpose

Gets a diagnostic global internal parameter.

Format



Input

U16

property ID defines the parameter whose value is to be retrieved. You can create the values using an Enum control.

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If this value is

set to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** returns whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** returns the CAN frame content if filled with defined data or random data bytes.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** returns how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by **success?** = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** is the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** is the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

error in is a cluster that describes error conditions occurring before the VI executes. It is copied unchanged to **error out** and has no other effect on the VI. It is provided for sequencing purposes only.

Output

property value is the requested property value.

error out describes error conditions. It is copied unchanged from the **error in** cluster. It is provided for sequencing purposes only.

Description

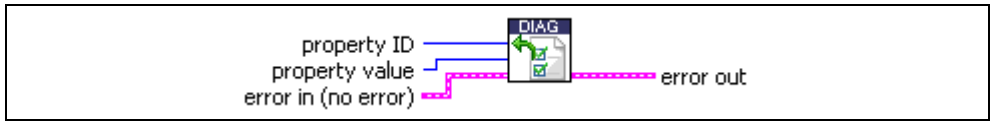
Use this VI to request several internal diagnostic parameters, such as timeouts for the transport protocol. Use **Diag Set Property.vi** to modify them.

Diag Set Property.vi

Purpose

Sets a diagnostic global internal parameter.

Format



Input

U16

property ID defines the parameter whose value is to be retrieved. You can create the values using an Enum control.

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If this value is set to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** specifies whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** specifies the CAN frame content, filled with defined data or random data.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** specifies how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by **success?** = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** defines the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** sets the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

property value is the value of the property to be set.

error in is a cluster that describes error conditions occurring before the VI executes. It is copied unchanged to **error out** and has no other effect on the VI. It is provided for sequencing purposes only.

Output

error out describes error conditions. It is copied unchanged from the **error in** cluster. It is provided for sequencing purposes only.

Description

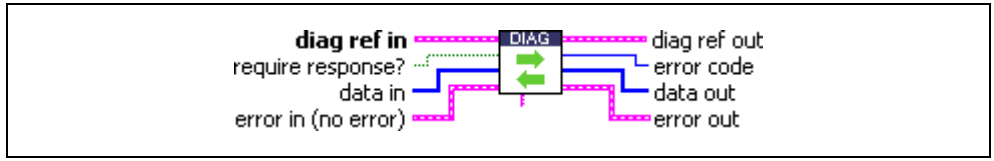
Use this VI to set several internal diagnostic parameters such as timeouts for the transport protocol. Use **Diag Get Property.vi** to read them out.

Diagnostic Service.vi

Purpose

Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this VI.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

require response? indicates whether a diagnostic service expects a response (TRUE) or not (FALSE). In the latter case, **error code** is returned as 0, and **data out** as an empty array.

data in defines the diagnostic service request message sent to the ECU as a stream of bytes.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error code is the error code sent with a negative response message. In addition, the error cluster indicates an error and gives a more detailed description. If no negative response message occurred, 0 is returned.

data out returns the diagnostic service response message (positive or negative) the ECU sends as a stream of bytes.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Diagnostic Service.vi is a generic routine to execute any diagnostic service. The request and response messages are fed unmodified to the **data in** input and retrieved from the **data out** output, respectively. No interpretation of the contents is done, with one exception: the error number is retrieved from a negative response, if one occurs. In this case, an error also is communicated through the **error out** cluster.

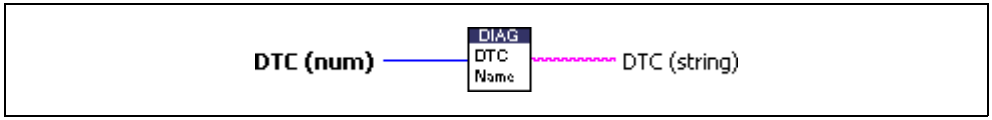
All specialized diagnostic services call **Diagnostic Service.vi** internally.

DTC to String.vi

Purpose

Returns a string representation (such as *P1234*) for a 2-byte Diagnostic Trouble Code (DTC).

Format



Input

DTC (num) is the DTC number as returned in the clusters of **ReadDTCByStatus.vi**, **ReadStatusOfDTC.vi**, **UDS ReportDTCBySeverityMaskRecord.vi**, **UDS ReportDTCByStatusMask.vi**, **UDS ReportSeverityInformationOfDTC.vi**, **UDS ReportSupportedDTCs.vi**, **OBD Request Emission Related DTCs.vi**, or **OBD Request Emission Related DTCs During Current Drive Cycle.vi**.

Note This VI converts only 2-byte DTCs. If you feed in larger numbers, the VI returns garbage.

Output

DTC (string) is the DTC string representation.

Description

The SAE J2012 standard specifies a naming scheme for 2-byte DTCs consisting of one letter and four digits. Use **DTC to String.vi** to convert a DTC numerical representation to this name.

OBD Open.vi

Purpose

Opens an OBD-II diagnostic session on a CAN port.

Format

Input

CAN interface specifies the CAN interface on which the diagnostic communication should take place.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@ni_genie_nixnet* to the protocol string (for example, *CAN1@ni_genie_nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special

RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name as defined in your LabVIEW Project definition. The *lvbitx* filename represents the filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

baudrate is the diagnostic communication baud rate.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a cluster containing all necessary diagnostic session information. Wire this cluster as a handle to all subsequent diagnostic VIs and close it using **Close Diagnostic.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use this VI to open a diagnostic communication channel to an ECU for OBD-II. The CAN port specified as input is initialized, and a handle to it is stored (among other internal data) in the **diag ref out** cluster, which serves as reference for further diagnostic functions.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx.*
- **CAN256**—uses virtual NI-CAN interface 256.

Using NI-XNET hardware:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1,c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at location `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

First, communication to the ECU is tried on the default 11-bit OBD CAN identifiers; if that fails, the default 29-bit OBD CAN identifiers are tried. If that also fails, the VI returns an error.

You can overwrite the default OBD CAN identifiers optionally with any other identifiers.

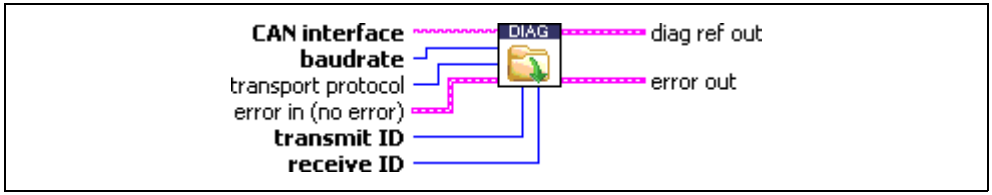
In general, it is not necessary to manipulate the **diag ref out** cluster contents.

Open Diagnostic.vi

Purpose

Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.

Format



Input

CAN interface specifies the CAN interface on which the diagnostic communication should take place. The values are CAN0, CAN1, and so on.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@ni_genie_nixnet* to the protocol string (for example, *CAN1@ni_genie_nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name

after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name as defined in your LabVIEW Project definition. The *lvbitx* filename represents the filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

baudrate is the diagnostic communication baud rate.

transport protocol specifies the transport protocol for transferring the diagnostic service messages over the CAN network. The following values are valid and can be obtained through an enum control:

0 **ISO TP—Normal Mode**: The ISO TP as specified in ISO 15765-2 is used; all eight data bytes of the CAN messages are used for data transfer.

1 **ISO TP—Mixed Mode**: The ISO TP as specified in ISO 15765-2 is used; the first data byte is used as address extension.

2 VW TP 2.0

transmit ID is the CAN identifier for sending diagnostic request messages from the host to the ECU. To specify an extended (29-bit) ID, OR the value with 0x20000000.

receive ID is the CAN identifier or sending diagnostic response messages from the ECU to the host. To specify an extended (29-bit) ID, OR the value with 0x20000000.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a cluster containing all necessary diagnostic session information. Wire this cluster as a handle to all subsequent diagnostic VIs and close it using **Close Diagnostic.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Open Diagnostic.vi opens a diagnostic communication channel to an ECU. The CAN port specified as input is initialized, and a handle to it is stored (among other internal data) in the **diag ref out** cluster, which serves as reference for further diagnostic functions.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx*.
- **CAN256**—uses virtual NI-CAN interface 256.
- **CAN257**—uses virtual NI-CAN interface 257.

Using NI-XNET hardware:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1,c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

Note No communication to the ECU takes place at this point. To open a diagnostic session on the ECU, call **StartDiagnosticSession.vi** or **UDS DiagnosticSessionControl.vi**.

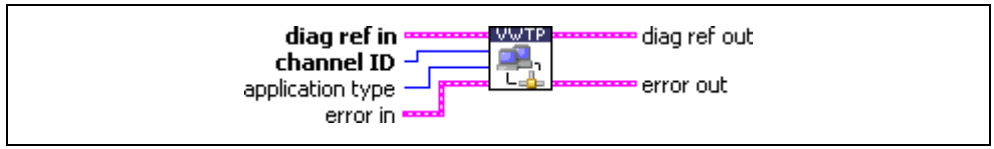
In general, it is not necessary to manipulate the **diag ref out** cluster contents, with one notable exception: If you use the **ISO TP—Mixed Mode** transport protocol, you must store the address extensions for transmit and receive in the appropriate cluster members.

VWTP Connect.vi

Purpose

Establishes a connection channel to an ECU using the VW TP 2.0.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

channel ID defines the CAN identifier on which the ECU responds for this connection. The ECU defines the ID on which the host transmits.

application type specifies the type of communication that takes place on the communication channel. For diagnostic applications, specify *KWP2000 (1)*. The other values are for manufacturer-specific purposes.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

For the VW TP 2.0, you must establish a connection to the ECU before any diagnostic communication can occur. This VI sets up a unique communication channel to an ECU for subsequent diagnostic service requests.

Note You must maintain the communication link you created by periodically (at least once a second) calling **VWTP Connection Test.vi**.

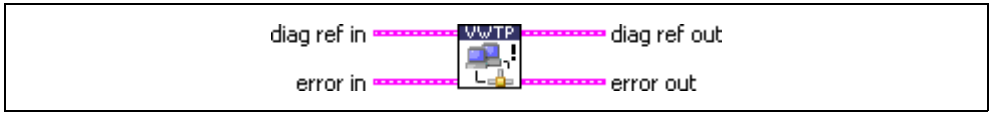
There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

VWTP Connection Test.vi

Purpose

Maintains a connection channel to an ECU using the VW TP 2.0.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

For the VW TP 2.0, you must periodically maintain the connection link to the ECU so that the ECU does not terminate it.

This VI sends a Connection Test message to the ECU and evaluates its response, performing the steps necessary to maintain the connection.

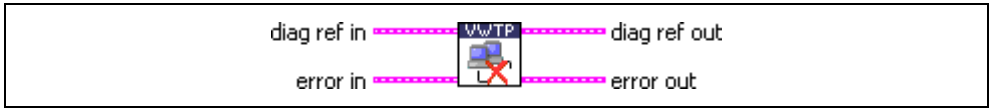
There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

VWTP Disconnect.vi

Purpose

Terminates a connection channel to an ECU using the VW TP 2.0.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

For the VW TP 2.0, you must disconnect the connection link to the ECU to properly terminate communication to the ECU. This VI sends the proper disconnect messages and unlinks the communication.

You can create a new connection to the same ECU using **VWTP Connect.vi** again.

There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

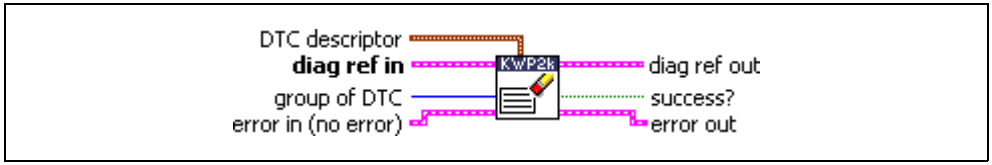
KWP2000 Services

ClearDiagnosticInformation.vi

Purpose

Executes the ClearDiagnosticInformation service and clears selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

The **DTC descriptor** is given here as a parameter basically to convert the **group of DTC** parameter to a binary representation according to **DTC Byte Length** and **Byte Order**.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

group of DTC specifies the group of Diagnostic Trouble Codes to be cleared. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI clears the diagnostic information on the ECU memory. If the **group of DTC** parameter is present, the ECU is requested to clear all memory including the DTCs.

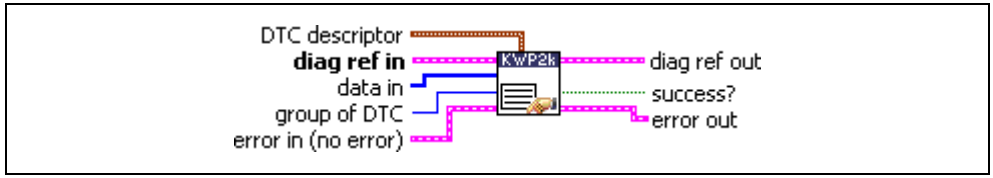
For further details about this service, refer to the ISO 14230-3 standard.

ControlDTCSetting.vi

Purpose

Executes the ControlDTCSetting service and modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

The **DTC descriptor** is given here as a parameter basically to convert the **group of DTC** parameter to a binary representation according to **DTC Byte Length** and **Byte Order**.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in specifies application-specific data that control DTC generation.

group of DTC specifies the group of Diagnostic Trouble Codes to be controlled. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

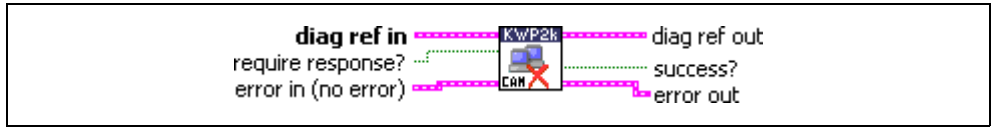
source identifies the VI where the error occurred.

DisableNormalMessageTransmission.vi

Purpose

Executes the DisableNormalMessageTransmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

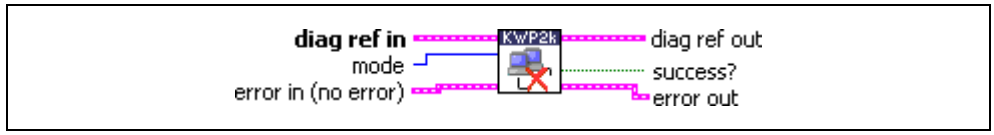
source identifies the VI where the error occurred.

ECUReset.vi

Purpose

Executes the ECUReset service. Resets the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the reset mode:

Hex	Description
01	**PowerOn** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the client (tester) re-establishes communication.
02	**PowerOnWhileMaintainingCommunication** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the server (ECU) maintains communication with the client (tester).
03–7F	**Reserved**
80–FF	**ManufacturerSpecific** This range of values is reserved for vehicle manufacturer-specific use.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

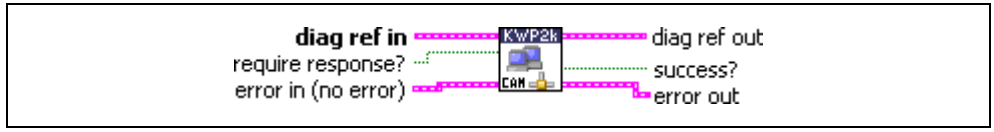
This VI requests the ECU to perform an ECU reset effectively based on the **mode** parameter value content. The vehicle manufacturer determines when the positive response message is sent.

EnableNormalMessageTransmission.vi

Purpose

Executes the EnableNormalMessageTransmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

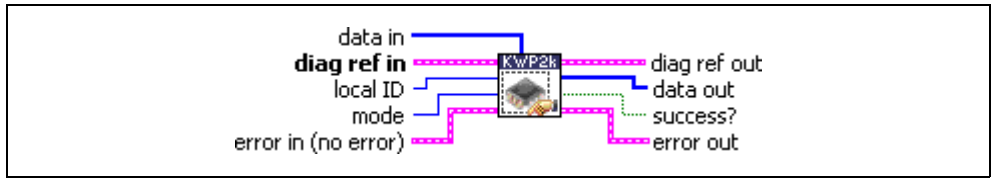
source identifies the VI where the error occurred.

InputOutputControlByLocalIdentifier.vi

Purpose

Executes the InputOutputControlByLocalIdentifier service. Modifies ECU I/O port behavior.

Format



Input

data in defines application-specific data for this service.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the I/O to be manipulated. The values are application specific.

mode defines the type of I/O control. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ReportCurrentState

4: ResetToDefault

5: FreezeCurrentState

7: ShortTermAdjustment

8: LongTermAdjustment

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific data for this service.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by the **local ID** parameter.

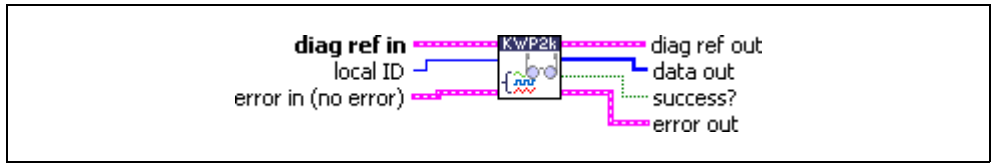
For further details about this service, refer to the ISO 14230-3 standard.

ReadDataByLocalIdentifier.vi

Purpose

Executes the ReadDataByLocalIdentifier service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the data to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the data record from the ECU. If you know the record data description, you can interpret this record using **Convert from Phys.vi**.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests data record values from the ECU identified by the **local ID** parameter.

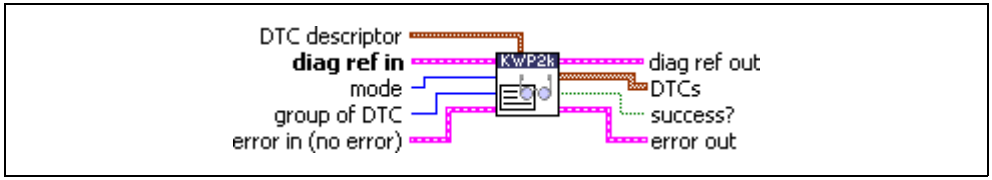
For further details about this service, refer to the ISO 14230-3 standard.

ReadDTCByStatus.vi

Purpose

Executes the ReadDiagnosticTroubleCodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode defines the type of DTCs to be read. The values are application specific. The usual values are:

2: AllIdentified

3: AllSupported

group of DTC specifies the group of Diagnostic Trouble Codes to be read. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**)

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI reads DTCs by status from the ECU memory. If you use the optional **group of DTC** parameter, the ECU reports DTCs only with status information based on the functional group selected by **group of DTC**.

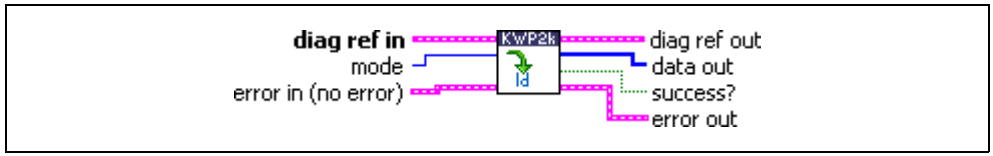
For further details about this service, refer to the ISO 14230-3 standard.

ReadECUIdentification.vi

Purpose

Executes the ReadECUIdentification service. Returns ECU identification data.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the type of identification information to be returned. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU identification data.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests identification data from the ECU. The **mode** parameter identifies the type of identification data requested. The ECU returns identification data that the **data out** parameter can access. The **data out** format and definition are vehicle manufacturer specific.

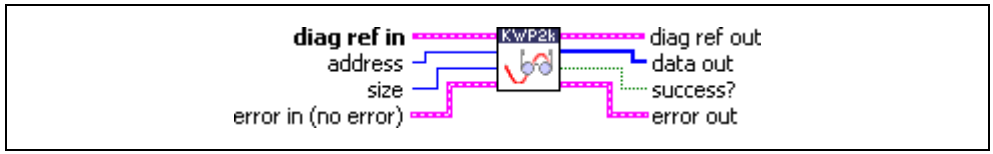
For further details about this service, refer to the ISO 14230-3 standard.

ReadMemoryByAddress.vi

Purpose

Executes the ReadMemoryByAddress service. Reads data from the ECU memory.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address from which data are to be read. Notice that only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be read.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

Data out returns the memory data from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests memory data from the ECU identified by the **address** and **size** parameters. The **data out** format and definition are vehicle manufacturer specific. **data out** includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

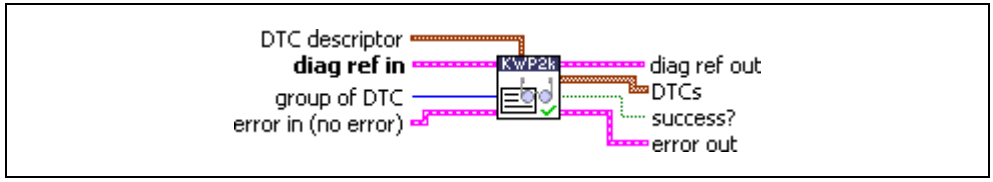
For further details about this service, refer to the ISO 14230-3 standard.

ReadStatusOfDTC.vi

Purpose

Executes the ReadStatusOfDiagnosticTroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

group of DTC specifies the group of Diagnostic Trouble Codes to be read. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle

2 pendingDTC

3 confirmedDTC

4 testNotCompletedSinceLastClear

5 testFailedSinceLastClear

6 testNotCompletedThisMonitoringCycle

7 warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**)

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI reads diagnostic trouble codes from the ECU memory. If you use the optional **group of DTC** parameter, the ECU reports DTCs based only on the functional group selected by **group of DTC**.

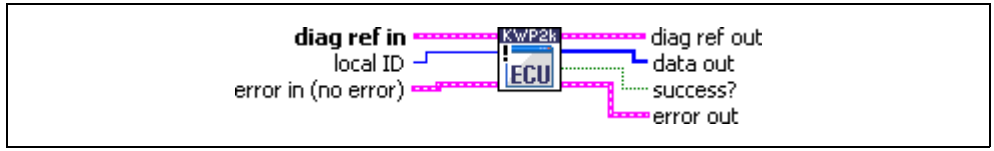
For further details about this service, refer to the ISO 14230-3 standard.

RequestRoutineResultsByLocalIdentifier.vi

Purpose

Executes the RequestRoutineResultsByLocalIdentifier service. Returns results from a routine on the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the routine from which this VI retrieves results. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests results (for example, exit status information) referenced by **local ID** and generated by the routine executed in the ECU memory.

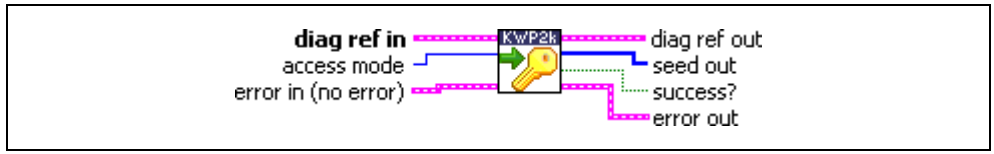
For further details about this service, refer to the ISO 14230-3 standard.

RequestSeed.vi

Purpose

Executes the SecurityAccess service to retrieve a seed from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

seed out returns the seed from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

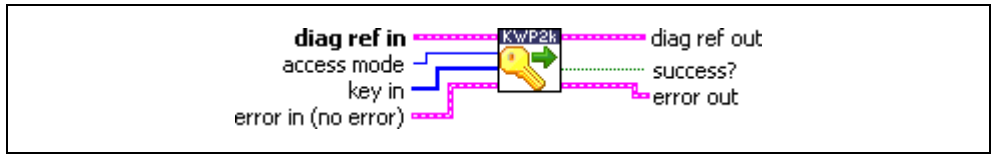
1. Request a seed from the ECU using **RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using SendKey.vi with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

SendKey.vi

Purpose

Executes the SecurityAccess service to send a key to the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

key in defines the key data to be sent to the ECU.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

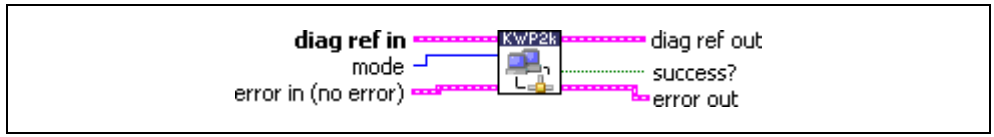
1. Request a seed from the ECU using **RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using **SendKey.vi** with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

StartDiagnosticSession.vi

Purpose

Executes the StartDiagnosticSession service. Sets up the ECU in a specific diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the diagnostic mode into which the ECU is brought. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI enables different diagnostic modes in the ECU. The possible diagnostic modes are not defined in the ISO 14230 standard and are application specific. A diagnostic session starts only if communication with the ECU is established. For more details about starting communication, refer to the ISO 14230-2 standard. If no diagnostic session has been requested after **Open Diagnostic.vi**, a default session is automatically enabled in the ECU. The default session supports at least the following services:

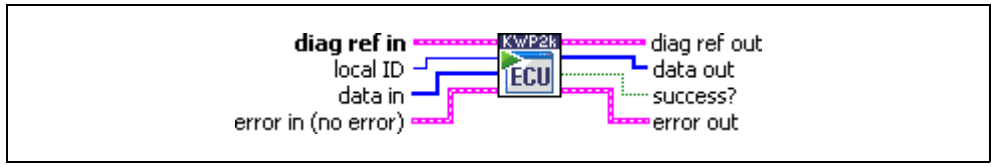
- The StopCommunication service (refer to **Close Diagnostic.vi** and the ISO 14230-2 standard).
- The TesterPresent service (refer to **TesterPresent.vi** and the ISO 14230-3 standard).

StartRoutineByLocalIdentifier.vi

Purpose

Executes the StartRoutineByLocalIdentifier service. Executes a routine on the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the routine to be started. The values are application specific.

data in defines application-specific input parameters for the routine.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI starts a routine in the ECU memory. The routine in the ECU starts after the positive response message is sent. The routine stops until **StopRoutineByLocalIdentifier.vi** is issued. The routines could be either tests run instead of normal operating code or routines enabled and executed with the normal operating code running. In the first case, you may need to switch the ECU to a specific diagnostic mode using **StartDiagnosticSession.vi** or unlock the ECU using the SecurityAccess service prior to using **StartRoutineByLocalIdentifier.vi**.

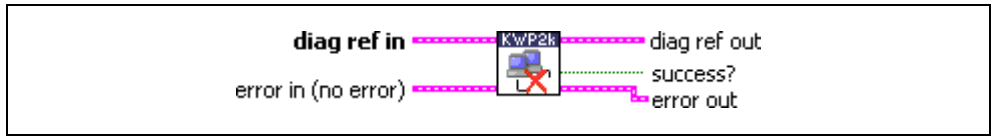
For further details about this service, refer to the ISO 14230-3 standard.

StopDiagnosticSession.vi

Purpose

Executes the StopDiagnosticSession service. Returns the ECU to normal mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

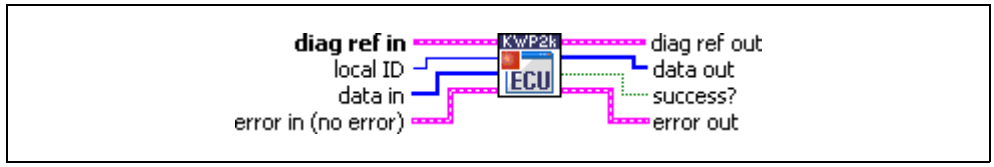
This VI disables the current ECU diagnostic mode. A diagnostic session stops only if communication is established with the ECU and a diagnostic session is running. If no diagnostic session is running, the default session is active. **StopDiagnosticSession.vi** cannot disable the default session. If the ECU has stopped the current diagnostic session, it performs the necessary action to restore its normal operating conditions. Restoring the normal operating conditions of the ECU may include resetting all controlled actuators if they were activated during the diagnostic session being stopped, and resuming all normal ECU algorithms. You should call **StopDiagnosticSession.vi** before disabling communication with **Close Diagnostic.vi**, but only if you previously used **StartDiagnosticSession.vi**.

StopRoutineByLocalIdentifier.vi

Purpose

Executes the StopRoutineByLocalIdentifier service. Stops a routine on the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the routine to be stopped. The values are application specific.

data in defines application-specific input parameters for the routine.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI stops a routine in the ECU memory referenced by the **local ID** parameter.

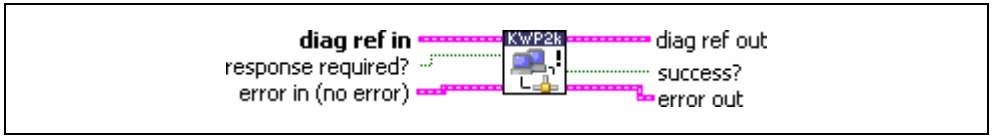
For further details about this service, refer to the ISO 14230-3 standard.

TesterPresent.vi

Purpose

Executes the TesterPresent service. Keeps the ECU in diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

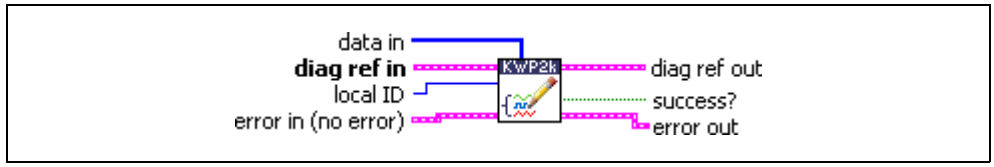
Keep calling **TesterPresent.vi** within the ECU timeout period if no other service is executed.

WriteDataByLocalIdentifier.vi

Purpose

Executes the WriteDataByLocalIdentifier service. Writes a data record to the ECU.

Format



Input

data in defines the data record written to the ECU. If you know the record data description, you can use **Convert from Phys.vi** to generate this record.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the data to be written. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the KWP2000 WriteDataByLocalIdentifier service and writes RecordValues (data values) to the ECU. **data in** identifies the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

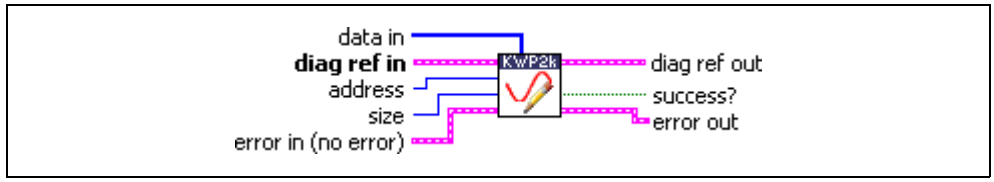
For further details about this service, refer to the ISO 14230-3 standard.

WriteMemoryByAddress.vi

Purpose

Executes the WriteMemoryByAddress service. Writes data to the ECU memory.

Format



Input

data in defines the memory block to be written to the ECU.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address to which data are written. Notice that only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be written.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the KWP2000 WriteDataByAddress service and writes RecordValues (data values) to the ECU. **address** and **size** identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 14230-3 standard.

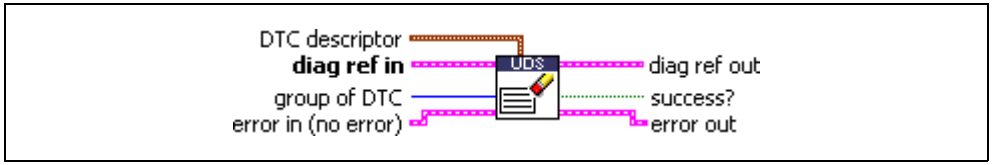
UDS (DiagOnCAN) Services

UDS ClearDiagnosticInformation.vi

Purpose

Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

The **DTC descriptor** is given here as a parameter basically to convert the **group of DTC** parameter to a binary representation according to **DTC Byte Length** and **Byte Order**.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

group of DTC specifies the group of Diagnostic Trouble Codes to be cleared. The values are application specific. The following value has a special meaning, and you can specify it through a ring control:

0xFFFFFF All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI clears the diagnostic information on the ECU memory. If the **group of DTC** parameter is present, the ECU is requested to clear all memory including the DTCs.

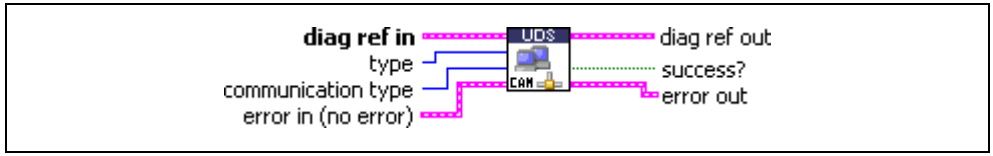
For further details about this service, refer to the ISO 15765-3 standard.

UDS CommunicationControl.vi

Purpose

Executes the UDS CommunicationControl service. Use this VI to switch transmission and/or reception of the normal communication messages (usually CAN messages) on or off.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

type indicates whether transmission/reception is to be switched on/off. The usual values are:

00: enableRxAndTx

01: enableRxAndDisableTx

02: disableRxAndEnableTx

03: disableRxAndTx

communication type is a bitfield indicating the application level to change. The usual values are:

01: application

02: networkManagement

You can change more than one level at a time.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

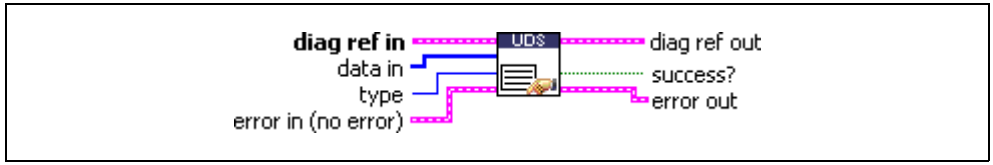
This VI executes the UDS CommunicationControl service and switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off. The **type** and **communication type** parameters are vehicle manufacturer specific (one OEM may disable the transmission only, while another OEM may disable the transmission and the reception based on vehicle manufacturer specific needs). The request is either transmitted functionally addressed to all ECUs with a single request message, or transmitted physically addressed to each ECU in a separate request message.

UDS ControlDTCSetting.vi

Purpose

Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) generation behavior.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in specifies application-specific data that control DTC generation.

type specifies the control mode:

1: on

2: off

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

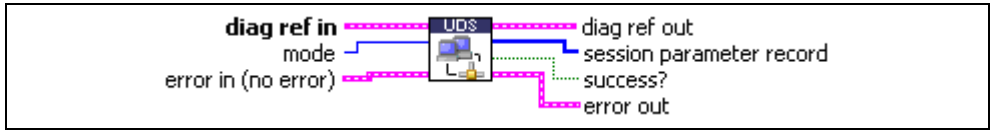
source identifies the VI where the error occurred.

UDS DiagnosticSessionControl.vi

Purpose

Executes the UDS DiagnosticSessionControl service. Sets up the ECU in a specific diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the diagnostic mode into which the ECU is brought. The values are application specific. The usual values are:

01: defaultSession

02: ECUProgrammingSession

03: ECUExtendedDiagnosticSession

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

session parameter record returns implementation-dependent data from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

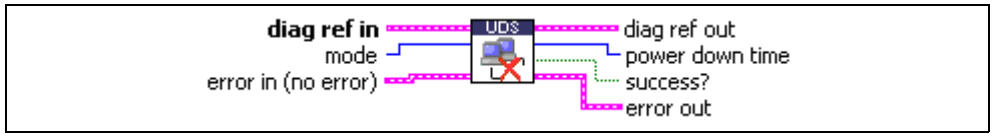
source identifies the VI where the error occurred.

UDS ECUReset.vi

Purpose

Executes the UDS ECUReset service. Resets the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the reset mode:

Hex	Description
01	hardReset
02	keyOffOnReset
03	softReset
04	enableRapidPowerShutDown
05	disableRapidPowerShutDown

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

power down time returns the minimum standby sequence time that the server remains in the power-down sequence in seconds. A value of FF hex indicates a failure or time not available.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests the ECU to perform an ECU reset effectively based on the **mode** parameter value content. The vehicle manufacturer determines when the positive response message is sent.

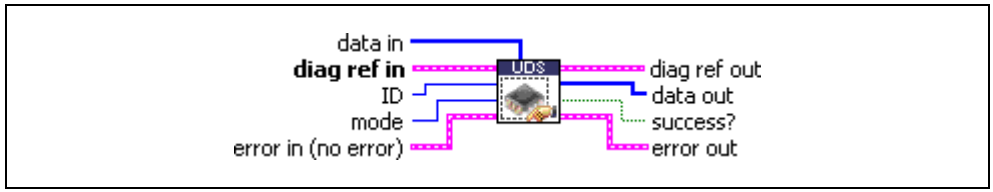
For further details about this service, refer to the ISO 15765-3 standard.

UDS InputOutputControlByIdentifier.vi

Purpose

Executes the UDS InputOutputControlByIdentifier service. Modifies ECU I/O port behavior.

Format



Input

data in defines application specific data for this service.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the I/O to be manipulated. The values are application specific.

mode defines the I/O control type. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ResetToDefault

2: FreezeCurrentState

3: ShortTermAdjustment

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific data for this service.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by the **local ID** parameter.

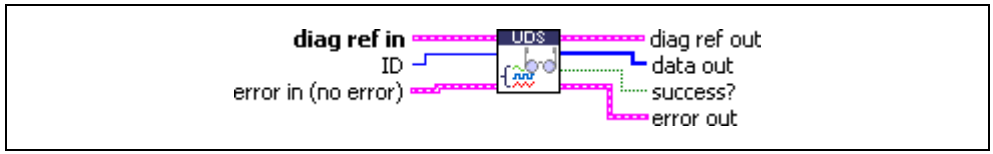
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReadDataByIdentifier.vi

Purpose

Executes the UDS ReadDataByIdentifier service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the data to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the data record from the ECU. If you know the record data description, you can use **Convert to Phys.vi** to interpret this record.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests data record values from the ECU identified by the **ID** parameter.

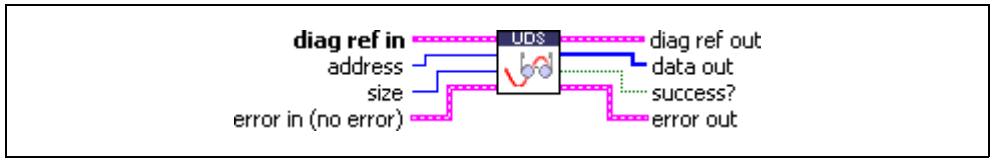
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReadMemoryByAddress.vi

Purpose

Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address from which data are to be read. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be read.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU memory data.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests ECU memory data identified by the **address** and **size** parameters. The **data out** format and definition are vehicle manufacturer specific. **data out** includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

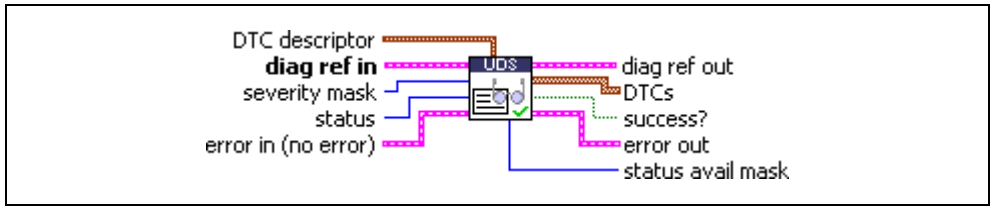
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReportDTCBySeverityMaskRecord.vi

Purpose

Executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

severity mask defines the status of DTCs to be read. The values are application specific.

status defines the status of DTCs to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs.

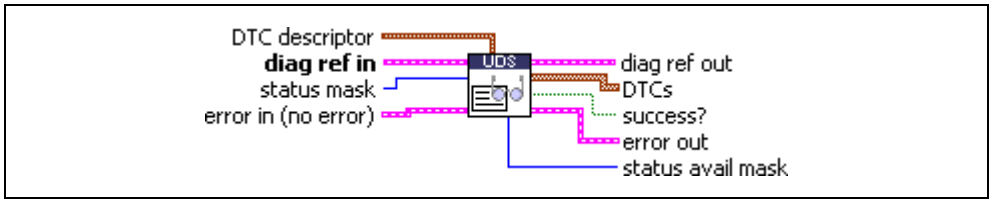
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReportDTCByStatusMask.vi

Purpose

Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

status mask defines the status of DTCs to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU.

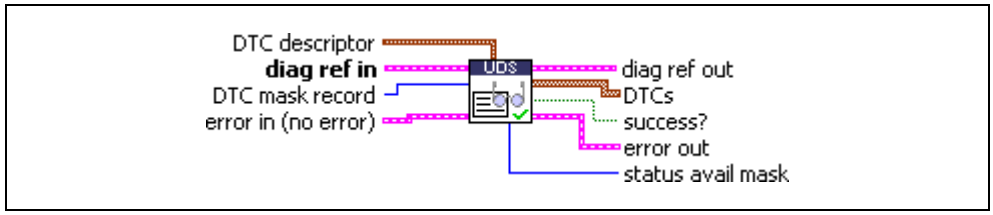
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReportSeverityInformationOfDTC.vi

Purpose

Executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

DTC mask record defines the status of DTCs to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU memory.

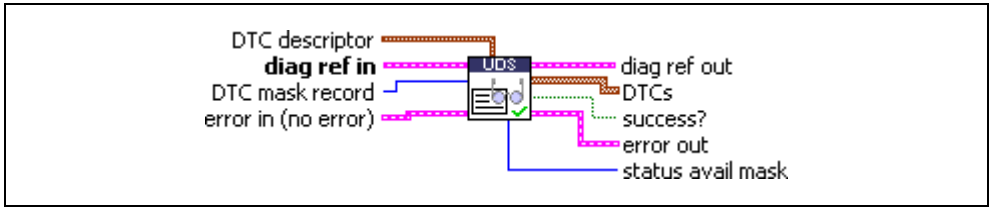
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReportSupportedDTCs.vi

Purpose

Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads all supported DTCs from the ECU memory.

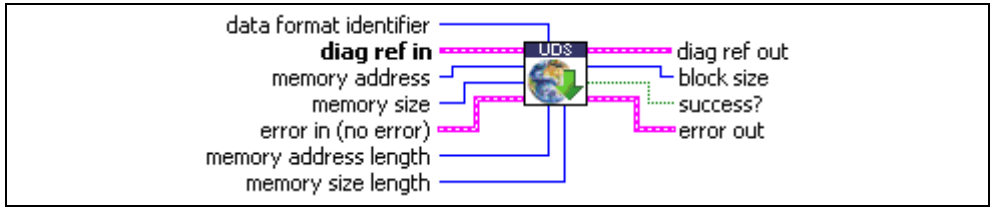
For further details about this service, refer to the ISO 15765-3 standard.

UDS RequestDownload.vi

Purpose

Initiates a download of data to the ECU.

Format



Input

data format identifier defines the compression and encryption scheme to be used for the data blocks written to the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

diag ref in specifies the handle for the diagnostic session. This is obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

memory address defines the memory address to which data are written.

memory size defines the size of the data to be written.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

memory address length defines the number of bytes of the **memory address** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

memory size length defines the number of bytes of the **memory size** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

Output

diag ref out is a copy of **diag ref in.** It can be wired to subsequent diagnostic VIs.

block size returns the number of data bytes to be transferred to the ECU in subsequent **UDS TransferData.vi** requests.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

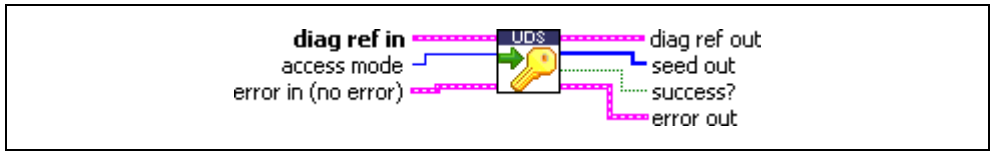
UDS RequestDownload.vi initiates the download of a data block to the ECU. This is required to set up the download process; the actual data transfer occurs with subsequent **UDS TransferData.vi** requests. The transfer must occur in blocks of the size that this service returns (the **block size** parameter). After the download completes, use the **UDS RequestTransferExit.vi** service to terminate the process.

UDS RequestSeed.vi

Purpose

Executes the UDS SecurityAccess service to retrieve a seed from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

seed out returns the seed from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

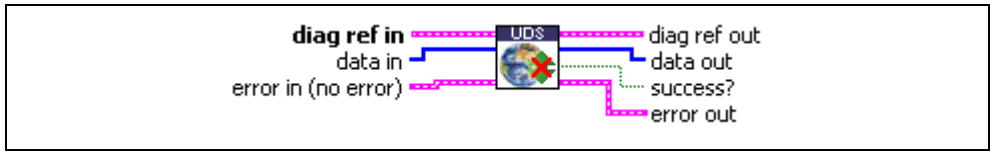
1. Request a seed from the ECU using **UDS RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using **UDS SendKey.vi** with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

UDS RequestTransferExit.vi

Purpose

Terminates a download/upload process.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in defines a data record to be written to the ECU as part of the termination process. The meaning is implementation dependent; this might be a checksum or a similar verification instrument.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. It can be wired to subsequent diagnostic VIs.

data out returns a memory data block from the ECU as part of the termination process. The meaning is implementation dependent; this might be a checksum or a similar verification instrument.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

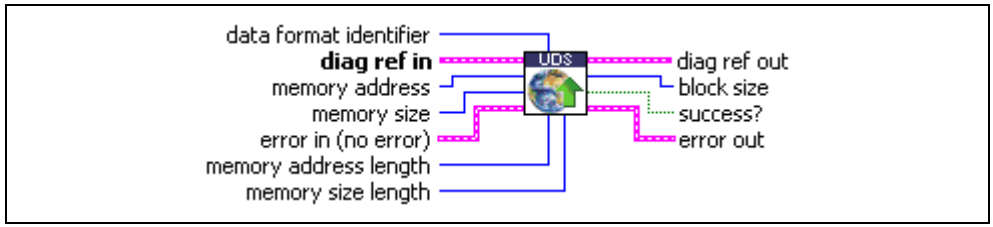
UDS RequestTransferExit.vi terminates a download or upload process initialized with **UDS RequestDownload.vi** or **UDS RequestUpload.vi**.

UDS RequestUpload.vi

Purpose

Initiates an upload of data from the ECU.

Format



Input

data format identifier defines the compression and encryption scheme to be used for the data blocks read from the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

diag ref in specifies the handle for the diagnostic session. This is obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

memory address defines the memory address from which data are read.

memory size defines the size of the data to be read.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

memory address length defines the number of bytes of the **memory address** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

memory size length defines the number of bytes of the **memory size** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

block size returns the number of data bytes to be transferred from the ECU in subsequent **UDS TransferData.vi** requests.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

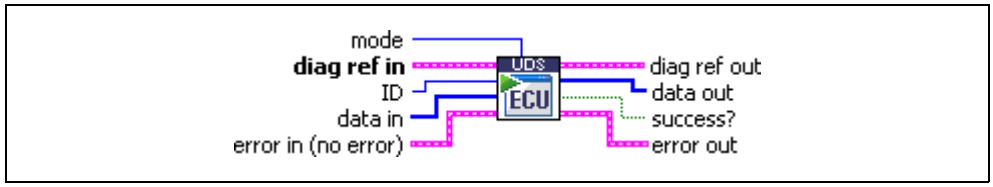
UDS RequestUpload.vi initiates the upload of a data block from the ECU. This is required to set up the upload process; the actual data transfer occurs with subsequent **UDS TransferData.vi** requests. The transfer must occur in blocks of the size that this service returns (the **block size** parameter). After the upload completes, use the **UDS RequestTransferExit.vi** service to terminate the process.

UDS RoutineControl.vi

Purpose

Executes the UDS RoutineControl service. Executes a routine on the ECU.

Format



Input

mode defines the service operation mode. You can obtain the values from a ring control:

1: Start Routine

2: Stop Routine

3: Request Routine Results

Other values are application specific.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the routine to be started. The values are application specific.

data in defines application-specific input parameters for the routine.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI executes the UDS RoutineControl service and launches an ECU routine, stops an ECU routine, or requests ECU routine results from the ECU.

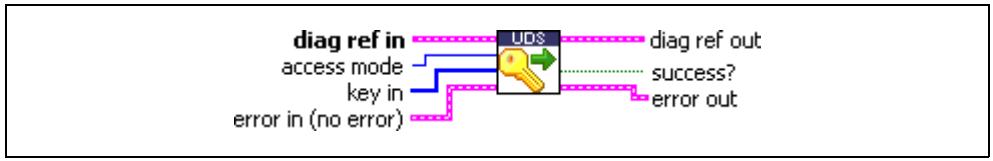
For further details about this service, refer to the ISO 15765-3 standard.

UDS SendKey.vi

Purpose

Executes the SecurityAccess service to send a key to the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

key in defines the key data to be sent to the ECU.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

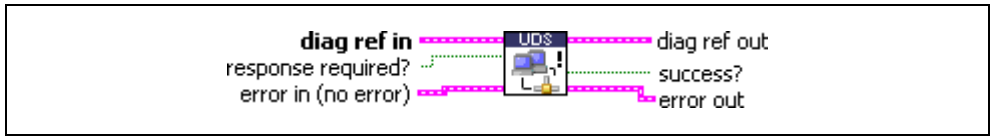
1. Request a seed from the ECU using **UDS RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using **UDS SendKey.vi** with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

UDS TesterPresent.vi

Purpose

Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

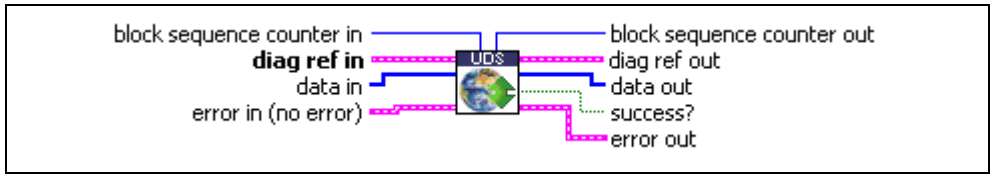
Keep calling **UDS TesterPresent.vi** within the ECU timeout period if no other service is executed.

UDS TransferData.vi

Purpose

Transfers data to/from the ECU in a download/upload process.

Format



Input

block sequence counter in is used to number the data blocks to be transferred to/from the ECU. The block sequence counter value starts at 01 hex with the first **UDS TransferData.vi** request that follows the **UDS RequestDownload.vi** or **UDS RequestUpload.vi** service. Its value is incremented by 1 for each subsequent **UDS TransferData.vi** request. At the value of FF hex, the block sequence counter rolls over and starts at 00 hex with the next **UDS TransferData.vi** request.

The block sequence counter is updated automatically and returned in the **block sequence counter out** parameter.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in defines the data block to be written to the ECU.

For a download, this is a memory data block to be downloaded to the ECU.

For an upload, the meaning is implementation dependent; in most cases, it is sufficient to leave the parameter empty (default).

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

block sequence counter out returns the updated value of the block sequence counter.

diag ref out is a copy of **diag ref in**. It can be wired to subsequent diagnostic VIs.

data out returns the memory data from the ECU.

For a download, this might contain a checksum or similar verification instrument; the meaning is implementation dependent.

For an upload, this is a memory data block uploaded from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

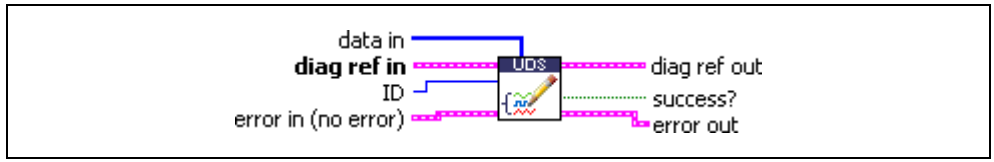
UDS TransferData.vi executes the data transfer of a download process (initiated with a previous **UDS RequestDownload.vi** request) or an upload process (initiated with a previous **UDS RequestUpload.vi** request). The data transfer must occur in blocks of the size returned in the **block size** parameter of the respective request service. After the data transfer has completed, terminate the operation by calling the **UDS RequestTransferExit.vi** service.

UDS WriteDataByIdentifier.vi

Purpose

Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.

Format



Input

data in defines the data record to be written to the ECU. If you know the the data description record, you can use **Convert from Phys.vi** to generate this record.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the data to be written. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the UDS service WriteDataByIdentifier and writes RecordValues (data values) to the ECU. **data in** identifies the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

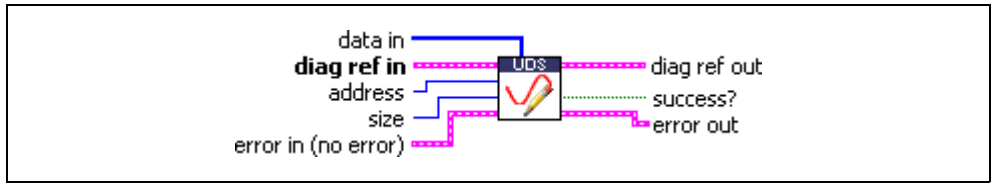
For further details about this service, refer to the 15765-3 standard.

UDS WriteMemoryByAddress.vi

Purpose

Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.

Format



Input

data in defines the memory block to be written to the ECU.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address to which data are to be written. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be written.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the UDS service WriteMemoryByAddress and writes RecordValues (data values) to the ECU. **address** and **size** identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 15765-3 standard.

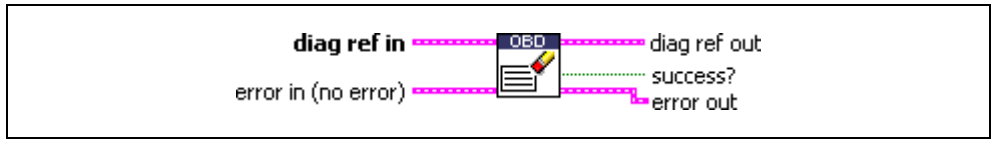
OBD (On-Board Diagnostics) Services

OBD Clear Emission Related Diagnostic Information.vi

Purpose

Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related Diagnostic Trouble Codes (DTCs) in the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

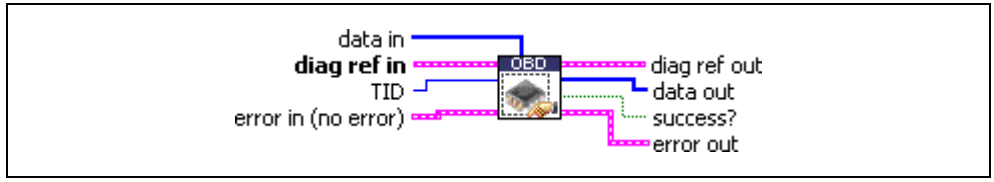
source identifies the VI where the error occurred.

OBD Request Control Of On-Board Device.vi

Purpose

Executes the OBD Request Control Of On-Board Device service. Modifies ECU I/O port behavior.

Format



Input

data in defines application-specific data for this service.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

TID defines the test identifier of the I/O to be manipulated. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific data for this service.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

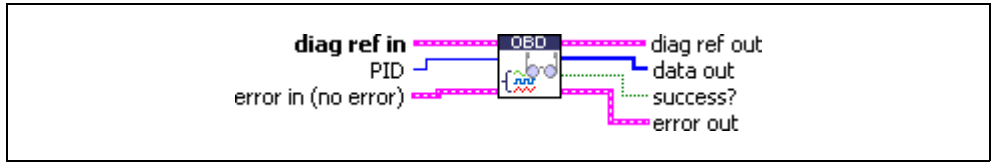
source identifies the VI where the error occurred.

OBD Request Current Powertrain Diagnostic Data.vi

Purpose

Executes the OBD Request Current Powertrain Diagnostic Data service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

PID defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record. If you know the record data description, you can use **Convert from Phys.vi** to interpret this record. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

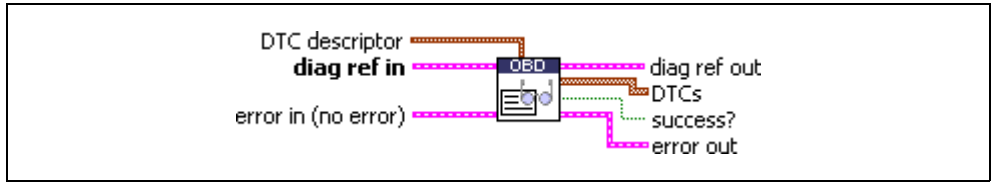
source identifies the VI where the error occurred.

OBD Request Emission Related DTCs.vi

Purpose

Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

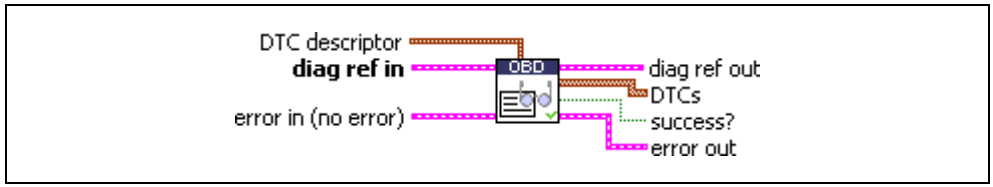
source identifies the VI where the error occurred.

OBD Request Emission Related DTCs During Current Drive Cycle.vi

Purpose

Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

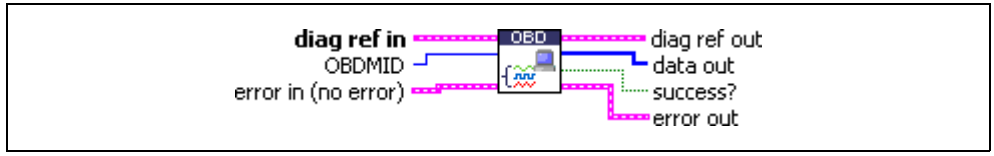
source identifies the VI where the error occurred.

OBD Request On-Board Monitoring Test Results.vi

Purpose

Executes the OBD Request On-Board Monitoring Test Results service. Reads a test data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

OBDMID defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record. If you know the record data description, you can use **Convert from Phys.vi** to interpret this record. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

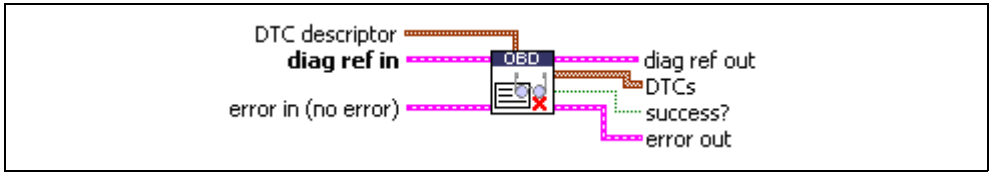
source identifies the VI where the error occurred.

OBD Request Permanent Fault Codes.vi

Purpose

Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

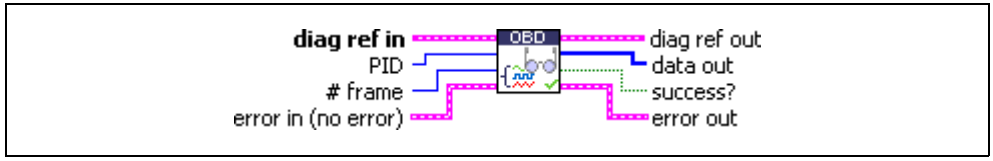
source identifies the VI where the error occurred.

OBD Request Powertrain Freeze Frame Data.vi

Purpose

Executes the OBD Request Powertrain Freeze Frame Data service. Reads an ECU data record stored while a Diagnostic Trouble Code occurred.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

PID defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

 # frame is the number of the freeze frame from which the data are to be retrieved.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record. If you know the record data description, you can use **Convert from Phys.vi** to interpret this record. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

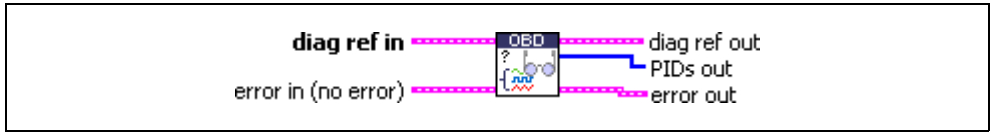
source identifies the VI where the error occurred.

OBD Request Supported PIDs.vi

Purpose

Executes the OBD Request Current Powertrain Diagnostic Data service to retrieve the valid PID values for this service.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

PIDs out returns an array of valid PIDs for the OBD Request Current Powertrain Diagnostic Data service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

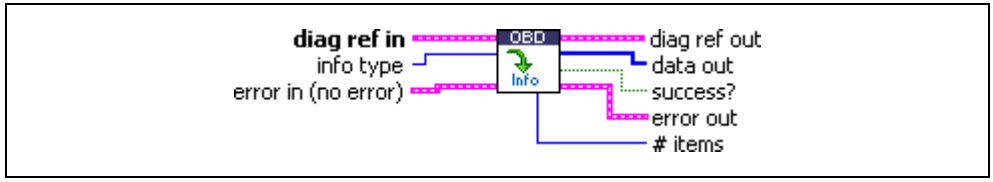
source identifies the VI where the error occurred.

OBD Request Vehicle Information.vi

Purpose

Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

info type defines the type of information to be read. The values are defined in the SAE J1979 standard.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the vehicle information from the ECU. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

 # items is the number of data items (not bytes) this service returns.

6

Automotive Diagnostic Command Set API for C

This chapter lists the Automotive Diagnostic Command Set API functions and describes their format, purpose, and parameters. Unless otherwise stated, each Automotive Diagnostic Command Set function suspends execution of the calling thread until it completes. The functions are listed alphabetically in four categories: general functions, KWP2000 services, UDS (DiagOnCAN) services, and OBD (On-Board Diagnostics) services.

Section Headings

The following are section headings found in the Automotive Diagnostic Command Set for C functions.

Purpose

Each function description includes a brief statement of the function purpose.

Format

The format section describes the function format for the C programming language.

Input and Output

The input and output sections list the function parameters.

Description

The description section gives details about the function purpose and effect.

List of Data Types

The following data types are used with the Automotive Diagnostic Command Set API for C functions.

Table 6-1. Data Types for the Automotive Diagnostic Command Set for C

Data Type	Purpose
`i8`	8-bit signed integer
`i16`	16-bit signed integer
`i32`	32-bit signed integer
`u8`	8-bit unsigned integer
`u16`	16-bit unsigned integer
`u32`	32-bit unsigned integer
`f32`	32-bit floating-point number
`f64`	64-bit floating-point number
`str`	ASCII string represented as an array of characters terminated by null character (`'\0'`). This type is used with output strings. `str` is typically used in the Automotive Diagnostic Command Set API as a pointer to a string, as `char*`.
`cstr`	ASCII string represented as an array of characters terminated by null character (`'\0'`). This type is used with input strings. `cstr` is typically used in the Automotive Diagnostic Command Set as a pointer to a string, as `const char*`.

List of Functions

The following table contains an alphabetical list of the Automotive Diagnostic Command Set API functions.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C

Function	Purpose
ndClearDiagnosticInformation	Executes the ClearDiagnostic Information service. Clears selected Diagnostic Trouble Codes (DTCs).
ndCloseDiagnostic	Closes a diagnostic session.
ndControlDTCSetting	Executes the ControlDTCSetting service. Modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).
ndConvertFromPhys	Converts a physical data value into a binary representation using a type descriptor.
ndConvertToPhys	Converts a binary representation of a value into its physical value using a type descriptor.
ndCreateExtendedCANIds	Creates diagnostic CAN identifiers according to ISO 15765-2.
ndDiagnosticService	Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this function.
ndDisableNormalMessageTransmission	Executes the DisableNormalMessage Transmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndDTCToString	Returns a string representation (such as *P1234*) for a 2-byte diagnostic trouble code (DTC).
ndECUReset	Executes the ECUReset service. Resets the ECU.
ndEnableNormalMessageTransmission	Executes the EnableNormalMessage Transmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).
ndGetProperty	Get a diagnostic global internal parameter.
ndInputOutputControlByLocalIdentifier	Executes the InputOutputControlBy LocalIdentifier service. Modifies the ECU I/O port behavior.
ndOBDClearEmissionRelatedDiagnosticInformation	Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related diagnostic trouble codes (DTCs) in the ECU.
ndOBDOpen	Opens a diagnostic session on a CAN port for OBD-II.
ndOBDRequestControlOfOnBoardDevice	Executes the OBD Request Control Of On-Board Device service. Modifies ECU I/O port behavior.
ndOBDRequestCurrentPowertrainDiagnosticData	Executes the OBD Request Current Powertrain Diagnostic Data service. Reads an ECU data record.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndOBDRequestEmissionRelatedDTCs	Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).
ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle	Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.
ndOBDRequestOnBoardMonitoringTestResults	Executes the OBD Request On-Board Monitoring Test Results service. Reads an ECU test data record.
ndOBDRequestPermanentFaultCodes	Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.
ndOBDRequestPowertrainFreezeFrameData	Executes the OBD Request Powertrain Freeze Frame Data service. Reads an ECU data record stored while a diagnostic trouble code occurred.
ndOBDRequestVehicleInformation	Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.
ndOpenDiagnostic	Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.
ndReadDataByLocalIdentifier	Executes the ReadDataByLocal Identifier service. Reads an ECU data record.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndReadDTCByStatus	Executes the ReadDiagnosticTrouble CodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).
ndReadECUIdentification	Executes the ReadECUIdentification service. Returns ECU identification data from the ECU.
ndReadMemoryByAddress	Executes the ReadMemoryByAddress service. Reads data from the ECU memory.
ndReadStatusOfDTC	Executes the ReadStatusOfDiagnostic TroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).
ndRequestRoutineResultsByLocalIdentifier	Executes the RequestRoutineResultsByLocalI dentifier service. Returns results from an ECU routine.
ndRequestSeed	Executes the SecurityAccess service to retrieve a seed from the ECU.
ndSendKey	Executes the SecurityAccess service to send a key to the ECU.
ndSetProperty	Set a diagnostic global internal parameter.
ndStartDiagnosticSession	Executes the StartDiagnosticSession service. The ECU is set up in a specific diagnostic mode.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndStartRoutineByLocalIdentifier	Executes the StartRoutineByLocal Identifier service. Executes a routine on the ECU.
ndStatusToString	Returns a description for an error code.
ndStopDiagnosticSession	Executes the StopDiagnosticSession service. Returns the ECU to normal mode.
ndStopRoutineByLocalIdentifier	Executes the StopRoutineByLocal Identifier service. Stops a routine on the ECU.
ndTesterPresent	Executes the TesterPresent service. Keeps the ECU in diagnostic mode.
ndUDSClearDiagnosticInformation	Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).
ndUDSCommunicationControl	Executes the UDS CommunicationControl service. Switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off.
ndUDSControlDTCSetting	Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) behavior.
ndUDSDiagnosticSessionControl	Executes the UDS DiagnosticSessionControl service. The ECU is set up in a specific diagnostic mode.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndUDSECUReset	Executes the UDS ECUReset service. Resets the ECU.
ndUDSInputOutputControlByIdentifier	Executes the UDS InputOutputControlByIdentifier service. Modifies ECU I/O port behavior.
ndUDSReadDataByIdentifier	Executes the UDS ReadDataByIdentifier service. Reads an ECU data record.
ndUDSReadMemoryByAddress	Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.
ndUDSReportDTCBySeverityMaskRecord	Executes the ReportDTCBySeverity MaskRecord subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
ndUDSReportDTCByStatusMask	Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
ndUDSReportSeverityInformationOfDTC	Executes the ReportSeverity InformationOfDTC subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs) are read.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndUDSReportSupportedDTCs	Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).
ndUDSRequestDownload	Initiates a download of data to the ECU.
ndUDSRequestSeed	Executes the UDS SecurityAccess service to retrieve a seed from the ECU.
ndUDSRequestTransferExit	Terminates a download/upload process.
ndUDSRequestUpload	Initiates an upload of data from the ECU.
ndUDSRoutineControl	Executes the UDS RoutineControl service. Executes a routine on the ECU.
ndUDSSendKey	Executes the UDS SecurityAccess service to send a key to the ECU.
ndUDSTesterPresent	Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.
ndUDSTransferData	Transfers data to/from the ECU in a download/upload process.
ndUDSWriteDataByIdentifier	Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.
ndUDSWriteMemoryByAddress	Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.
ndVWTPConnect	Establishes a connection channel to an ECU using the VW TP 2.0.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndVWTPConnectionTest	Maintains a connection channel to an ECU using the VW TP 2.0.
ndVWTPDisconnect	Terminates a connection channel to an ECU using the VW TP 2.0.
ndWriteDataByLocalIdentifier	Executes the WriteDataByLocal Identifier service. Writes a data record to the ECU.
ndWriteMemoryByAddress	Executes the WriteMemoryByAddress service. Writes data to the ECU memory.

General Functions

ndCloseDiagnostic

Purpose

Closes a diagnostic session.

Format

```
long ndCloseDiagnostic(
       TD1 *diagRefIn);
```

Input

`diagRefIn`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The diagnostic session `diagRefIn` specifies is closed, and you can no longer use it for communication to an ECU. This command does not communicate the closing to the ECU before terminating; if this is necessary, you must manually do so (for example, by calling `ndStopDiagnosticSession`) before calling `ndCloseDiagnostic`.

ndConvertFromPhys

Purpose

Converts a physical data value into a binary representation using a type descriptor.

Format

```
void ndConvertFromPhys(
       TD2 *typeDescriptor,
       double value,
       unsigned char dataOut[],
       long *len);
```

Input

`typeDescriptor`

A struct that specifies the conversion of the physical value to its binary representation:

```
typedef struct {
    long StartByte;
    long ByteLength;
    unsigned short ByteOrder;
    unsigned short DataType;
    double ScaleFactor;
    double ScaleOffset;
    } TD2;
```

`StartByte` is ignored by `ndConvertFromPhys`.

`ByteLength` is the number of bytes in the binary representation.

`ByteOrder` defines the byte order for multibyte representations. The values are:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

`DataType` is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths 4 or 8 are allowed.

`ScaleFactor` defines the physical value scaling:

Phys = (`ScaleFactor`) * (binary representation) + (`ScaleOffset`)

`ScaleOffset` (refer to `ScaleFactor`)

`value`

The physical value to be converted to a binary representation.

Output

`dataOut`

Points to the byte array to be filled with the binary representation of `value`.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

Description

Data input to diagnostic services (for example, `ndWriteDataByLocalIdentifier`) is usually a byte array of binary data. If you have the data input description (for example, *byte 3 and 4 are engine RPM scaled as .25 * × RPM in Motorola representation*), you can use `ndConvertFromPhys` to convert the physical value to the byte stream by filling an appropriate `typeDescriptor` struct.

`ndConvertFromPhys` converts only the portion specified by one type descriptor to a binary representation. If your data input consists of several values, you can use `ndConvertFromPhys` multiple times on different parts of the byte array.

ndConvertToPhys

Purpose

Converts a binary representation of a value into its physical value using a type descriptor.

Format

```
void ndConvertToPhys(
       TD2 *typeDescriptor,
       unsigned char dataIn[],
       long len,
       double *value);
```

Input

`typeDescriptor`

A struct that specifies the conversion of the physical value to its binary representation:

```
typedef struct {
    long StartByte;
    long ByteLength;
    unsigned short ByteOrder;
    unsigned short DataType;
    double ScaleFactor;
    double ScaleOffset;
    } TD2;
```

`StartByte` gives the start byte of the binary representation in the `dataIn` record.

`ByteLength` is the number of bytes in the binary representation.

`ByteOrder` defines the byte order for multibyte representations. The values are:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

`DataType` is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths 4 or 8 are allowed.

`ScaleFactor` defines the physical value scaling:

Phys = (`ScaleFactor`) * (binary representation) + (`ScaleOffset`)

`ScaleOffset` (refer to `ScaleFactor`)

`dataIn`

Points to the byte array that contains the binary representation of `value`.

`len`

Must contain the `dataIn` array length.

Output

`value`

The physical value converted from the binary representation.

Description

Data output from diagnostic services (for example, `ndReadDataByLocalIdentifier`) is usually a byte stream of binary data. If you have a description of the data output (for example, *byte 3 and 4 are engine RPM scaled as .25 * × RPM in Motorola representation*), you can use `ndConvertToPhys` to extract the physical value from the byte stream by filling an appropriate `typeDescriptor` struct.

ndCreateExtendedCANIds

Purpose

Creates diagnostic CAN identifiers according to ISO 15765-2.

Format

```
void ndCreateExtendedCANIds (
       unsigned short addressingMode,
       unsigned short transportProtocol,
       unsigned char sourceAddress,
       unsigned char targetAddress,
       unsigned long *transmitID,
       unsigned long *receiveID);
```

Input

`addressingMode`

Specifies whether the ECU is physically or functionally addressed:

0: physical addressing

1: functional addressing

`transportProtocol`

Specifies whether normal or mixed mode addressing is used. The following values are valid:

0 **ISO TP—Normal Mode**. The ISO TP as specified in ISO 15765-2 is used; all eight data bytes of the CAN messages are used for data transfer.

1 **ISO TP—Mixed Mode**. The ISO TP as specified in ISO 15765-2 is used; the first data byte is used as address extension.

`sourceAddress`

The host (diagnostic tester) logical address.

`targetAddress`

The ECU logical address.

Output

`transmitID`

The generated CAN identifier for sending diagnostic request messages from the host to the ECU.

`receiveID`

The generated CAN identifier for sending diagnostic response messages from the ECU to the host.

Description

ISO 15765-2 specifies a method for creating (extended/29 bit) CAN identifiers for diagnostic applications given the addressing mode (physical/functional), the transport protocol (normal/mixed), and the 8-bit source and target addresses. This function implements the construction of these CAN identifiers. You can use them directly in the `ndOpenDiagnostic` function.

ndDiagnosticService

Purpose

Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this function.

Format

```
long ndDiagnosticService(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, `dataOut` returns no values, and `len2` returns 0. This parameter is passed by reference.

`dataIn`

Contains the request message byte sequence for the diagnostic service sent to the ECU.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Contains the response message byte sequence of the diagnostic service returned from the ECU.

`len2`

On input, `len2` must contain the number of bytes provided for the `dataOut` buffer. On output, it returns the number of valid data bytes in `dataOut`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDiagnosticService` is a generic routine to execute any diagnostic service. The request and response messages are fed unmodified to the `dataIn` input and retrieved from the `dataOut` output, respectively. No interpretation of the contents is done, with one exception: The error number is retrieved from a negative response, if one occurs. In this case, an error is communicated through the return value.

All specialized diagnostic services call `ndDiagnosticService` internally.

ndDTCToString

Purpose

Returns a string representation (such as *P1234*) for a 2-byte diagnostic trouble code (DTC).

Format

```
void ndDTCToString(
       unsigned long DTCNum,
       char DTCString[],
       long *len);
```

Input

`DTCNum`

The DTC number as returned in the DTCs structs of `ndReadDTCByStatus`, `ndReadStatusOfDTC`, `ndUDSReportDTCBySeverityMaskRecord`, `ndUDSReportDTCByStatusMask`, `ndUDSReportSeverityInformationOfDTC`, `ndUDSReportSupportedDTCs`, `ndOBDRequestEmissionRelatedDTCs`, or `ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle`.

Note This function converts only 2-byte DTCs. If you feed in larger numbers, the function returns garbage.

Output

`DTCString`

The DTC string representation.

`len`

On input, `len` must contain the `DTCString` array length (at least 6). On return, it contains the number of valid data bytes in the `DTCString` array.

Description

The SAE J2012 standard specifies a naming scheme for 2-byte DTCs consisting of one letter and four digits. Use `ndDTCToString` to convert the DTC numerical representation to this name.

ndGetProperty

Purpose

Gets a diagnostic global internal parameter.

Format

```
void ndGetProperty(
       unsigned short propertyID,
       unsigned long *propertyValue);
```

Input

`propertyID`

Defines the parameter whose value is to be retrieved:

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If you set this value to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** returns whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** returns the CAN frame content if filled with defined data or random data bytes.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** returns how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by `success` = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** is the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** is the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

Output

`propertyValue`

The requested property value.

Description

Use this function to request several internal diagnostic parameters, such as timeouts for the transport protocol. Use `ndSetProperty` to modify the parameters.

ndOBDOpen

Purpose

Opens a diagnostic session on a CAN port for OBD-II.

Format

```
long ndOBDOpen (
       char CANInterface[],
       unsigned long baudrate,
       unsigned long transmitID,
       unsigned long receiveID,
       TD1 *diagRefOut);
```

Input

`CANInterface`

Specifies the CAN interface on which the diagnostic communication should take place.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@ni_genie_nixnet* to the protocol string (for example, *CAN1@ni_genie_nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name as defined in your LabVIEW Project definition. The *lvbitx* filename represents the

filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

`baudrate`

The diagnostic communication baud rate.

`transmitID`

The CAN identifier for sending diagnostic request messages from the host to the ECU. Set to –1 (0xFFFFFFFF) for the default OBD CAN identifier.

`receiveID`

The CAN identifier for sending diagnostic response messages from the ECU to the host. Set to –1 (0xFFFFFFFF) for the default OBD CAN identifier.

Output

`diagRefOut`

A struct containing all necessary information about the diagnostic session. This is passed as a handle to all subsequent diagnostic functions, and you must close it using `ndCloseDiagnostic`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndOBDOpen` opens a diagnostic communication channel to an ECU for OBD-II. The CAN port specified as input is initialized, and a handle to it is stored (among other internal data) into the `diagRefOut` struct, which serves as reference for further diagnostic functions.

If the `transmitID` and `receiveID` parameters are set to –1, communication is first tried on the default 11-bit OBD CAN identifiers; if that fails, the default 29-bit OBD CAN identifiers are tried. If that also fails, an error is returned.

If valid `transmitID` or `receiveID` parameters (11-bit or 29-bit with bit 29 set) are given, communication is tried on these identifiers. If that fails, an error is returned.

In general, it is not necessary to manipulate the `diagRefOut` struct contents.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx*.
- **CAN256**—uses virtual NI-CAN interface 256.

Using NI-XNET hardware:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1,c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at location `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO:

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

ndOpenDiagnostic

Purpose

Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.

Format

```
long ndOpenDiagnostic(
       char CANInterface[],
       unsigned long baudrate,
       unsigned short transportProtocol,
       unsigned long transmitID,
       unsigned long receiveID,
       TD1 *diagRefOut);
```

Input

`CANInterface`

Specifies the CAN interface on which the diagnostic communication should take place.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@ni_genie_nixnet* to the Protocol string (for example, *CAN1@ni_genie_nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name

as defined in your LabVIEW Project definition. The *lvbitx* filename represents the filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

`baudrate`

The diagnostic communication baud rate.

`transportProtocol`

Specifies the transport protocol for transferring the diagnostic service messages over the CAN network. The following values are valid:

0 **ISO TP—Normal Mode**. The ISO TP as specified in ISO 15765-2 is used; all eight data bytes of the CAN messages are used for data transfer.

1 **ISO TP—Mixed Mode**. The ISO TP as specified in ISO 15765-2 is used; the first data byte is used as address extension.

2 **VW TP 2.0**.

`transmitID`

The CAN identifier for sending diagnostic request messages from the host to the ECU.

`receiveID`

The CAN identifier for sending diagnostic response messages from the ECU to the host.

Output

`diagRefOut`

A struct containing all necessary information about the diagnostic session. This is passed as a handle to all subsequent diagnostic functions, and you must close it using `ndCloseDiagnostic`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndOpenDiagnostic` opens a diagnostic communication channel to an ECU. This function initializes the CAN port specified as input and stores a handle to it (among other internal data) into `diagRefOut`, which serves as reference for further diagnostic functions.

No communication to the ECU takes place at this point. To open a diagnostic session on the ECU, call `ndStartDiagnosticSession` or `ndUDSDiagnosticSessionControl`.

In general, you do not need to manipulate the `diagRefOut` struct contents, except if you use the **ISO TP—Mixed Mode** transport protocol, in which case you must store the address extensions for transmit and receive in the appropriate members of that struct.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx*.
- **CAN256**—uses virtual NI-CAN interface 256.

Using NI-XNET hardware:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1,c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at location `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

ndSetProperty

Purpose

Sets a diagnostic global internal parameter.

Format

```
void ndSetProperty(
       unsigned short propertyID,
       unsigned long propertyValue);
```

Input

`propertyID`

Defines the parameter whose value is to be modified:

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If you set this value to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** specifies whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** specifies the CAN frame content, filled with defined data or random data.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** specifies how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by `success` = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** defines the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** is the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

`propertyValue`

The requested property value.

Output

None.

Description

Use this function to set several internal diagnostic parameters, such as timeouts for the transport protocol. Use `ndGetProperty` to read them out.

ndStatusToString

Purpose

Returns a description for an error code.

Format

```
void ndStatusToString(
       long errorCode,
       char message[],
       long *len);
```

Input

`errorCode`

The status code (return value) of any other diagnostic functions.

Output

`message`

Returns a descriptive string for the error code.

`len`

On input, `len` must contain the `message` array length. On return, it contains the number of valid data bytes in the `message` array.

Description

When the status code returned from an Automotive Diagnostic Command Set function is nonzero, an error or warning is indicated. This function obtains an error/warning description for debugging purposes.

The return code is passed into the `errorCode` parameter. The `len` parameter indicates the number of bytes available in the string for the description. The description is truncated to size `len` if needed, but a size of 1024 characters is large enough to hold any description. The text returned in `message` is null-terminated, so you can use it with ANSI C functions such as `printf`. For C or C++ applications, each Automotive Diagnostic Command Set function returns a status code as a signed 32-bit integer. The following table summarizes the Automotive Diagnostic Command Set use of this status.

Status Code Use

Status Code	Definition
Negative	Error—Function did not perform the expected behavior.
Positive	Warning—Function performed as expected, but a condition arose that may require attention.
Zero	Success—Function completed successfully.

The application code should check the status returned from every Automotive Diagnostic Command Set function. If an error is detected, close all Automotive Diagnostic Command Set handles and exit the application. If a warning is detected, you can display a message for debugging purposes or simply ignore the warning.

The following code shows an example of handling Automotive Diagnostic Command Set status during application debugging.

```
Status = ndOpenDiagnostic ("CAN0", 500000, 0, 0x7E0, 0x7E8,
&MyDiagHandle);
PrintStat (status, "ndOpenDiagnostic");
```

where the function `PrintStat` has been defined at the top of the program as:

```
void PrintStat(mcTypeStatus status, char *source)
{
    char statusString[1024];
    long len = sizeof(statusString);
    if (status != 0)
    {
        ndStatusToString(status, statusString, &len);
        printf("\n%s\nSource = %s\n", statusString, source);
        if (status < 0)
        {
            ndCloseDiagnostic(&MyDiagHandle);
            exit(1);
        }
    }
}
```

ndVWTPConnect

Purpose

Establishes a connection channel to an ECU using the VW TP 2.0.

Format

```
long ndVWTPConnect(
       TD1 *diagRef,
       unsigned long channelID,
       unsigned char applicationType);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`channelID`

Defines the CAN identifier on which the ECU responds for this connection. The ECU defines the ID on which the host transmits.

`applicationType`

Specifies the communication type that takes place on the communication channel. For diagnostic applications, specify *KWP2000 (1)*. The other values are for manufacturer-specific purposes.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

For the VW TP 2.0, you must establish a connection to the ECU before any diagnostic communication can occur. This function sets up a unique communication channel to an ECU that you can use in subsequent diagnostic service requests.

You must maintain the communication link thus created by periodically (at least once a second) calling `ndVWTPConnectionTest`.

No equivalent exists for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

ndVWTPConnectionTest

Purpose

Maintains a connection channel to an ECU using the VW TP 2.0.

Format

```
long ndVWTPConnectionTest(
       TD1 *diagRef);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

For the VW TP 2.0, you must periodically maintain the connection link to the ECU, so that the ECU does not terminate it. You must execute this periodic refresh at least once per second.

This function sends a Connection Test message to the ECU and evaluates its response, performing the necessary steps to maintain the connection.

There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

ndVWTPDisconnect

Purpose

Terminates a connection channel to an ECU using the VW TP 2.0.

Format

```
long ndVWTPDisconnect(
       TD1 *diagRef);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

For the VW TP 2.0, you must disconnect the ECU connection link to properly terminate communication to the ECU. This function sends the proper disconnect messages and unlinks the communication.

Use `ndVWTPConnect` the create a new connection to the same ECU.

There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

KWP2000 Services

ndClearDiagnosticInformation

Purpose

Executes the ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndClearDiagnosticInformation(
       TD1 *diagRef,
       unsigned short groupOfDTC,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network related DTCs
0xFF00	All DTCs

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function clears the diagnostic information on the ECU memory. `groupOfDTC` specifies the type of diagnostic trouble codes to be cleared on the ECU memory.

For further details about this service, refer to the ISO 14230-3 standard.

ndControlDTCSetting

Purpose

Executes the ControlDTCSetting service. Modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndControlDTCSetting(
       TD1 *diagRef,
       unsigned short groupOfDTC,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network related DTCs
0xFF00	All DTCs

`dataIn`

Specifies application-specific data that control DTC generation.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndDisableNormalMessageTransmission

Purpose

Executes the DisableNormalMessageTransmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).

Format

```
long ndDisableNormalMessageTransmission(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndECUReset

Purpose

Executes the ECUReset service. Resets the ECU.

Format

```
long ndECUReset(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the reset mode:

Hex	Description
01	**PowerOn** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the client (tester) re-establishes communication.
02	**PowerOnWhileMaintainingCommunication** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the server (ECU) maintains communication with the client (tester).
03–7F	**Reserved**
80–FF	**ManufacturerSpecific** This range of values is reserved for vehicle manufacturer-specific use.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests the ECU to perform an ECU reset effectively based on the `mode` value content. The vehicle manufacturer determines when the positive response message is sent.

ndEnableNormalMessageTransmission

Purpose

Executes the EnableNormalMessageTransmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).

Format

```
long ndEnableNormalMessageTransmission(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndInputOutputControlByLocalIdentifier

Purpose

Executes the InputOutputControlByLocalIdentifier service. Modifies the ECU I/O port behavior.

Format

```
long ndInputOutputControlByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char mode,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the I/O to be manipulated. The values are application specific.

`mode`

Defines the I/O control type. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ReportCurrentState

4: ResetToDefault

5: FreezeCurrentState

7: ShortTermAdjustment

8: LongTermAdjustment

`dataIn`

Defines application-specific data for this service.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific data for this service.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by `localID`.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadDataByLocalIdentifier

Purpose

Executes the ReadDataByLocalIdentifier service. Reads an ECU data record.

Format

```
long ndReadDataByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the data to be read. The values are application specific.

Output

`dataOut`

Returns the data record from the ECU. If you know the record data description, you can use the `ndConvertToPhys` function to interpret it.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests data record values from the ECU identified by the `localID` parameter.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadDTCByStatus

Purpose

Executes the ReadDiagnosticTroubleCodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndReadDTCByStatus(
       TD1 *diagRef,
       unsigned char mode,
       unsigned short groupOfDTC,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Defines the type of DTCs to be read. The values are application specific. The usual values are:

2: AllIdentified

3: AllSupported

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network related DTCs
0xFF00	All DTCs

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
        unsigned long DTC;
        unsigned long Status;
        unsigned long AddData;
        } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear

6 testNotCompletedThisMonitoringCycle

7 warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function reads diagnostic trouble codes by status from the ECU memory. If you set the optional `groupOfDTC` parameter to the above specified codes, the ECU reports DTCs only with status information based on the functional group selected by `groupOfDTC`.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadECUIdentification

Purpose

Executes the ReadECUIdentification service. Returns ECU identification data.

Format

```
long ndReadECUIdentification(
       TD1 *diagRef,
       unsigned char mode,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the type of identification information to be returned. The values are application specific.

Output

`dataOut`

Returns the ECU identification data.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests identification data from the ECU. `mode` identifies the type of identification data requested. The ECU returns identification data that `dataOut` can access. The `dataOut` format and definition are vehicle manufacturer specific.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadMemoryByAddress

Purpose

Executes the ReadMemoryByAddress service. Reads data from the ECU memory.

Format

```
long ndReadMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address from which data are read. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be read.

Output

`dataOut`

Returns the ECU memory data.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests ECU memory data identified by the `address` and `size` parameters. The `dataOut` format and definition are vehicle manufacturer specific. `dataOut` includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadStatusOfDTC

Purpose

Executes the ReadStatusOfDiagnosticTroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndReadStatusOfDTC(
       TD1 *diagRef,
       unsigned short groupOfDTC,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network related DTCs
0xFF00	All DTCs

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
        unsigned long DTC;
        unsigned long Status;
        unsigned long AddData;
        } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function reads diagnostic trouble codes from the ECU memory. If you specify `groupOfDTC`, the ECU reports DTCs based only on the functional group selected by `groupOfDTC`.

For further details about this service, refer to the ISO 14230-3 standard.

ndRequestRoutineResultsByLocalIdentifier

Purpose

Executes the RequestRoutineResultsByLocalIdentifier service. Returns results from an ECU routine.

Format

```
long ndRequestRoutineResultsByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the routine from which this function retrieves results. The values are application specific.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests results (for example, exit status information) referenced by `localID` and generated by the routine executed in the ECU memory.

For further details about this service, refer to the ISO 14230-3 standard.

ndRequestSeed

Purpose

Executes the SecurityAccess service to retrieve a seed from the ECU.

Format

```
long ndRequestSeed(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char seedOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

Output

`seedOut`

Returns the seed from the ECU.

`len`

On input, `len` must contain the `seedOut` array length. On return, it contains the number of valid data bytes in the `seedOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndSendKey

Purpose

Executes the SecurityAccess service to send a key to the ECU.

Format

```
long ndSendKey(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char keyIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

`keyIn`

Defines the key data to be sent to the ECU.

`len`

Must contain the number of valid data bytes in `keyIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndStartDiagnosticSession

Purpose

Executes the StartDiagnosticSession service. The ECU is set up in a specific diagnostic mode.

Format

```
long ndStartDiagnosticSession(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the diagnostic mode into which the ECU is brought. The values are application specific.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function enables different ECU diagnostic modes. The possible diagnostic modes are not defined in ISO 14230 and are application specific. A diagnostic session starts only if communication with the ECU is established. For more details about starting communication, refer to ISO 14230-2. If no diagnostic session is requested after `ndOpenDiagnostic`, a default session is enabled automatically in the ECU. The default session supports at least the following services:

- The StopCommunication service (refer to `ndCloseDiagnostic` and the ISO 14230-2 standard).
- The TesterPresent service (refer to `ndTesterPresent` and the ISO 14230-3 standard).

ndStartRoutineByLocalIdentifier

Purpose

Executes the StartRoutineByLocalIdentifier service. Executes a routine on the ECU.

Format

```
long ndStartRoutineByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the routine to be started. The values are application specific.

`dataIn`

Defines application-specific input parameters for the routine.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function starts a routine in the ECU memory. The ECU routine starts after the positive response message is sent. The routine stops until the `ndStopRoutineByLocalIdentifier` function and corresponding service are issued. The routines could be either tests that run instead of normal operating code or routines enabled and executed with the normal operating code running. In the first case, you may need to switch the ECU to a specific diagnostic mode using `ndOpenDiagnostic` or unlock the ECU using the SecurityAccess service prior to using `ndStartRoutineByLocalIdentifier`.

For further details about this service, refer to the ISO 14230-3 standard.

ndStopDiagnosticSession

Purpose

Executes the StopDiagnosticSession service. Returns the ECU to normal mode.

Format

```
long ndStopDiagnosticSession(
       TD1 *diagRef,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function disables the current ECU diagnostic mode. A diagnostic session stops only if communication with the ECU is established and a diagnostic session is running. If no diagnostic session is running, the default session is active. `ndStopDiagnosticSession` cannot disable the default session. If the ECU stops the current diagnostic session, it performs the necessary action to restore its normal operating conditions. Restoring the normal ECU operating conditions may include resetting all controlled actuators activated during the diagnostic session being stopped, and resuming all normal ECU algorithms. You should call `ndStopDiagnosticSession` before disabling communication with `ndCloseDiagnostic`, but only if you previously used `ndStartDiagnosticSession`.

ndStopRoutineByLocalIdentifier

Purpose

Executes the StopRoutineByLocalIdentifier service. Stops a routine on the ECU.

Format

```
long ndStopRoutineByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the routine to be stopped. The values are application specific.

`dataIn`

Defines application-specific input parameters for the routine.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function stops a routine in the ECU memory referenced by `localID`.

For further details about this service, refer to the ISO 14230-3 standard.

ndTesterPresent

Purpose

Executes the TesterPresent service. Keeps the ECU in diagnostic mode.

Format

```
long ndTesterPresent(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

Keep calling `ndTesterPresent` within the ECU timeout period if no other service is executed.

ndWriteDataByLocalIdentifier

Purpose

Executes the WriteDataByLocalIdentifier service. Writes a data record to the ECU.

Format

```
long ndWriteDataByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the data to be read. The values are application specific.

`dataIn`

Defines the data record to be written to the ECU. If you know the record data description, use `ndConvertFromPhys` to generate this record.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function performs the WriteDataByLocalIdentifier service and writes RecordValues (data values) to the ECU. `dataIn` identifies the data values to be transmitted. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 14230-3 standard.

ndWriteMemoryByAddress

Purpose

Executes the WriteMemoryByAddress service. Writes data to the ECU memory.

Format

```
long ndWriteMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address to which data are written. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be written.

`dataIn`

Defines the memory block to be written to the ECU.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This VI performs the KWP2000 WriteDataByAddress service and writes RecordValues (data values) to the ECU. `address` and `size` identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 14230-3 standard.

UDS (DiagOnCAN) Services

ndUDSClearDiagnosticInformation

Purpose

Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSClearDiagnosticInformation(
       TD1 *diagRef,
       unsigned long groupOfDTC,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The values are application specific. The following value has a special meaning:

0xFFFFFF All DTCs

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function clears the diagnostic information on the ECU memory. Depending on the value of `groupOfDTC`, the ECU is requested to clear the corresponding DTCs. The `groupOfDTC` values are application specific.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSCommunicationControl

Purpose

Executes the UDS CommunicationControl service. Switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off.

Format

```
long ndUDSCommunicationControl(
       TD1 *diagRef,
       unsigned char type,
       unsigned char communicationType,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`type`

Indicates whether transmission/reception is to be switched on/off. The usual values are:

00: enableRxAndTx

01: enableRxAndDisableTx

02: disableRxAndEnableTx

03: disableRxAndTx

`communicationType`

A bitfield indicating which application level is to be changed. The usual values are:

01: application

02: networkManagement

You can change more than one level at a time.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the UDS CommunicationControl service and switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off. The `type` and `communication type` parameters are vehicle manufacturer specific (one OEM may disable the transmission only, while another OEM may disable the transmission and reception based on vehicle manufacturer specific needs). The request is either transmitted functionally addressed to all ECUs with a single request message, or transmitted physically addressed to each ECU in a separate request message.

ndUDSControlDTCSetting

Purpose

Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) behavior.

Format

```
long ndUDSControlDTCSetting(
       TD1 *diagRef,
       unsigned char type,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`type`

Specifies the control mode:

1: on

2: off

`dataIn`

Specifies application-specific data that control DTC generation.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndUDSDiagnosticSessionControl

Purpose

Executes the UDS DiagnosticSessionControl service. The ECU is set up in a specific diagnostic mode.

Format

```
long ndUDSDiagnosticSessionControl(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the diagnostic mode into which the ECU is brought. The values are application specific. The usual values are:

01: defaultSession

02: ECUProgrammingSession

03: ECUExtendedDiagnosticSession

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndUDSECUReset

Purpose

Executes the UDS ECUReset service. Resets the ECU.

Format

```
long ndUDSECUReset(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the reset mode:

Hex	Description
01	hardReset
02	keyOffOnReset
03	softReset
04	enableRapidPowerShutDown
05	disableRapidPowerShutDown

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests the ECU to perform an ECU reset effectively based on the `mode` parameter value content. The vehicle manufacturer determines when the positive response message is sent. Depending the value of `mode`, the corresponding ECU reset event is executed as a hard reset, key off/on reset, soft reset, or other reset.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSInputOutputControlByIdentifier

Purpose

Executes the UDS InputOutputControlByIdentifier service. Modifies ECU I/O port behavior.

Format

```
long ndUDSInputOutputControlByIdentifier(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char mode,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the I/O to be manipulated. The values are application specific.

`mode`

Defines the I/O control type. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ResetToDefault

2: FreezeCurrentState

3: ShortTermAdjustment

`dataIn`

Defines application-specific data for this service.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific data for this service.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by the `ID` parameter.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReadDataByIdentifier

Purpose

Executes the UDS ReadDataByIdentifier service. Reads an ECU data record.

Format

```
long ndUDSReadDataByIdentifier(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the data to be read. The values are application specific.

Output

`dataOut`

Returns the ECU data record. If you know the record data description, use `ndConvertToPhys` to interpret this record.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests data record values from the ECU identified by the `ID` parameter.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReadMemoryByAddress

Purpose

Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Format

```
long ndUDSReadMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address from which data are read. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be read.

Output

`dataOut`

Returns the ECU memory data.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests memory data from the ECU identified by the `address` and `size` parameters. The `dataOut` format and definition are vehicle manufacturer specific. `dataOut` includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportDTCBySeverityMaskRecord

Purpose

Executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSReportDTCBySeverityMaskRecord(
       TD1 *diagRef,
       unsigned char severityMask,
       unsigned char status,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`severityMask`

Defines the status of DTCs to be read. The values are application specific.

`status`

Defines the status of DTCs to be read. The values are application specific.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC.

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportDTCByStatusMask

Purpose

Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSReportDTCByStatusMask(
       TD1 *diagRef,
       unsigned char statusMask,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`statusMask`

Defines the status of DTCs to be read. The values are application specific.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportSeverityInformationOfDTC

Purpose

Executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs) are read.

Format

```
long ndUDSReportSeverityInformationOfDTC(
       TD1 *diagRef,
       unsigned long DTCMaskRecord,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCMaskRecord`

Defines the status of DTCs to be read. The values are application specific.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC.

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU memory.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportSupportedDTCs

Purpose

Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSReportSupportedDTCs(
       TD1 *diagRef,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads all supported DTCs from the ECU memory.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSRequestDownload

Purpose

Initiates a download of data to the ECU.

Format

```
long ndUDSRequestDownload (
       TD1 *diagRef,
       unsigned long memoryAddress,
       unsigned long memorySize,
       unsigned char memoryAddressLength,
       unsigned char memorySizeLength,
       unsigned char dataFormatIdentifier,
       unsigned long *blockSize,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`memoryAddress`

Defines the memory address to which data are to be written.

`memorySize`

Defines the size of the data to be written.

`memoryAddressLength`

Defines the number of bytes of the `memoryAddress` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

`memorySizeLength`

Defines the number of bytes of the `memorySize` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

`dataFormatIdentifier`

Defines the compression and encryption scheme for the data blocks written to the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

Output

`blockSize`

Returns the number of data bytes to be transferred to the ECU in subsequent `ndUDSTransferData` requests.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndUDSRequestDownload` initiates the download of a data block to the ECU. This is required to set up the download process; the actual data transfer occurs with subsequent `ndUDSTransferData` requests. The transfer must occur in blocks of the size this service returns (the `blockSize` parameter). After the download completes, use the `ndUDSRequestTransferExit` service to terminate the process.

ndUDSRequestSeed

Purpose

Executes the UDS SecurityAccess service to retrieve a seed from the ECU.

Format

```
long ndUDSRequestSeed(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char seedOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

Output

`seedOut`

Returns the seed from the ECU.

`len`

On input, `len` must contain the `seedOut` array length. On return, it contains the number of valid data bytes in the `seedOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndUDSRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndUDSSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndUDSRequestTransferExit

Purpose

Terminates a download/upload process.

Format

```
long ndUDSRequestTransferExit (
       TD1 *diagRef,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`dataIn`

Defines a data record to be written to the ECU as part of the termination process. The meaning is implementation dependent; this may be a checksum or a similar verification instrument.

`len`

Must be set to the buffer size for the `dataIn` parameter.

Output

`dataOut`

Returns a memory data block from the ECU as part of the termination process. The meaning is implementation dependent; this may be a checksum or a similar verification instrument.

`len2`

Must be set to the buffer size for the `dataOut` parameter. On return, it contains the actual data size returned in `dataOut`.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndUDSRequestTransferExit` terminates a download or upload process initialized with `ndUDSRequestDownload` or `ndUDSRequestUpload`.

ndUDSRequestUpload

Purpose

Initiates an upload of data from the ECU.

Format

```
long ndUDSRequestUpload (
       TD1 *diagRef,
       unsigned long memoryAddress,
       unsigned long memorySize,
       unsigned char memoryAddressLength,
       unsigned char memorySizeLength,
       unsigned char dataFormatIdentifier,
       unsigned long *blockSize,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`memoryAddress`

Defines the memory address from which data are to be read.

`memorySize`

Defines the size of the data to be read.

`memoryAddressLength`

Defines the number of bytes of the `memoryAddress` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

`memorySizeLength`

Defines the number of bytes of the `memorySize` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

`dataFormatIdentifier`

Defines the compression and encryption scheme used for the data blocks written to the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

Output

`blockSize`

Returns the number of data bytes to be transferred from the ECU in subsequent `ndUDSTransferData` requests.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndUDSRequestUpload` initiates the upload of a data block from the ECU. This is required to set up the upload process; the actual data transfer occurs with subsequent `ndUDSTransferData` requests. The transfer must occur in blocks of the size that this service returns (the `blockSize` parameter). After the download completes, use the `ndUDSRequestTransferExit` service to terminate the process.

ndUDSRoutineControl

Purpose

Executes the UDS RoutineControl service. Executes a routine on the ECU.

Format

```
long ndUDSRoutineControl(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char mode,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the routine to be started. The values are application specific.

`mode`

Defines the operation mode for this service:

1: Start Routine

2: Stop Routine

3: Request Routine Results

Other values are application specific.

`dataIn`

Defines application-specific input parameters for the routine.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the UDS RoutineControl service and launches an ECU routine, stops an ECU routine, or requests ECU routine results from the ECU.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSSendKey

Purpose

Executes the UDS SecurityAccess service to send a key to the ECU.

Format

```
long ndUDSSendKey(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char keyIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

`keyIn`

Defines the key data to be sent to the ECU.

`len`

Must contain the number of valid data bytes in `keyIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndUDSRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndUDSSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndUDSTesterPresent

Purpose

Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.

Format

```
long ndUDSTesterPresent(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

Keep calling `ndUDSTesterPresent` within the ECU timeout period if no other service is executed.

ndUDSTransferData

Purpose

Transfers data to/from the ECU in a download/upload process.

Format

```
long ndUDSTransferData (
       TD1 *diagRef,
       unsigned char *blockSequenceCounter,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`blockSequenceCounter`

Used to number the data blocks to be transferred to/from the ECU. The block sequence counter value starts at 01 hex with the first `ndUDSTransferData` request that follows the `ndUDSRequestDownload` or `ndUDSRequestUpload` service. Its value is incremented by 1 for each subsequent `ndUDSTransferData` request. At the value of FF hex, the block sequence counter rolls over and starts at 00 hex with the next `ndUDSTransferData` request.

The block sequence counter is updated automatically, and the updated value is returned.

`dataIn`

Defines the data block to be written to the ECU.

For a download, this is a memory data block to be downloaded to the ECU.

For an upload, the meaning is implementation dependent.

`len`

Must be set to the buffer size for the `dataIn` parameter.

Output

`blockSequenceCounter`

Returns the updated value of the block sequence counter (refer to the description in the *Input* section).

`dataOut`

Returns the memory data from the ECU.

For a download, this may contain a checksum or similar verification instrument; the meaning is implementation dependent.

For an upload, this is a memory data block uploaded from the ECU.

`len2`

Must be set to the buffer size for the `dataOut` parameter. On return, it contains the actual data size returned in `dataOut`.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndUDSTransferData` executes the data transfer of a download process (initiated with a previous `ndUDSRequestDownload` request) or an upload process (initiated with a previous `ndUDSRequestUpload` request). The data transfer must occur in blocks of the size that has been returned in the block size parameter of the respective request service. After the data transfer completes, terminate the operation by calling the `ndUDSRequestTransferExit` service.

ndUDSWriteDataByIdentifier

Purpose

Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.

Format

```
long ndUDSWriteDataByIdentifier(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the data to be read. The values are application specific.

`dataIn`

Defines the data record written to the ECU. If you know the record data description, use `ndConvertFromPhys` to generate this record.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function performs the UDS service WriteDataByIdentifier and writes RecordValues (data values) into the ECU. `dataIn` identifies the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSWriteMemoryByAddress

Purpose

Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.

Format

```
long ndUDSWriteMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address to which data are written. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be written.

`dataIn`

Defines the memory block to be written to the ECU.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function performs the UDS service WriteMemoryByAddress and writes RecordValues (data values) into the ECU. `address` and `size` identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 15765-3 standard.

OBD (On-Board Diagnostics) Services

ndOBDClearEmissionRelatedDiagnosticInformation

Purpose

Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related diagnostic trouble codes (DTCs) in the ECU.

Format

```
long ndOBDClearEmissionRelatedDiagnosticInformation(
       TD1 *diagRef,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestControlOfOnBoardDevice

Purpose

Executes the OBD Request Control Of On-Board Device service. Modifies ECU I/O port behavior.

Format

```
long ndOBDRequestControlOfOnBoardDevice(
       TD1 *diagRef,
       unsigned char TID,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`TID`

Defines the test identifier of the I/O to be manipulated. The values are application specific.

`dataIn`

Defines application-specific data for this service.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific data for this service.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestCurrentPowertrainDiagnosticData

Purpose

Executes the OBD Request Current Powertrain Diagnostic Data service. Reads an ECU data record.

Format

```
long ndOBDRequestCurrentPowertrainDiagnosticData(
       TD1 *diagRef,
       unsigned char PID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`PID`

Defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

Output

`dataOut`

Returns the ECU data record. If you know the record data description, use `ndConvertToPhys` to interpret this record. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestEmissionRelatedDTCs

Purpose

Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).

Format

```
long ndOBDRequestEmissionRelatedDTCs(
       TD1 *diagRef,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle

Purpose

Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.

Format

```
long ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle(
       TD1 *diagRef,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestOnBoardMonitoringTestResults

Purpose

Executes the OBD Request On-Board Monitoring Test Results service. Reads an ECU test data record.

Format

```
long ndOBDRequestOnBoardMonitoringTestResults(
       TD1 *diagRef,
       unsigned char OBDMID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`OBDMID`

Defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

Output

`dataOut`

Returns the ECU test data record. If you know the record data description, use `ndConvertToPhys` to interpret this record. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestPermanentFaultCodes

Purpose

Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.

Format

```
long ndOBDRequestPermanentFaultCodes(
       TD1 *diagRef,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the DTCs array length in elements. On return, it contains the number of valid elements in the DTCs array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestPowertrainFreezeFrameData

Purpose

Executes the OBD Request Powertrain Freeze Frame Data service. Reads an ECU data record stored while a diagnostic trouble code occurred.

Format

```
long ndOBDRequestPowertrainFreezeFrameData(
       TD1 *diagRef,
       unsigned char PID,
       unsigned char nFrame,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`PID`

Defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

`nFrame`

The number of the freeze frame from which the data are to be retrieved.

Output

`dataOut`

Returns the ECU data record. If you know the record data description, use `ndConvertToPhys` to interpret this record. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestVehicleInformation

Purpose

Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.

Format

```
long ndOBDRequestVehicleInformation(
       TD1 *diagRef,
       unsigned char infoType,
       unsigned char *nItems,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`infoType`

Defines the type of information to be read. The SAE J1979 standard defines the values.

Output

`nItems`

The number of data items (not bytes) this service returns.

`dataOut`

Returns the ECU vehicle information. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

A

Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Technical support at ni.com/support includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support as well as exclusive access to on demand training modules via the Services Resource Center. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

    For information about other technical support options in your area, visit ni.com/services, or contact your local office at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index

A

C

D

E

G

H

I

K

L

N

O

P

R

S

T

U

V

W