CANoe and CANalyzer as diagnostic tools

Version 1.4
2012/07/17

Application Note AN-IND-1-001

Author(s) Thomas R. Schmidt, Michael Webler
Restrictions Public Document
Abstract This application gives an introduction into working with diagnostics in CANoe/CANalyzer. It presents the basic technical aspects and possibilities with the Diagnostic Feature Set, complements the help file of CANoe/CANalyzer and may be used as a tutorial.

Table of Contents

1.0 Overview 3
1.1 Introduction 3
1.2 Diagnostic components 4
2.0 Diagnostics in CANoe 6
2.1 ISO TP support 6
2.2 Diagnostic Descriptions 6
2.2.1 CDD – CANdela Diagnostic Description 6
2.2.2 ODX – Open Diagnostic Data Exchange 6
2.2.3 MDX – Multiplex Diagnostic Data Exchange 6
2.3 Trace window 7
2.4 Diagnostic Feature Set 7
2.4.1 Interactive Diagnostics Console window 7
2.4.2 Fault Memory window 8
2.4.3 Diagnostic Session Control window 9
2.4.4 ECU or tester simulations using CAPL 9
2.4.5 Test modules using CAPL (CANoe only) 10
2.4.7 Physical Network Request and Network Diagnostic Description 10
2.4.8 Functional Group Requests 11
2.5 Access to diagnostics features via COM 11
2.6 Standard Diagnostics - Generic Diagnostic Descriptions for KWP2000 and UDS 11
2.7 Basic Diagnostics 11
2.8 OBD II window 12
2.9 Security Access handling 12
3.0 First steps 13
3.1.1 Add a Diagnostic Description 13
3.1.2 Configure the Diagnostic Description 13
3.2 Usage of Diagnostics Console and Fault Memory window 14
3.2.1 Send a diagnostic request and receive a response 14
3.2.2 Read fault memory 14
3.2.3 Physical Network Requests and Functional Group Requests 15
3.3 Usage of the CAPL Browser 15
3.4 Necessary preparations when using CAPL to simulate diagnostics 15
3.4.1 Define the Network Node to be simulated in the database 15
3.4.2 Add a Network Node to the Simulation Setup 15
3.4.3 Configure the Network Node in Simulation Setup 15
3.4.4 Add a diagnostic description and assign it to the network node 16

Contact Information: www.vector.com or ++49-711-80 670-0

3.4.5 Add the CAPL callback interface 17
3.4.6 Mandatory variables 18
3.4.7 Debug level 18
3.4.8 Set the diagnostic target 18
3.4.9 Qualifier for Diagnostics request 19
3.4.10 Qualifier for Diagnostics response 19
3.4.11 Create a diagnostic request 19
3.4.12 Create a diagnostic response 19
3.4.13 Work with parameters 20
3.4.14 Negative Response handling 21
3.5 Combine Test Feature Set and Diagnostic Feature Set 22
3.5.1 Difference between a Prog Node (ECU) and Test Module 22
3.5.2 Timeout handling 22
3.5.3 A simple Diagnostic Tester 22
3.5.4 Diagnostics tests with XML patterns 23
4.0 Advanced examples 24
4.1 ECU simulation of “Response Pending” 24
4.2 Modifying the length of a diagnostic object 25
4.3 Fill diagnostic content 25
4.4 Fault injection 26
4.5 Access a node via a gateway simulation 26
5.0 Common mistakes 28
6.0 Abbreviations 30
7.0 References 30
8.0 Additional Resources 30
9.0 Contacts 31

1.0 Overview

1.1 Introduction

Diagnostics is used to configure, maintain, support, control and extend an ECU before or after it is installed in a system, e.g. a vehicle. Diagnostics is usually performed in a request – response scheme: a tester (client) sends a request to an ECU (or even more than one ECU) and the ECU (server) responds by sending a “positive response message” containing the requested information, or a “negative response” indicating the reason for the negative response.

The purpose of this application note is to give a general introduction into working with diagnostics in the Vector tool CANoe. The basic technical aspects and possibilities (“first steps”) with the Diagnostic Feature Set will be presented. Examples are used to get the test engineer started with testing diagnostics in CANoe.

This document is a complement to the online help in CANoe and should be used as a tutorial to learn the “first steps” of the Diagnostic Feature Set. For more detailed information about the Diagnostic Feature Set, please refer to the CANoe help file and to CANoe demo applications, both of which come with a standard CANoe installation.

Note:	The functionality described here refers to CANoe (and CANalyzer, unless otherwise noted) version 8.0. For older program versions application notes can be requested from the Vector support (cf. 9.0).

1.2 Diagnostic components

The following table lists the names of the components relevant for diagnostics in CANoe, how to activate them and where to find more information.

Component	Description	Activation	More information
OSEK TP DLL	Implementation of the ISO TP for CANoe simulation nodes	Database attribute “NodeLayerModules”; Simulation setup: node configuration, tab “Modules”	In folder Doc: CanTP_Manual.PDF (new API) OSEK_TP_E.pdf (old API)
ISO TP Observer	Displays TP information in the trace window for the CAN messages used by the ISO TP	Menu: “Configuration-> Diagnostics/ISO TP configuration”, page “ISO TP Observer”	Online help: “Diagnostics/ISO TP Configuration: ISO TP Observer”
KWP 2000 interpreter	Extension of the ISO TP Observer that interprets the transported data according to KWP 2000	Like ISO TP Observer, check box “Interpretation according to KWP2000”	Online help: “Diagnostics/ISO TP Configuration: ISO TP Observer”
Diagnostics interpreter	Extension of the ISO TP Observer, interpret the transported data according to the available diagnostic specification(s)	Menu: “Configuration-> Diagnostics/ISO TP configuration”, corresponding network in which Diagnostic Descriptions can be loaded	Online help: “Diag. / ISO TP Obs.: Diagnostic Descriptions”
Interactive Diagnostics Console	Direct sending of requests defined in a Diagnostic Description, display of responses	By assigning a Diagnostic Description to any of the available networks (see Diagnostics interpreter above) a corresponding Diagnostics Console window is made available, and can be accessed via the “View” menu	Online help: “Diagnostics Console: Overview”
Fault Memory	Direct access to an ECU’s fault memory	By assigning Diagnostic Descriptions to any of the available networks (see Diagnostics interpreter above) a corresponding Fault Memory window is made available, and can be accessed via the “View” menu	Online help: “Fault Memory window: Overview”
Diagnostic Session Control	Easy switching of the session state (e.g. Default, Extended, Programming),helpful especially in combination with services protected by security access	By assigning a Diagnostic Description to any of the available networks (see Diagnostics interpreter above) a corresponding Diagnostic Session Control window is made available, and can be accessed via the “View” menu	Online help: “Diagnostic Session Control: Overview”
OBD II window	Support of on-board diagnostics	By choosing the addressing mode (11 bit Normal or 29 bit NormalFixed) at the page “OBD-II Functionality” (Menu:	Online help: “Diagnostics Description Settings: OBD-II Functionality”

		“Configuration-> Diagnostics/ISO TP configuration”)	
CAPL extensions for diagnostics	Specialized CAPL functions to access diagnostics objects specified via Diagnostic Description(s)	The extended CAPL API is available after assigning at least one Diagnostic Description to the CANoe configuration	Online help: “Diagnostics: Expanded Functions in CAPL”

2.0 Diagnostics in CANoe

CANoe can be used in all steps of developing ECUs and performing diagnostics on them:

- Design of the diagnostic functionality (system simulation)
- Implementation of diagnostic functionality in an ECU (remaining bus simulation)
  → ISO/OSEK TP DLL and CAPL extensions
- Specification-/Integration-/Regression tests
  → CAPL extensions and CANoe test support. ISO/OSEK TP DLL only necessary if TP fault injection requested.
- Analysis of real ECU communication
  → ISO TP Observer OR KWP2000 Interpreter OR Diagnostics Interpreter
- Perform diagnostics of ECUs with integrated tester functionality
  → Diagnostics Console
- Error search
  → Fault Memory window
- On-board diagnostics
  → OBD II window

2.1 ISO TP support

CANoe includes the ISO TP Observer, which interprets messages sent over the CAN bus according to the ISO Transport Protocol ISO/TF2 and displays the results in the Trace window in clear text.

It also includes an implementation of the transport protocol that enables easy sending and receiving of diagnostic objects. This implementation is realized by a node layer DLL that comes with every CANoe standard installation and takes care of transport protocol specific functions such as segmentation, flow control etc.

To enable transport layer interpretation it is needed to activate the ISO TP Observer. Please refer to the online help in CANoe on how to activate the observer.

To enable the TP functionality for a simulated node, please refer to paragraph 0.

2.2 Diagnostic Descriptions

To work with diagnostics in CANoe Diagnostic Descriptions have to be assigned to the configuration. Diagnostic Descriptions may be in the shape of (generic) **CDD**, **ODX** (PDX) or **MDX** files. All of these types are referred to as “Diagnostic Descriptions” in this application note. It is possible to mix those file types within one CANoe configuration.

2.2.1 CDD – CANdela Diagnostic Description

CANdela Diagnostic Descriptions (CDD) files are databases for diagnostic data, comparable to the .dbc-file used for CAN messages and signals. The CDD files are created in the Vector tool CANdelaStudio and can be used in CANoe and CANalyzer for symbolic access and interpretation of diagnostic services and parameters.

2.2.2 ODX – Open Diagnostic Data Exchange

ODX files (Open Diagnostic Data Exchange) also carry diagnostic data. This data can be divided into several ODX files and stored in PDX files (ODX archives). The usage of ODX files is similar to the usage of CDD files.

2.2.3 MDX – Multiplex Diagnostic Data Exchange

MDX files (Multiplex Diagnostic Exchange) is an OEM-specific format carrying diagnostic data as well. The usage of MDX files is similar to the usage of ODX archive files.

Note: The below mentioned features can only be used after including a Diagnostic Description into the CANoe/CANalyzer configuration.

2.3 Trace window

A Diagnostic Description allows tracing diagnostic services (requests/responses) and their parameters in a symbolic fashion. You can expand the requests/responses in the same way as with ordinary bus messages:

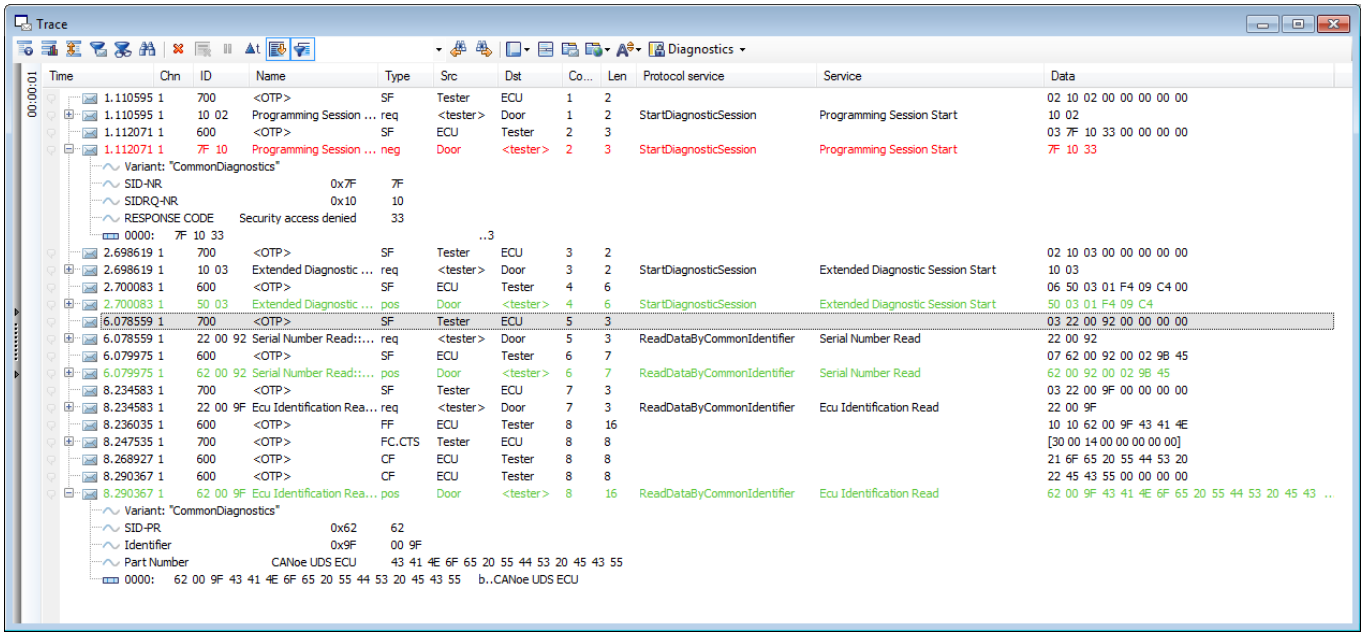


Figure 1: Trace window

2.4 Diagnostic Feature Set

The Vector Diagnostic Feature Set includes several functions that are necessary for development, test and application of ECUs with/via diagnostics.

Based on the Diagnostic Description, the Diagnostics Console provides interactive access to all diagnostic services. Diagnostic requests can be selected, parameterised and displayed with their dedicated response.

The Fault Memory Console provides quick and easy access to the fault memory of an ECU.

Apart from CANoe the Diagnostic Features Set is also included in the Vector products CANape MC+D and CANdito. Thereby the complete development process is supported identically.

2.4.1 Interactive Diagnostics Console window

The Interactive Diagnostics Console fetches its information from the Diagnostic Description and presents an easy way to select a diagnostic request, manipulate its parameters and to send the request. The response received is presented together with its parameters.

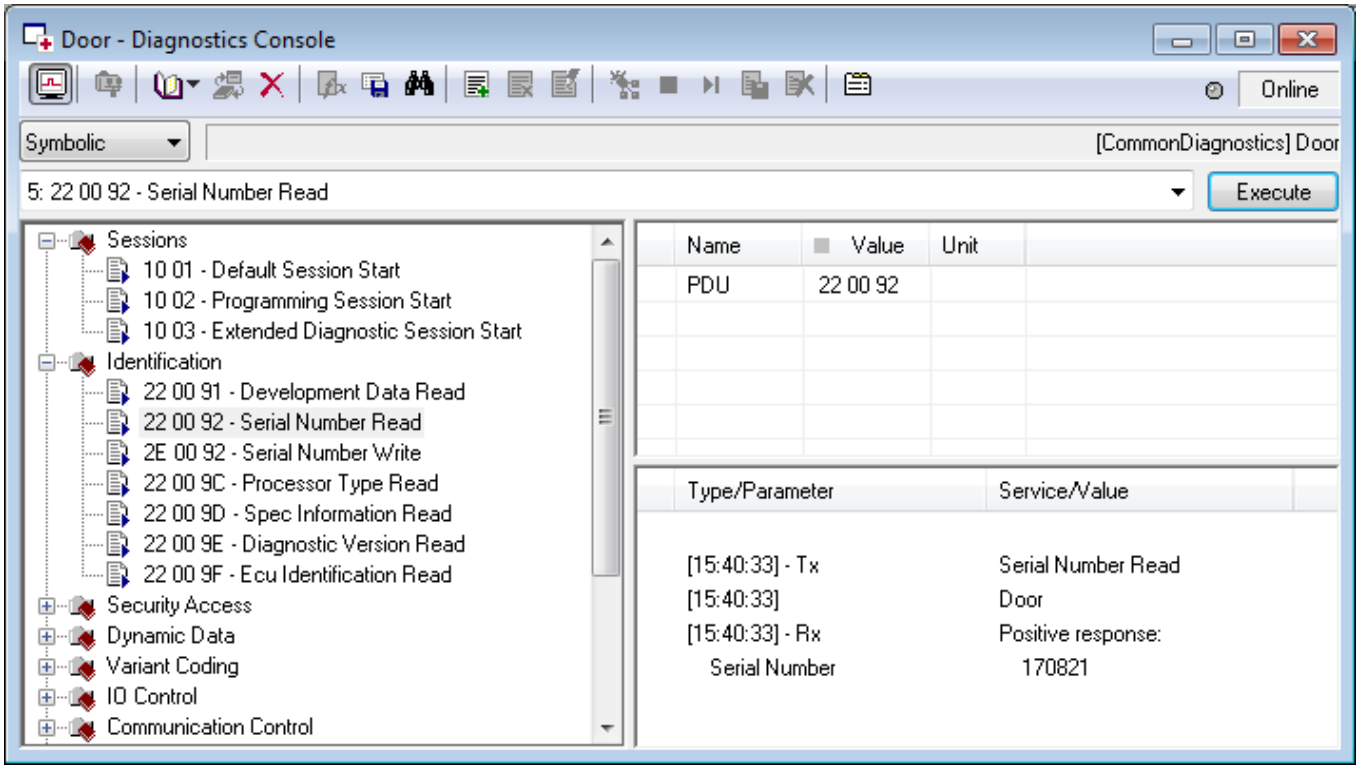


Figure 2: Interactive Diagnostics Console window

2.4.2 Fault Memory window

The Fault Memory window presents a possibility to read out the fault memory list of an ECU once or cyclically.

It is also possible to configure one of the generic CDD files to access the fault memory of an ECU, and the requests sent to the ECU can be defined as raw byte streams.

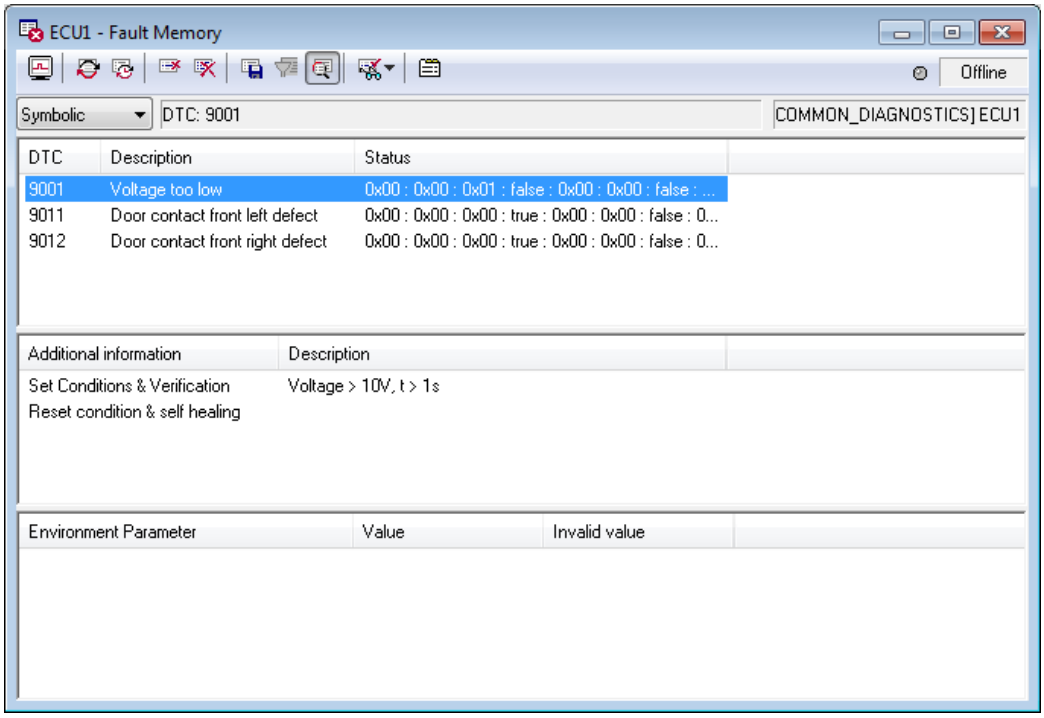


Figure 3: Fault Memory window

2.4.3 Diagnostic Session Control window

With the Diagnostic Session Control window, the user can easily switch between different session states like Default, Extended, or Programming session.

In combination with a security DLL assigned to the corresponding Diagnostic Description (see chapter 2.9 for details), it is possible to switch the session state without having to care about the computation and exchange of security keys. After switching the session state via the Diagnostic Session Control window, the user can easily execute diagnostic services from the Diagnostics Console which are only accessible within sessions protected by a certain security level.

2.4.4 ECU or tester simulations using CAPL

CAPL can be used to simulate an ECU or a diagnostic tester even if no real ECU or tester is present. The diagnostics commands in CAPL enable access to the diagnostics services and data using symbolic names that were defined in the CANdela description file. The simulation has to react on the requests or responses from its counterpart (real or simulated by CANoe/CANalyzer) that are received and processed in appropriate event procedures. It is even possible to implement interactive tester applications where the user accesses the diagnostic functionality via a GUI (panel).

Note:	CANalyzer provides only limited simulation possibilities. However, for simple use cases, it is possible to access the diagnostics services and data using symbolic names like in CANoe and use them in CAPL code.

2.4.5 Test modules using CAPL (CANoe only)

In CANoe, it is possible to implement automated tests that run without user interaction and perform a sequence of sending requests and processing of responses. The result of such a test can be written to a report file (in XML/HTML format).

Common tests can even be implemented without a CAPL program in an XML test specification.

2.4.6 Symbolic selection dialog for diagnostics objects and parameters

In order to simplify the specification of diagnostics qualifiers for requests, responses and parameters, these parameters and diagnostic objects - defined in Diagnostic Descriptions - can be inserted into CAPL code via drag & drop from the symbol explorer in the CPL browser. Simply drag the object named with the diagnostics primitive, service, parameter or target qualifier and drop it at the current cursor location into the CAPL program.

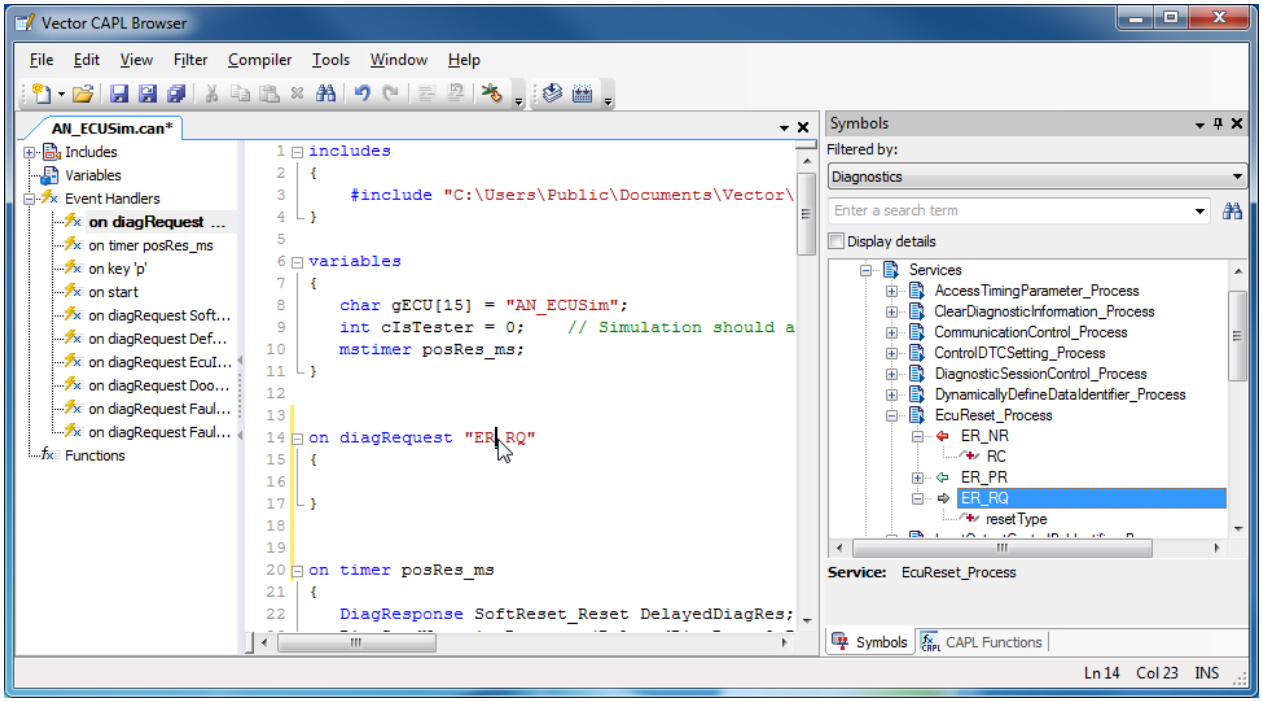


Figure 4: Symbolic selection of diagnostic objects and parameters in CAPL Browser

Note that the content of the Diagnostic Description will be displayed in a structured form (classes, instances and services). The dialog in Figure 4is also used to select diagnostics parameters to be displayed in data and graphics windows.

2.4.7 Physical Network Request and Network Diagnostic Description

It is possible to specify one Diagnostic Description per network as “Network Diagnostic Description”, i.e. select “Physical Network Requests” as its usage, indicating that all ECUs on the network implement the Diagnostic Description’s content as common subset. The Network Diagnostic Description can then be selected as target in CAPL programs, and when a request is sent, it will be sent as a physical request to every ECU. The responses received from the ECUs will be processed individually in the tester, and the interpretation will be based on the concrete Diagnostic Description for each ECU.

It is also possible to open the Diagnostics Console, the session control window and the fault memory window for the Network Diagnostic Description, offering a simple interface to send a request to every ECU on the CAN bus. The responses will be displayed in the console's trace window, while the DTCs reported will be listed in the fault memory window with their originating ECU. DTCs can then be deleted individually, or all fault memories can be deleted with the press of one button.

2.4.8 Functional Group Requests

Similar to Physical Network Requests, selecting "Functional Group Requests" as the usage of a Diagnostic Description, the diagnostic requests for it will be sent using the transport protocol parameters for functional requests so that all ECUs which react on this functional request will send responses. The responses received from the ECUs will be processed individually in the tester, and the interpretation will be based on the concrete Diagnostic Description for each ECU.

Like for Physical Network Requests, it is possible to open the Diagnostics Console and the fault memory window with a Diagnostic Description configured for Functional Group Requests. The responses will be displayed in the console's trace window, while the DTCs reported will be listed in the fault memory window with their originating ECU.

2.5 Access to diagnostics features via COM

The COM server has an additional interface that allows external application written in VisualBasic, VisualBasicScript etc. to access diagnostics features in CANoe. This allows the simple implementation of special functionality, e.g. a manufacturer dependent process. More information can be found in the Technical Reference on the COM interface included in the CANoe online help.

2.6 Standard Diagnostics - Generic Diagnostic Descriptions for KWP2000 and UDS

Keyword Protocol 2000 (KWP2000, ISO 14230) and Unified Diagnostic Services (UDS, ISO 14229) are standard diagnostic protocols used by many OEMs. Please note that most manufacturers use diagnostics specifications that differ from the standards!

Three "Generic CDDs" are provided. They describe diagnostics on the level of the standards. This has the following advantages:

- All mechanisms implemented for concrete Diagnostic Descriptions can be applied for the standard CDDs, though certain restrictions apply, e.g. the parameter definitions cannot be as precise as for concrete Diagnostic Descriptions.
- It is possible to set the communication parameters in the configuration dialog, removing the need to enter them in a database or code them into a CAPL program.
- The Diagnostics Console and fault memory window can be used to get fast access to ECUs.
- The interpretation of the transmitted data can also be parameterized with the Diagnostic Description(s).

For KWP 2000 *interpretation only* no Diagnostic Description (or DLL) is necessary, though it is recommended to configure a generic CDD. If the TP level settings can be read from a CAN database, it is sufficient to activate the check box "Interpret data according to KWP2000" on the "ISO TP observer" page of the .

More details about this protocol support can be found in the online help in CANoe. You can also look at the standard definitions by opening the generic CDDs [2] with the CANdelaStudio Viewer application provided with CANoe.

2.7 Basic Diagnostics

Using Basic Diagnostics, you can exchange diagnostic information with an ECU also without a Diagnostic Description (CDD/ODX/MDX).

You can describe simple diagnostic services (UDS & KWP) with the Basic Diagnostics Editor and afterwards send/receive the defined requests/responses on CAN, LIN, FlexRay, K-Line and via DoIP using the Diagnostics Console, CAPL and in CAPL test modules/ test case libraries. Additionally, CANoe/CANalyzer supports symbolic interpretation of the Basic Diagnostics services and their parameters in the Trace window.

For simple applications, Basic Diagnostics thus represents an extension to the process-oriented approach with CANdela Diagnostic Descriptions.

In order to use Basic Diagnostics, you need to add a “Basic Diagnostics ECU” (KWP or UDS) to the CANoe diagnostic configuration. Outside of the measurement, you can now define/modify the diagnostic services in the Basic Diagnostics Editor and commit them to the Diagnostics Console. Saving the CANoe configuration also commits these changes.

After the measurement starts, you can call the services in the Diagnostics Console, in CAPL and also in CAPL test modules/ test case libraries. On the same network, you are able to work with ECUs configured using Diagnostic Description files as well as with multiple Basic Diagnostics ECUs at the same time.The TP parameters of these control units must be different though.

2.8 OBD II window

On-Board Diagnostics, or OBD, in an automotive context, is a generic term referring to a vehicle's self-diagnostic and reporting capability. OBD systems give the vehicle owner or a repair technician access to state of health information for various vehicle sub-systems.

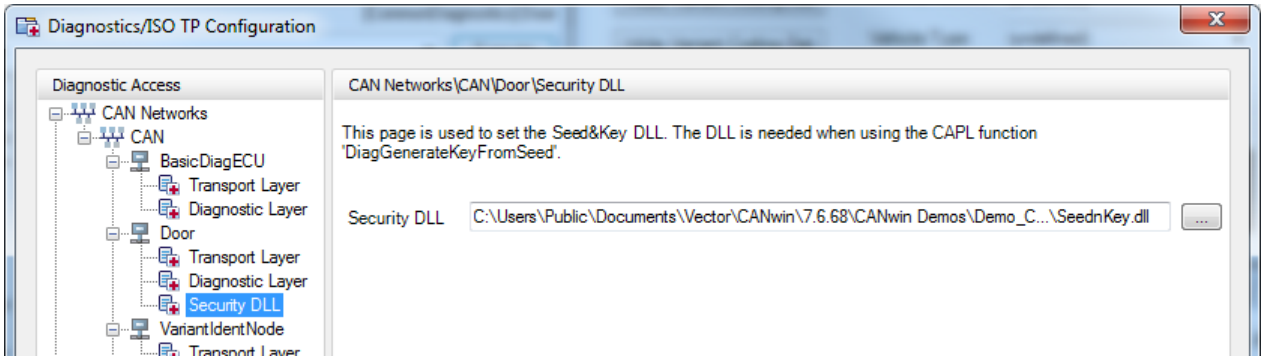
OBD implementations use a standardized fast digital communications port to provide realtime data in addition to a standardized series of diagnostic trouble codes, or DTCs, which allow one to rapidly identify and remedy malfunctions within the vehicle.

The OBD II window in CANoe supplies access to this diagnostic functionality.

2.9 Security Access handling

In order to execute locked diagnostic functions in the control unit (ECU), the tester requires a key to unlock the ECU. The key is calculated from a seed which is received from the ECU in a diagnostic response. The algorithm used to calculate the key can be implemented as a dll (security dll).

The DiagGenerateKeyFromSeed function encapsulates the concrete implementation of the key algorithm in the provider-dependent SeedKey DLL. The SeedKey DLL must be configured in the diagnostics configuration dialog (Diagnostics / ISO TP Configuration|<Network> | <Diagnostic Description>|Security DLL) for each diagnostics description that shall be used with DiagGenerateKeyFromSeed:



For information on requirements on the security key dll as well as on usage in older versions, please consult the online help in CANoe.

3.0 First steps

The diagnostic features in CANoe may be used for either tracing diagnostic communication on the bus or for acting as a diagnostic tester (via the Diagnostics Console or via CAPL). Furthermore they provide capabilities for simulating the diagnostic services of an ECU.

All these use cases demand that a Diagnostic Description (a diagnostic database) is used, and it may be necessary to use the ISO TP DLL (or a different TP implementation) to transfer data.

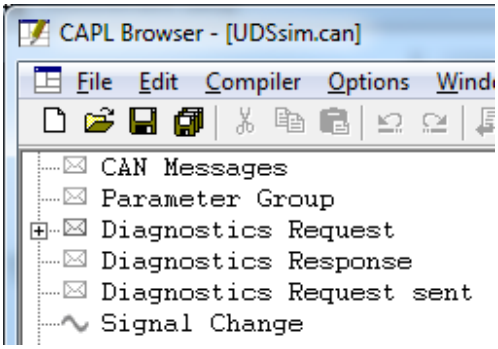
3.1 Usage of diagnostic database files

3.1.1 Add a Diagnostic Description

Diagnostic Descriptions (CDD/ODX/MDX) describe the diagnostic data (services and parameters), i.e. they are diagnostic databases.

A Diagnostic Description is added to the CANoe configuration in menu “Configuration | Diagnostics/ISO TP Configuration…”. Note that you can add Standard Diagnostics Descriptions (“generic CDDs”) for KWP and UDS standards here [2].

After adding a Diagnostic Description to the CANoe configuration there will be additional event categories in the CAPL Browser: Diagnostics Request, Diagnostics Response and Diagnostics Response sent:



If CANoe is connected to a real vehicle with ongoing diagnostic communication it will now be possible to have symbolic interpretation in the trace window.

When a Diagnostic Description is added to the CANoe configuration, the Diagnostics Console, Fault Memory and Session Control windows will appear (they can also be accessed via the View menu). This makes it possible to select a service in the Diagnostics Console, send the request, receive the response, and inspect certain parameters.

Note:	For Standard Diagnostics Descriptions, no Session Control Window will be displayed since the session handling is done individually by each OEM.

3.1.2 Configure the Diagnostic Description

After adding a Diagnostic Description (CDD/ODX/MDX, i.e. *.cdd, *.odx, *.pdx or *.mdx file) to a network in the Diagnostics/ISO TP Configuration dialogue and selecting the Diagnostic Description or one of its sub-components (e.g. Transport Layer, Diagnostic layer), you can change the settings for this particular Diagnostic Description.

Depending on the intended usage of the Diagnostic Description, you have several options: If only TP data should be interpreted, it is sufficient to choose “interpretation only”. If you choose “Diagnostic Tester (“send only”), you can use the Diagnostics Console and Fault Memory Windows, but it is not possible to use the Diagnostic Description in a CAPL node. If ECU simulation (CAPL) functionality is wanted, it is important to assign the Diagnostic Description to the node (ECU) that it describes.

Note:	An ECU can be selected only if a network node in the simulation setup is available and assigned to a node defined in the database of the corresponding network.

Additionally, you can choose “(Physical Network Requests)” or “(Functional Group Requests)” as the function of a Diagnostic Description if all ECUs on the network implement the contents of the Diagnostic Description as a common subset. The Diagnostics Console and fault memory window can then send a request to all ECUs on the bus as a “physical network request” (i.e. send the requests via the physical address to all the configured network nodes in sequence) or as a “functional group request” (i.e. broadcast using the functional address).

The “Variant” setting will determine the set of services you see in the Diagnostics Console, e.g. if the “Common” variant is chosen, the user might not find services that are only present for other variants. The “Target group” setting additionally restricts the set of services offered in the Diagnostics Console to those services intended for a specific user group.

Selecting an interface here will determine which TP parameters are available for configuration:

- The “Interface” combo box shows the interfaces defined in the Diagnostic Description. Each interface defines an ISO TP addressing mode and address parameters that are used as default. Note that the addressing mode of an interface cannot be changed – chose a different interface instead.
- For a Network Diagnostic Description, i.e. if you defined “Physical Network Requests” as its usage, no TP parameters have to be set since the TP parameters of the individual ECUs will be used.
- If the “VAG Addons packet“ (version 1.10 or later) is installed, it is possible to select the interface called "VWTP 2.0 (CANoe)". You then have to enter the correct TP parameters on the page “Transport Layer (VW TP 2.0)”.
- If a node from a LIN database file is selected, the corresponding parameters can be defined on the page “LIN settings”.
- If a node from a FlexRay database is selected, the FlexRay TP parameters can be defined in the pages “FrTP Parameter”, “FrTP Functional Parameter” and “FrTP Timing Parameter”.
- Since for MOST ECUs, only interpretation of the diagnosis messages is possible, there are no dedicated MOST TP parameters for MOST nodes.

For further details of the configuration dialog please refer to the online help.

3.2 Usage of Diagnostics Console and Fault Memory window

3.2.1 Send a diagnostic request and receive a response

When you add a Diagnostic Description to your CANoe configuration and choose “OK”, the Diagnostics Console and Fault Memory Window for that Diagnostic Description will automatically become visible. You can easily send a request by selecting it in the Diagnostics Console “Explorer-like” tree structure. The parameters in the request can also be selected in an easy manner by choosing a value in a drop-down menu or writing a value (e.g. serial part number of ECU). The response will be presented accordingly in this window.

If you do not have a real ECU with implemented diagnostics, you should create a simulated node in CANoe and implement relevant functionality in CAPL (please see paragraph 0)

3.2.2 Read fault memory

With the Fault Memory Window you can easily read out the fault memory of an ECU. Depending on the Diagnostic Description, a KWP2000 standard request ($18 02) or a UDS request ($19 02) is used to read out the trouble codes. Additionally, the services specified in the Diagnostic Description can be used if the manufacturer scheme

can be recognized by CANoe/CANalyzer (via the qualifier paths). It is also possible to specify the requests explicitly.

3.2.3 Physical Network Requests and Functional Group Requests

It is also possible to send physical requests to all ECUs defined on a network or Functional Group Requests from the Diagnostics Console and the Fault Memory Window. If a Diagnostic Description is configured to be a “network description”, i.e. if you defined “Physical Network Requests” as its usage in the configuration dialog, both windows will open for that Diagnostic Description. In analogy to the “Physical Network Requests”, you are able to configure “Functional Group Requests” as the usage of a Diagnostic Description; this will result in sending a functional request (“broadcast”) to all ECUs defined on a network.

3.3 Usage of the CAPL Browser

Three additional event categories are present after adding a Diagnostic Description to the CANoe configuration; “Diagnostics Request”, “Diagnostics Response” and “Diagnostics Request sent”. The actual requests and responses are accessed through their qualifier paths as described in the Diagnostic Description with the syntax <class>::< instance>::<service>.

Example:

```
diagRequest START_SESSION::DEFAULT_SESSION::StartSession request;
```

This code snippet initializes the variable “request” using the qualifier path as a request to start the default diagnostic session.

3.4 Necessary preparations when using CAPL to simulate diagnostics

Note: This chapter only applies to CANoe.

3.4.1 Define the Network Node to be simulated in the database

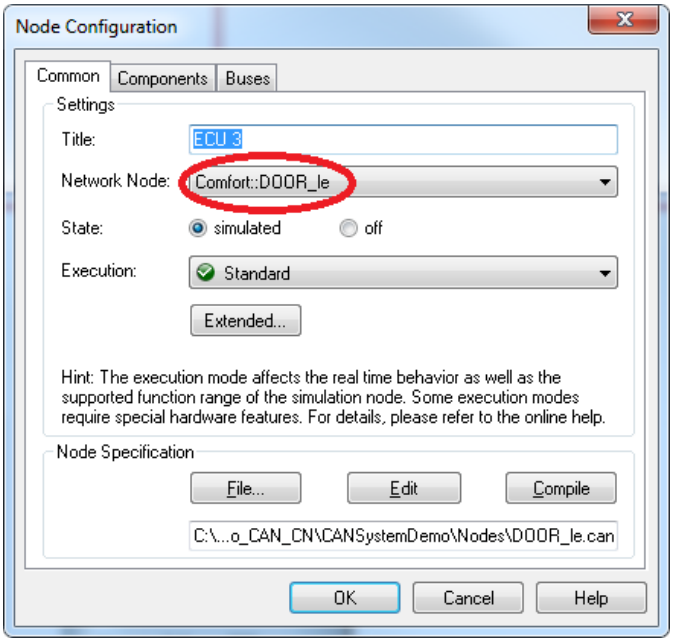
As the first step, a Network Node which shall be used for simulating diagnostic functionality needs to be available in the corresponding database. If necessary, you may add it using the CANdb++ editor.

3.4.2 Add a Network Node to the Simulation Setup

Next, you need to add a Network Node to the Simulation Setup. In order to do this, right-click on the bus line in the Simulation Setup window and choose the context menu “Insert Network Node”.

3.4.3 Configure the Network Node in Simulation Setup

In the configuration of the Network Node just added to the Simulation Setup, you then need to assign the Network Node defined in the database. To do this, select the database Network Node you defined as described in chapter 3.4.1 in the configuration dialog of this node:



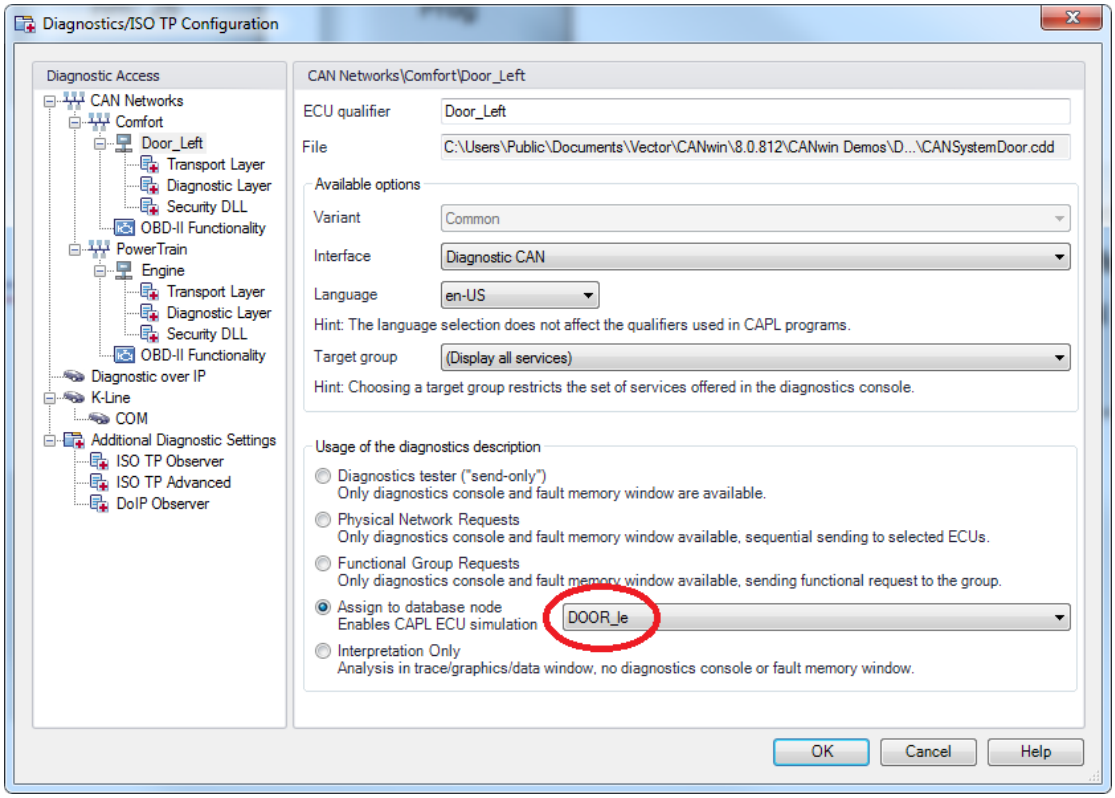
For your convenience it is possible to work with diagnostic objects in CAPL even if these objects exceed the size of one CAN frame. E.g. a diagnostic response with 24 data bytes is still treated as one object - the diagnostic response. This is possible due to the built-in transport protocol included with every CANoe installation (cf. to [1] for further explanation). The OSEK transport protocol is implemented in a dynamic link library (DLL) named “osek_tp.dll”. In order to use this DLL, you need to add it as a component to the network node.

The assignment can be made either in the .dbc-file via the “NodeLayerModules” attribute or by configuring the node in the simulation setup. Select the node in simulation setup and right-click the mouse button – this presents a context menu. Choose “Configuration…” and then the tab “Components”. Add the transport layer DLL by browsing for it (it is found in the exec32 directory of your CANoe installation).

Note:	The “osek_tp.dll” uses several callback functions (the so-called CAPL callback interface) which need to be implemented in the network node (see section 3.4.5 for details).

3.4.4 Add a diagnostic description and assign it to the network node

Next, you should add a diagnostic description. To to so, select the menu “Configuration | Diagnostics/ISO-TP Configuration…” and add the diagnostic description to the desired network. Afterwards, you need to set the usage of the added diagnostic description to “Assign to database node” and select the database node you defined in the first step (see section 3.4.1 for details):



3.4.5 Add the CAPL callback interface

To simulate an ECU or a Diagnostic Tester in CAPL it is important to include certain callback functions (referenced as the CAPL callback interface, CCI) to allow the diagnostics layer in CANoe to access the transport protocol layer properly.

Note: *Tester* nodes can use the standard communication channels created for the diagnostics controls. Therefore you do *not* need to implement the CAPL callback interface in a tester node/module that does not need direct access to the transport layer.

The mandatory callback functions for the osek_tp.dll (the transport layer calls into CAPL that forwards the events to the diagnostics layer, acc. to [1]) are:

```
CanTp_SendCon()             // returns the number of data bytes successfully sent, replaces the
                            // deprecated function OSEKTL_DataCon()
CanTp_ErrorInd()            // error message from the transport layer, replaces the deprecated
                            // function OSEKTL_ErrorInd()
CanTp_ReceptionInd()        // returns the number of data bytes received, replaces the deprecated
                            // function OSEKTL_DataInd()
CanTp_FirstFrameInd()       // no return value, indicates that the first frame of a transport
                            // protocol frame sequence was received
```

The following functions are called by CANoe to automatically configure the transport protocol and process data (the diagnostics layer calls into CAPL that forwards the calls to the transport layer):

```
_Diag_DataRequest()         // places a transmit request
_Diag_SetChannelParameters() // Instructs the CAPL node to configure the TP layer
_Diag_SetupChannelReq()     // Tester only: the tester should open a "channel" to the ECU
```

Notes: There is also the deprecated ISO TP specific callback interface (prefix _OSEKTL_...).
TP layer parameter values can be retrieved with the function DiagGetCommParameter().

To include these functions, the following procedure is recommended:

- Include the following file into your CAPL code (you need to replace <Version Nr> by your CANoe/CANalyzer version):
  "%PUBLIC%\Documents\Vector\CANwin\<Version Nr>\CANwin Demos\Demo_CAN_CN\Diagnostics\UDSSim\Nodes\CCI_Implementation.cin"

- Declare the mandatory variables used by the implementation in "`CCI_Implementation.cin`" as described in section 3.4.6.

3.4.6 Mandatory variables

The sample implementation of the CCI uses a global variable gECU. If a function of the transport layer interface is used, the variable is used to indicate the node in which the function was executed; therefore it should be defined in the variables section and assigned a reasonable value.

Examples:
```
char gECU[10] = "ECU";
char gECU[15] = "Tester node";
```

Additionally, the CCI needs the information whether the simulation should act as a tester or not. For this purpose, the constant cIsTester needs to be defined.

Examples:
```
int cIsTester = 0;    // Simulation should act as an ECU
int cIsTester = 1;    // Simulation should act as a tester
```

3.4.7 Debug level

The debug level of the ISO TP functions can be controlled by a parameter to the function setWriteDbgLevel(). To set the debug level to verbose use setWriteDbgLevel(1) or to set the debug level to quiet use setWriteDbgLevel(0).

Example:
```
on start
{
   setWriteDbgLevel(0);
}
```

3.4.8 Set the diagnostic target

Note:	This paragraph only applies to tester simulations. In an ECU simulation node, DiagSetTarget() should not be used.

If you simulate a diagnostic tester that should send requests and receive responses from a specific (simulated or real) ECU, you must set the target name in the tester CAPL code. The target is usually set in the "on start"-handler in the CAPL Browser, but may be changed later.

Example:
```
on start
{
  if( 0 != DiagSetTarget( "ECU" )) write( "Error setting target!" );
}
```

The string "ECU" should be changed to the actual ECU qualifier contained in the Diagnostic Description. Note that you can edit this ECU identifier in the "Diagnostics/ISO TP Configuration" dialog.

3.4.9 Qualifier for Diagnostics request

In order to add an event procedure for a specific diagnostic request (e.g. in a simulated ECU), right-click on “Event handlers” and select the respective handler under “New handlers | Diagnostics | on diagRequest <new request>”. This code will be created:

```
on diagRequest NewRequest
{
}
```

Use the diagnostics symbol explorer (see chapter 2.4.6) to enter the qualifier path via drag and drop.

Example:

```
on diagRequest START_SESSION::DEFAULT_SESSION::StartSession
{
}
```

3.4.10 Qualifier for Diagnostics response

In order to add an event procedure for a specific diagnostic response (e.g. in a simulated Tester), right-click on “Event handlers” and select the respective handler under “New handlers | Diagnostics | on diagResponse <new response>”. This code will be created:

```
on diagResponse NewResponse
{
}
```

Use the diagnostics symbol explorer (see chapter 2.4.6) to enter the qualifier path via drag and drop.

3.4.11 Create a diagnostic request

To create a request that should be sent (e.g. from a Diagnostic Tester) you can create a function where you create and send a request.

Example:

```
StartSession()
{
  diagRequest START_SESSION::DEFAULT_SESSION::StartSession req;
  // Send the request as a complete object (TP takes care of segmentation)
  DiagSendRequest ( req );
}
```

3.4.12 Create a diagnostic response

Note: This paragraph only applies to ECU simulations.

Usually a response is sent on reception of the event „diagRequest“, i.e. when the specific request arrives. By using the keyword „this“ the response object will reflect the request object by referring to the actual qualifier. Note how the diagnostic response is treated as an object rather than one or several CAN message(s). Due to the TP functionality a response can be sent in one function call even if the response object should exceed one CAN frame.

Example:

```
on diagRequest START_SESSION::DEFAULT_SESSION::StartSession
{
  // Create a response to this request
  diagResponse this resp;
  // Send the response
  DiagSendResponse( resp);
}
```

3.4.13 Work with parameters

The parameters can be accessed (read and written) symbolically as they are described in the Diagnostic Description. Diagnostic parameters are divided into two groups; simple and complex parameters. The two groups have corresponding Set- and Get- functions. Simple parameters are parameters that have fixed offsets in the diagnostic object. Complex parameters are parameters that have varying offsets since they are contained within container parameters, e.g. a list of DTCs (Diagnostic Trouble Codes).

Below is an example of a simple parameter that has the name “Voltage Terminal 15” (note the blank characters!) in the Diagnostic Description. Since the names are language dependent, the *qualifier* has to be used (accessible via the symbol explorer, see chapter 2.4.6) in CAPL, i.e. "Voltage_Terminal_15".

Example simple parameter:

```
on diagRequest DEVICE_CONTROL::InputOutput::Set
{
  diagResponse this resp;
  // Set simple parameters in response to 0
  DiagSetParameter ( resp, "Voltage_Terminal_15", 0 );
  DiagSetParameter ( resp, "Interior_Temperature", 0 );
  DiagSendResponse ( resp );
}
```

Below is an example on how to work with a complex parameter e.g. a list of DTCs. The DTCs together with their status masks are grouped in a list called “List of DTC” in the Diagnostic Description.

First of all, some memory has to be reserved for the DTCs. The response object is created with an iteration counter of 0, i.e. indicating that no DTCs will follow. As a first step, the response object has to be enlarged, i.e. the iteration counter has to be set to the number of DTCs that should be returned. In CANoe 5.2 this will automatically reserve the requested space, i.e. it is not necessary to resize the object.
Note that sometimes no iteration counter is specified. In these cases the number of DTCs to follow would be determined by the actual length of the response; here the total length in bytes has to be specified for the resize operation.

After enough memory is available, the DTCs can be initialized step by step, i.e. the ECU simulation sets the parameter for the DTCs it wants to report to the tester.

Example complex parameter:

```
on diagRequest FAULT_MEMORY::FAULT_MEMORY::ReadAll
{
  diagResponse this resp;
  // Set the number of DTCs returned
  DiagSetParameter( resp, “NUMBER_OF_DTC”, 2);
  // Create memory to hold the DTCs
  DiagResize( resp);     // Note: NOT necessary in CANoe 5.2!
  // Set complex parameters in response
  // Set the first DTC to a hex value
  DiagSetComplexParameter ( resp, "List_of_DTC", 0, "DTC", 0xFAFAF );
  // Set the status mask of the DTC to true
  DiagSetComplexParameter ( resp, "List_of_DTC", 0, "DtcStatusDataType.ConfirmedDTC", 1 );
  // Set next DTC
  DiagSetComplexParameter ( resp, "List_of_DTC", 1, "DTC", 0xCFCFC );
  DiagSetComplexParameter ( resp, "List_of_DTC", 1, "DtcStatusDataType.ConfirmedDTC", 1 );
  …
  DiagSendResponse ( resp );
}
```

The following example shows how to set the size of the response if the actual Diagnostic Description does not contain the parameter “NUMBER_OF_DTC”. In this case you need to count the amount of needed data bytes and fill in this value directly.

Example 2 complex parameter:

```
on diagRequest FAULT_MEMORY::FAULT_MEMORY::ReadAll
{
  diagResponse this resp;
  // Set the number of DTCs returned
```

```
  DiagResize( resp, 9);
  // in this example 9 bytes are needed to transfer the response
  DiagSetComplexParameter ( resp, "List_of_DTC", 0, "DTC", 0xFAFAF );
  …
}
```

3.4.14 Negative Response handling

A request sometimes results in a negative response e.g. if the request can not be performed by the ECU. This section describes how to implement a handling for this kind of situation.

The service of the request can be specified exactly, but the service of the response is not clear. To handle the ambiguity of negative responses, it is suggested to implement an “all-handler”.

Example:

```
on diagResponse *
{
  // Handle the ambiguity of neg responses by treating them as '*'
  if( DiagIsNegativeResponse ( this ) )
  {
    write( "Received negative response for service 0x%x, code 0x%x",
    (long) DiagGetParameter( this, "SIDRQ_NR" ),
    (long) DiagGetParameter( this, "NRC" ) );
  }
}
```

A special case of negative response is a response with a code that indicates “I’m busy, I’ll respond later”. This means that a (hopefully) positive response that should be mapped against the actual request will follow later. To simulate this special kind of negative response from a simulated ECU, it is suggested to implement a timer in the ECU. The simulated ECU first sends a negative response, then starts a timer where the positive response is sent.

Example:

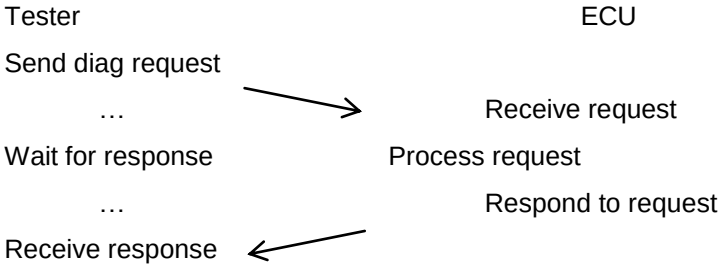
```
on diagRequest FaultMemory::FaultMemory::ReportByStatusMask
{
  // Send neg response with code 0x78 (requestCorrectlyReceived-ResponsePending)
  DiagSendNegativeResponse(this, 0x78);
  // Optionally set a timer to respond with a positive response later
  setTimer(posReq, 1);
  // pos resp after 1s
}
```

3.5 Combine Test Feature Set and Diagnostic Feature Set

Note: This chapter only applies to CANoe.

This document does not cover the Test Feature Set in detail. For details on Test Feature Set please refer to the online help in CANoe.

A very common diagnostic sequence is to send a request and to wait for a response before continuing with the next request.



To be able to use the Test Feature Set in combination with the Diagnostic Feature Set, you can create a *CAPL Test Module* instead of an ordinary ECU, and use it as a Diagnostic Tester. In this way you can use the built-in test functions like TestWaitForDiagnosticResponse() to get a smoother handling of the request/response scheme.

3.5.1 Difference between a Prog Node (ECU) and Test Module

You should put the code for setting the target in the TestControl/MainTest() section instead of the Start section (there is no Start section in a Test Module).

Example:

```
void MainTest()
{
  DiagSetTarget( "ECU_name");
}
```

3.5.2 Timeout handling

The timeout on a diagnostic request can be recognized automatically via the return value of TestWaitForDiagResponse() function.

Example:

```
TestWaitForDiagResponse( req, 5000 );  // wait 5s for a response on this request
```

3.5.3 A simple Diagnostic Tester

Create a test case in the TestControl section, e.g. tc_StartSession(), and call this test case from MainTest(). In tc_StartSession() create a diagnostic request and wait for response until timeout.

Example Diagnostic Tester:

```
void MainTest ()
{
  DiagSetTarget ( "Brake_Module" );
  tc_StartSession ();
}
testcase tc_StartSession ()
{
  // Create a request with correct qualifier
  diagRequest START_SESSION::DEFAULT_SESSION::StartSession req;
```

```
  // Send the request
  DiagSendRequest ( req );
  // Wait 5s for response and evaluate
  // Return values of Test Feature Set functions are defined in the help file
  if( 1 != TestWaitForDiagResponse ( req, 5000) )
  {
    // Response not received
    TestStepFail( "Start Default Session", "No response received!");
  } else
  {
    // evaluation of response data
  }
}
```

To produce a response to your Diagnostic Tester you should define another node which answers like this:

Example simulated ECU:

```
on diagRequest START_SESSION::DEFAULT_SESSION::StartSession
{
  // Create a response to this request
  DiagResponse this resp;
  // Send the response
  DiagSendResponse( resp);
}
```

3.5.4 Diagnostics tests with XML patterns

For common test sequences, it is not necessary to write a CAPL tester module. Instead an XML test specification can be written and executed using CANoe's test feature set. A test report can be created automatically (in XML format that is converted to HTML). Please refer to the help documentation, section "Test > Test Feature Set (TFS) > XML Test Module > Structure of XML files" for details about XML tests.

Example: The following test case will send a request to the ECU specified and check if the response parameter confirms to the predicate.

```
<testcase ident="Sample" title="Sample Diagnostics tests - EQ/NE">
  <description>
    This is a sample diagnostics XML test pattern.
  </description>
  <diagservicecheck timeout="2s" ecu="KWPSim"
    class="IDENTIFICATION" instance="ECU_Identification" service="Read"
    result="positive" title="Read Identification">
    <diagvalue qualifier="Diagnostic_Identification">
      <eq> 0x01 </eq>
    </diagvalue>
  </diagservicecheck>

  <diagservicecheck timeout="2s" ecu="KWPSim"
    class="IDENTIFICATION" instance="ECU_Identification" service="Read"
    result="positive" title="Read Identification">
    <diagvalue qualifier="Diagnostic_Identification">
      <ne> 0x01 </ne>
    </diagvalue>
  </diagservicecheck>
</testcase>
```

Since these checks contradict each other, one of them has to fail. This is reported in the resulting output:

1.1 Test Case Sample: Sample Diagnostics tests – EQ/NE: Failed

This is a sample diagnostics XML test pattern.

Test case begin: 2005-08-04 15:53.13 (logging timestamp: 0.000000)

Test case end: 2005-08-04 15:53.15 (logging timestamp: 0.195000)

1. Read Identification: Passed

Timestamp	Test Step	Description	Result
0.000000		Test pattern begin	-
0.001500	Resume reason	Resumed on AuxEvent 'DiagServiceCheck()' Reason: RequestConfirmation Elapsed time=2ms (max=2000ms)	-
0.022000	Resume reason	Resumed on AuxEvent 'DiagServiceCheck()' Reason: ResponseIndication Elapsed time=20ms (max=2000ms)	-
0.022000	2	Positive response arrived.	pass
0.022000	3	Parameter "Diagnostic_Identification": "0x01" == "0x01"	pass
0.022000		Test pattern end	-

2. Read Identification: Failed

Timestamp	Test Step	Description	Result
0.022000		Test pattern begin	-
0.174500	Resume reason	Resumed on AuxEvent 'DiagServiceCheck()' Reason: RequestConfirmation Elapsed time=152ms (max=2000ms)	-
0.195000	Resume reason	Resumed on AuxEvent 'DiagServiceCheck()' Reason: ResponseIndication Elapsed time=20ms (max=2000ms)	-
0.195000	2	Positive response arrived.	pass
0.195000	3	Parameter "Diagnostic_Identification" does not meet condition "0x01" != "0x01"	fail
0.195000		Test pattern end	-

4.0 Advanced examples

4.1 ECU simulation of “Response Pending”

If an ECU is not able to respond to a request immediately, it can send a negative response with the response code “requestCorrectlyReceived-ResponsePending” (RCR-RP, 0x78) to indicate that it will delay the final response. To simulate this behavior in CAPL, the following code pattern can be used:

Example:

```
variables
{
  // ...
  BYTE gDelayedResponse[ 500];         // global buffer for one delayed response
  dword gDelayedResponseLen = 0;       // length of the response stored, or 0 if none
  msTimer gDelayTimer;   // timer for delayed response
}´

on diagRequest Class1::Instance1::Action1
{
  diagResponse this resp;
  // Set the parameters in the response.
  DiagSetParameter( resp, "Param1", 1);
  DiagSetParameter( resp, "Param2", 2);
```

```
  // Copy the response data into the global buffer for sending later.
  gDelayedResponseLen =
    DiagGetPrimitiveData( resp, gDelayedResponse, elcount( gDelayedResponse));
  // Send "response pending"
  DiagSendNegativeResponse( resp, 0x78);
  // Start the timer that will initiate the actual sending of the response.
  settimer( gDelayTimer, 100);
}



on timer gDelayTimer
{
  diagResponse ClassX::InstanceX::ActionX dummy;
  if( gDelayedResponseLen > 0)
  {
    DiagResize( dummy, gDelayedResponseLen);
    DiagSetPrimitiveData( dummy, gDelayedResponse, gDelayedResponseLen);
    // The diagnostics object may now have changed its "type"!
    DiagSendResponse( dummy);
    gDelayedResponseLen = 0;
  }
}
```

Note that the "type" of the response object in the "on timer gDelayTimer" handler can differ from the one it is initialized with, depending on the content of the data that is written into the object. Of course, a more elaborate ECU simulation might send the "Response Pending" negative response, start the timer, and fill the response object in the timer handler.

4.2 Modifying the length of a diagnostic object

The length of a diagnostic object e.g. a diagnostic response can be resized. This is typically useful for sending diagnostic responses containing DTCs from a simulated ECU because this kind of diagnostic response can be of very different length depending on the number of DTCs.

If there is a parameter that specifies the number of simple parameter sequences that follow (e.g. "NumberOfDTC"), set that parameter to the value you need:

Example:

```
DiagResponse this resp;
DiagSetParameter( resp, "NumberOfDTC", 12); // 12 sequences follow
DiagResize( resp); // Not necessary in CANoe 5.2
```

The example above will make room for 12 parameter sequences.

If no such parameter exists, you have to specify the number of bytes to reserve.

Example:

```
DiagResize( resp, 20); // resize the response to 20 bytes
```

4.3 Fill diagnostic content

A diagnostic parameter can be set directly via raw bytes instead of using symbolic values. This can also be useful for simulating errors in e.g. diagnostic responses.

Example:

```
  char ECUpartNo[25] = "030821111A";
  byte inbuffer[25];
  // Convert from char array to byte array
  for(i=0;i<25;i++) inBuffer[i] = ECUpartNo[i];
  // Set the parameter's raw byte representation in the response.
  DiagSetParameterRaw(resp, "Partnumber_for_ECU", inBuffer, 25);
```

4.4 Fault injection

To verify how the ECU reacts on e.g. an incorrect diagnostic request the parameters of the request can be manipulated by using the function `DiagSetPrimitiveData()`. This function can also be used to patch the content directly to simulate an error in the ECU implementation.

You can declare an object with

```
diagRequest STORED_DATA::Sinus::Read req;
```

and it will hold a buffer with the whole "bus message" (i.e. what will be transported over the bus as data, including service and subfuction IDs). If you put different data into the object with DiagSetPrimitiveData, it might no longer be a STORED_DATA::..., but any other "type" of response, or even something unspecified.

4.4.1 Make request length illegal

```
// Set length of request object to an incorrect value to check ECU action
// Length differs from Diagnostic Description
DiagResize( req, 2);
```

If this request is sent, the transport layer will operate correctly, i.e. the data will be transferred correctly.

4.4.2 Introduce errors on transport protocol level

It is possible to test the ECU's capacity to cope with errors on the transport layer.

Example:
The tester claims to send the full data (e.g. 10 bytes), but stops sending after the first frame, i.e. the Consecutive Frames are not sent:

```
_Diag_DataRequest( BYTE data[], DWORD count, long furtherSegments) {
   if( gAbortAfterFF) {  // Using a flag to trigger fault injection
      CanTpFI_Enable( gHandle);       // Activate fault injection functionality
        // gHandle contains the handle of the TP connection
      CanTpFI_SendXByte( gHandle, 1, 8, -1); // Send only 1 byte but fill First Frame to DLC 8
      gAbortAfterFF = 0; // Do this only once
   }
   CanTpSendData(gHandle, data, count);
}
```

For more details on the fault injection feature of the OSEK_TP.DLL, please cf. [1], chapter "Fault Injection".

4.5 Access a node via a gateway simulation

- In order to access a node using the diagnostics features of CANoe, it is possible to introduce a simple TP-level gateway simulation in the setup. The diagnostics description file can be configured to use standard ISO TP data transfer on CAN. Assign the Diagnostic Description to the CAN bus the gateway is attached to, *not* to the gateway node itself.
- In the simulation setup, configure the gateway node to use the ISO TP DLL for CAN (OSEK_TP.DLL) and the TP DLL for the corresponding network (e.g. LINtp.DLL for LIN), i.e. load these DLLs under "modules" or configure a database.
- The gateway simulation has to receive requests sent on the CAN bus (by the Diagnostics Console or fault memory window, real nodes, simulated nodes, test modules, etc.), and send the data on the LIN bus. The same approach has to be used for responses from the LIN to CAN.
- Note that the "bus context" is set to the bus where the data has been received on, and that the context has to be switched to the other bus before forwarding the data.
- In case of LIN, the gateway simulation has to act as the LIN master node.

The following implementation of a TP-level gateway simulation can be used as an example (the settings in "on start" have to be adapted in most cases).

```
variables
{
  char gECU[10] = "Gateway";
  long gNAD;              // node address of target node in LIN network
  long gCanTpHandle;      // handle of the CanTp connection
  dword gLinBusContext;
  dword gCanBusContext;
}

on start
{
  // !!!Adapt the parameters in this function!!!

  gCanTpHandle = CanTpCreateConnection(0);      // 0 = Normal mode
  CanTpSetTxIdentifier(gCanTpHandle, 0x400);
  CanTpSetRxIdentifier(gCanTpHandle, 0x200);

  gNAD = 1;

  gCanBusContext = GetBusNameContext("CAN");
  gLinBusContext = GetBusNameContext("LIN");

  setWriteDbgLevel(0);
}

CanTp_ReceptionInd (long handle, byte data[])
{
  // This function returns the data received
  writeDbgLevel(1,"%s: CanTp_RecepionInd", gECU);

  setBusContext(gLinBusContext);
  LINtp_DataReq(data, elcount( data), gNAD);
}

LINtp_DataInd(long count)
{
  /* This function returns the number of data received */
  byte rxBuffer[4096];
  writeDbgLevel(1,"%s: LINtp_DataInd", gECU);

  LINtp_GetRxData(rxBuffer, count);

  setBusContext(gCanBusContext);
  CanTPSendData( gCanTpHandle, rxBuffer, count);
}

LINtp_ErrorInd(int error)
{
}

CanTp_ErrorInd( long connHandle, long error)
{
}
```

5.0 Common mistakes

Problem: Why are the Diagnostics request or Diagnostics response event categories not available in the symbol explorer of the CAPL browser?

Solution: You must add a Diagnostic Description to your CANoe configuration, see paragraph 3.1.1 on how to do that.

Problem: Why is the diagnostic request that I send (from CAPL) not visible in trace?

Solution: You must assign the Diagnostic Description to the actual ECU (or bus) that the request is directed to. If you can use the console, the Diagnostic Description is already correctly assigned.

Problem: Why is the following system message displayed in the Write window:

„System OSEK_TP ECU: Could not find mandatory callback function OSEKTL_ErrorInd!“

Solution: You have forgotten to include the indicated mandatory callback function into your simulated ECU.

Problem: Why is the following system message displayed in the Write window:

„System DiagCreate Request: Accessing CANdelaLib lead to an error, e.g. exception, not found.“

Solution: A request could not be created because it is missing in the Diagnostic Description or you selected a different variant in the “Diagnostics/ISO TP…” configuration dialog. The request could be incorrectly defined in CAPL – check the request qualifier with the qualifier you can copy via drag & drop from the symbol explorer.

Problem: Why is the following system message displayed in the Write window:

“System Request services with complex/uncertain parameters are not supported!”

Solution: When initialising the Diagnostics Console, all defined requests are inspected. But since the Diagnostics Console is currently not able to create requests if they contain complex parameters, those requests are filtered out of the display, i.e. they cannot be sent directly from the console (but you can send them directly by entering the raw bytes in the edit line of the console). This does not have anything to do with the access in CAPL; except that using the Diagnostics Console to find the qualifier path of the service does not work here, since the service will not be listed in the Diagnostics Console tree. (Note that the symbolic selection dialog (see chapter 2.4.6) will display these requests and their parameters too.)

Problem: Why is my simulated ECU marked with “OSEK_TP” in the simulation set-up?

Solution: Nodes that use the transport layer functionality in CANoe (i.e. segmentation and other transport layer functions typically needed for diagnostics) must have this node layer module assigned to them. CANoe needs this information in order to use the DLL file that implements the transport protocol. You can either inform CANoe via the .dbc-file (please see help file how to do this) or via the configuration dialogue of the node itself in simulation set-up. Consult paragraph 0 for details.

Problem: Why does the Trace window display: “Unknown action::Unknown instance”?

Solution: Data bytes sent or received can not be found in the Diagnostic Description. Correct either your CAPL or your Diagnostic Description. This behaviour could also occur if the ECU implementation (software) does not comply with the specification, i.e. the ECU diagnostic response contains data bytes that are not described in the Diagnostic Description.

Problem: Why is the value of a diagnostics parameter always written as 0, even though no warning message (like “parameter not found”) is printed in the write window?

Solution: In the following typical statement

```
Write( "%d", DiagGetParameter( object, "Parameter"));
```

the double type return value of the access function (cf. CAPL reference) is treated as a long argument, which will lead to printing 0 in most cases. It is necessary to cast the value or use a float format:

```
Write( "%d", (long) DiagGetParameter( object, "Parameter"));
Write( "%g", DiagGetParameter( object, "Parameter"));
```

6.0 Abbreviations

API Application Programmer Interface

CAPL CAN Access Programming Language

CCI CAPL Callback Interface

CDD CANdela Diagnostic Description

DBC DataBase for CAN

DFS Diagnostic Feature Set - diagnostic support in CANoe

DIS Draft International Standard

ECU Electronic Control Unit

DLL Dynamic Link Library

DTC Diagnostic Trouble Code

ISO International Organization for Standardization

ODX Open Diagnostic Data Exchange

OEM Original Equipment Manufacturer

SP Service Pack

TFS Test Feature Set - test support in CANoe

TP Transport Protocol

7.0 References

The documents mentioned here are part of the documentation that is included with every CANoe/CANalyzer installation. They can be found from Start menu/Programs/CANoe/Help, or as files directly.

[1] ISO/DIS 15765-2 Transport Protocol documentation: Doc/CanTP_Manual.PDF

[2] Generic CDDs implementing standards: Exec32\StandardCDDs\GenericKWP.cdd
Exec32\StandardCDDs\GenericUDS.cdd

8.0 Additional Resources

VECTOR APPLICATION NOTE
AN-IND-1-002 Testing with CANoe

9.0 Contacts

Germany
and all countries not named below:

Vector Informatik GmbH
Ingersheimer Str. 24
70499 Stuttgart
GERMANY
Phone: +49 711-80670-0
Fax: +49 711-80670-111
E-mail: info@de.vector.com

France, Belgium, Luxemburg:

Vector France S.A.S.
168, Boulevard Camélinat
92240 Malakoff
FRANCE
Phone: +33 1 42 31 40 00
Fax: +33 1 42 31 40 09
E-mail: information@fr.vector.com

Sweden, Denmark, Norway, Finland, Iceland:

VecScan AB
Theres Svenssons Gata 9
41755 Göteborg
SWEDEN
Phone: +46 31 764 76 00
Fax: +46 31 764 76 19
E-mail: info@se.vector.com

United Kingdom, Ireland:

Vector GB Ltd.

Rhodium, Central Boulevard
Blythe Valley Park
Solihull, Birmingham
West Midlands B90 8AS
UNITED KINGDOM
Phone: +44 121 50681-50
Fax: +44 121 50681-69
E-mail: info@uk.vector.com

China:

Vector Automotive Technology (Shanghai) Co., Ltd.
Sunyoung Center
Room 1701, No.398 Jiangsu Road
Changning District
Shanghai 200050
P.R. CHINA
Phone: +86 21 6432 53530
Fax: +86 21 6432 5308
E-mail: info@cn.vector.com

India:

Vector Informatik India Pvt. Ltd.

4/1/1/1, Sutar Icon, Sus Road,
Pashan, Pune - 411 021
INDIA

Phone: +91 20 2587 2023
Fax: +91 20 2587 2025
E-mail: info@in.vector.com

USA, Canada, Mexico:

Vector CANtech, Inc.
39500 Orchard Hill Place, Suite 550
Novi, MI 48375
USA

Phone: +1 248 449 9290
Fax: +1 248 449 9704
E-mail: info@us.vector.com

Japan:

Vector Japan Co. Ltd.
Tennozu Yusen Bldg. 16F
2-2-20 Higashi-shinagawa,
Shinagawa-ku,
Tokyo 140-0002
JAPAN
Phone: +81 3 5769 7800
Fax: +81 3 5769 6975
E-mail: info@jp.vector.com

Korea:

Vector Korea IT Inc.
5F, Gomoas bldg., 12
Hannam-daero 11-gil, Yongsan-gu
Seoul, 140-889
REPUBLIC OF KOREA

Phone: +82 2 807 0600
Fax: +82 2 807 0601
E-mail: info@kr.vector.com