Author(s)	Samer White, Jun Lin, Niraj Gulave, Michael Kienast
Restrictions	Public Document
Abstract	This application note is a general introduction to the COM server functionality of CANalyzer and CANoe. It presents the basic technical aspects and possibilities, and explains these using Microsoft Visual Basic and C# examples. This application note is a general introduction to the COM server functionality of CANalyzer and CANoe. It presents the basic technical aspects and possibilities, and explains these using Microsoft Visual Basic and C# examples.

Table of Contents

1.0 Overview

1.1 Introduction

CANalyzer and CANoe are Vector's analysis and simulation tools for bus systems used in the automotive industry. Options for LIN, MOST and FlexRay are available. CANalyzer is the right tool to analyze, observe and simulate data traffic. CANoe supports the full feature set of CANalyzer and additionally allows creating a whole functional model of the network system. This model can be used in the entire development process of the network from planning to final testing.

The purpose of this application note is to give a general introduction to the use of CANalyzer/CANoe as a COM server. Using COM server users can write their own applications or components and interface/interoperate with CANalyzer/CANoe. The application note focuses on operations that are used during measurement.

The basic technical aspects and possibilities of the COM server functionality will be presented. Examples, created with Microsoft Visual Basic .NET (simply refer to as VB.NET in this document) and C# are used to get the application engineer started with programming for the CANalyzer/CANoe COM server.

Examples written in this document are using CANalyzer/CANoe 8.0 and Microsoft Visual Studio 2010.

1.2 COM – Component Object Model

COM is a standard defined by Microsoft for the communication between different software components. Different programming languages can be used to create such components. These components can be built by different software developers, independently from each other.

1.3 CANalyzer/CANoe as a COM Server

CANalyzer and CANoe have a built-in COM interface beginning with version 3.0. The following goals can be achieved by using this COM server functionality:

- Creation and modification of CANalyzer/CANoe configurations
- Measurement automation, i.e. configuration load, start and stop measurement, start test modules
- Exchange of data between CANalyzer/CANoe and other external programs or applications suites. The external programs must support the COM technology as well.
- Automation of test sequences with customer-specific control panels.
- Remote control of measurements using CANalyzer/CANoe. The remote control of CANalyzer/CANoe over a network is also possible.

Different programming and scripting languages can be used to access the COM server functionality of CANalyzer and CANoe. All used scripts must be based on Microsoft's Windows Script software component. Scripting languages such as VBScript or Jscript can be also used to create test reports in Microsoft Excel or Microsoft Word. Programming languages such as Microsoft Visual Basic, Microsoft Visual C++, and Borland Delphi can be used to create user specific applications.

1.4 COM Server Registration

The COM server is already registered when CANalyzer/CANoe is installed. If the installation directory has been moved, a new registration is necessary. To do this open the MS-DOS prompt and change to the installation directory (…\Exec32). Type the following command:

```
canw32 –regserver      //for CANalyzer
canoe32 –regserver     //for CANoe
```

Or execute \Exec32\RegisterComponents.exe.

To delete the registration of the COM Server type in the following:

```
canw32 –unregserver    //for CANalyzer
canoe32 –unregserver   //for CANoe
```

Or execute \Exec32\RegisterComponents.exe /u .

Application Note AN-AND-1-117

Independent from the registered CANalyzer/CANoe version the COM script will use any opened CANalyzer/CANoe.

2.0 Using .NET to Access the COM Server

2.1 Project Configuration

This application note uses examples written for VB.NET and C# to demonstrate in detail how to use the CANalyzer/CANoe COM server.

The CANalyzer/CANoe COM server consists of several objects that are arranged in an object hierarchy. Each COM server object consists of up to three element types:

Object Properties   example: `CANController.Baudrate`
Object Methods   example: `Measurement.Start`
Object Events   example: `Application.OnQuit`

Basic knowledge of the COM server object hierarchy will help the application engineer to obtain a good overview of the COM server functionality. A description of the COM server object hierarchy can be found in CANalyzer's and CANoe's online help system.
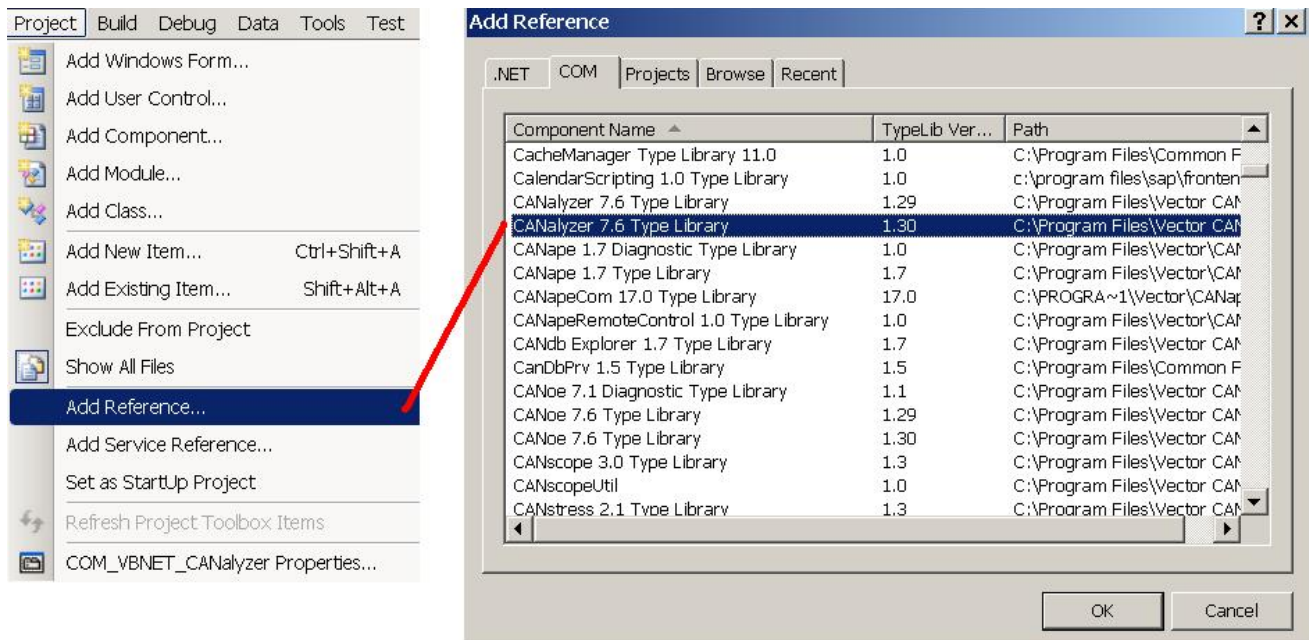

Figure 1: Referencing the CANalyzer Type Library

In order to have access to the COM server from VB.NET and C#, the VB.NET and C# project has to be configured to use the CANalyzer/CANoe type library. In VB.NET and C# this is accomplished by using the project's reference, see Figure 1.

The following sections use the *"COM.Net.cfg"* CANalyzer demo configuration that is available when you install CANalyzer/CANoe. The code snippets for VB.NET and C# which can be found below, demonstrate how to establish and interact with Vector CANalyzer. As these are only short parts of a whole application code please take a look at the delivered COM demo applications which are shipped with complete source code. If you are using Vector CANoe you will also find COM demo applications with their source code in the demo directory.

The snippets are annotated with a hint which refers to the part (or function) of the demo applications' source code in which the respective statements are used. This allows you to quickly look up in which context you could use the statements.

2.2 Controlling a Measurement

Once a VB.NET or C# project has been configured to use the CANalyzer/CANoe type library, the application has access to the COM server. A global variable here named `mCANalyzerApp` is needed in VB.NET and C# through which the COM server's *Application* object will be accessed. If you want to use CANoe instead of CANalyzer then you need to replace the (sub-)string "CANalyzer" with "CANoe" for all snippets below.

`mCANalyzerApp` should be declared as:

```
In VB.NET:
Private WithEvents mCANalyzerApp As CANalyzer.Application
In C#:
private CANalyzer.Application mCANalyzerApp;
```
(Demo Source Code → Global declaration)

The next step is to initialize mCANalyzerApp with the following statement. If CANalyzer/CANoe is not launched yet, the statement will also result in starting a new instance of CANalyzer/CANoe.

```
In VB.NET:
mCANalyzerApp = New CANalyzer.Application
In C#:
mCANalyzerApp = new CANalyzer.Application();
```
(Demo Source Code → EventHandler mButtonOpenConfiguration_Click)

The *Measurement* object which is necessary for the control of a measurement has to be declared like this:

```
In VB.NET:
Private WithEvents mCANalyzerMeasurement As CANalyzer.Measurement
In C#:
private CANalyzer.Measurement mCANalyzerMeasurement;
```
(Demo Source Code → Global declaration)

The initialization of `mCANalyzerMeasurement` happens with the following statement.

```
In VB.NET:
mCANalyzerMeasurement = mCANalyzerApp.Measurement
In C#:
mCANalyzerMeasurement = (CANalyzer.Measurement)mCANalyzerApp.Measurement;
```
(Demo Source Code → EventHandler mButtonOpenConfiguration_Click)

At next the COM server is used to load the application engineer's configuration with the *Open* method. Please notice that you have to change the path according to your own environment.

```
In VB.NET:
mCANalyzerApp.Open("D:\COM_CANalyzer\COM.Net.cfg", True, True)
In C#:
mCANalyzerApp.Open(@"D:\COM_CANalyzer\COM.Net.cfg", true, true);
```
(Demo Source Code → EventHandler mButtonOpenConfiguration_Click)

Note: In order to run the COM demo applications you must use the "COM.Net.cfg" configuration as it was designed to work with these demo applications. Do not move this configuration out of its directory, otherwise the demo applications will not work properly!

To make sure that a configuration has been opened successfully, you can use the *OpenConfigurationResult* object. It returns 0 if the opening of the configuration was successful.

Application Note AN-AND-1-117

```
In VB.NET:
Dim ocresult As CANalyzer.OpenConfigurationResult =
 mCANalyzerApp.Configuration.OpenConfigurationResult
If (ocresult.result = 0) Then {…} End If
```
```
In C#:
CANalyzer.OpenConfigurationResult ocresult =
 mCANalyzerApp.Configuration.OpenConfigurationResult;
if (ocresult.result == 0) {…}
```

(Demo Source Code → EventHandler mButtonOpenConfiguration_Click)

In case the configuration contains simulation nodes or test nodes (CAPL, .NET or XML), it is a good practice to re-compile all code before a new measurement is started. This ensures that the most recent binaries will be used in the measurement. Compilation of all nodes is done with the *Compile* method of the *CAPL* object. The *CAPL* object is accessed with the *Application* object (refer to the COM server object hierarchy). Any compilation error will be contained in the CompileResult property of the *CAPL* object.

```
In VB.NET:
Dim CANalyzerCAPL As CANalyzer.CAPL = mCANalyzerApp.CAPL
CANalyzerCAPL.Compile(Nothing)
```
```
In C#:
CANalyzer.CAPL CANalyzerCAPL = (CANalyzer.CAPL)mCANalyzerApp.CAPL;
CANalyzerCAPL.Compile(null);
```

(Demo Source Code → EventHandler MeasurementInitiated)

At this point everything is setup and ready for a new measurement. A new measurement can be started (and stopped) using the *Measurement* object.

```
In VB.NET:
mCANalyzerMeasurement.Start()
```
```
In C#:
mCANalyzerMeasurement.Start();
```

(Demo Source Code → EventHandler mButtonStartStop_Click)

```
In VB.NET:
mCANalyzerMeasurement.Stop()
```
```
In C#:
mCANalyzerMeasurement.Stop();
```

(Demo Source Code → EventHandler mButtonStartStop_Click)

The COM server also contains functionality to quit the CANalyzer/CANoe application. This is accomplished by using the *Quit* method of the *Application* object. The following code checks the *Running* property of the *Measurement* object to find out if a measurement is running and if so, stops the measurement before quitting the CANalyzer/CANoe application:

```
In VB.NET:
If (mCANalyzerMeasurement.Running) Then
    mCANalyzerMeasurement.Stop()
End If
mCANalyzerApp.Quit()
```
```
In C#:
if (mCANalyzerMeasurement.Running)
{
    mCANalyzerMeasurement.Stop();
}
mCANalyzerApp.Quit();
```

(Demo Source Code → Not used in demo)

5

Application Note AN-AND-1-117

2.3 Accessing Communication Data

The communication data on e.g. CAN buses is stored in CANdb databases. A CANdb database file has a .dbc file extension and is used as a common database for the entire Vector tool chain. Each CAN message contains up to 8 data bytes which represent one or more signals.
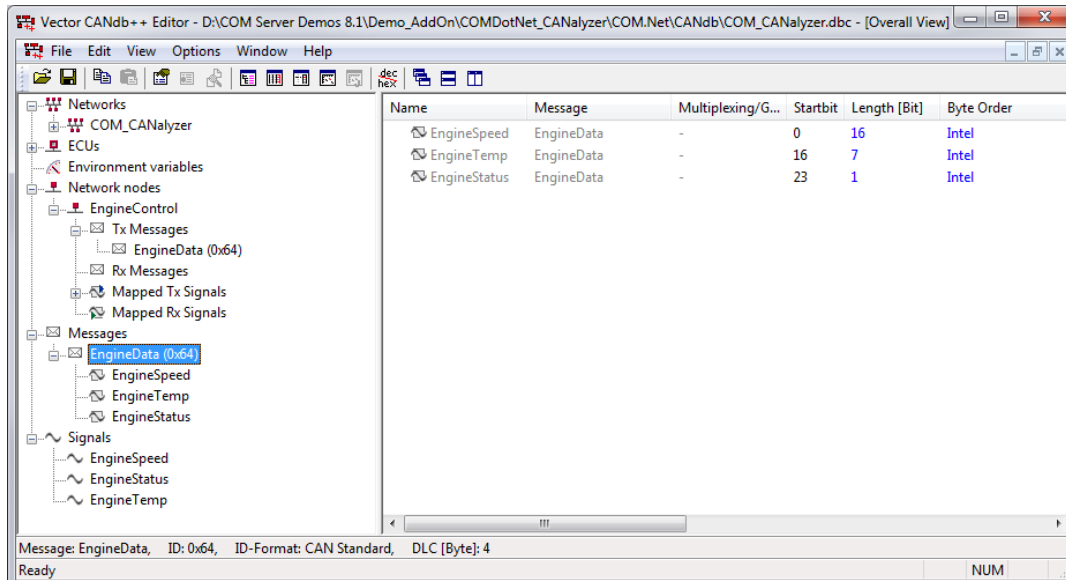

Figure 2: CAN messages and signals in CANdb++

Figure 2 displays a database screenshot when opened with the CANdb++ database editor. The CAN message 'EngineData' is selected in the left window and the right window shows its signals. The message has the identifier 64 (hex) and contains 3 data bytes. Signal 'EngineSpeed' is defined as a signal with a length of 16 bits (2 bytes) and it is located in byte 0 and byte 1 of the CAN message 'EngineData'.

Several other database types can be assigned to the buses.

The main advantage of using databases is that the communication data can be accessed using its descriptive name, i.e. if the 'EngineSpeed' signal should be accessed, the name 'EngineSpeed' can be used to read or write the signal value. The application engineer doesn't have to worry about the location of this signal.

The COM server provides functionality to access communication data through database signals. To access a *Signal* object in VB.NET and C#, it must first be assigned to a variable. The variable is declared as:

```
In VB.NET:
Private mCANalyzerEngineSpeed As CANalyzer.Signal
```
```
In C#:
private CANalyzer.Signal mCANalyzerEngineSpeed;
```

(Demo Source Code → Global declaration)

The method *GetSignal* of object *Bus* can be used to assign the signal to the `mCANalyzerEngineSpeed` variable. Therefore, it is necessary to pass the channel number on which the signal is sent, the message name to which the signal belongs und the signal name itself as parameters. The *Bus* object itself is accessed with the `mCANalyzerApp` variable:

```
In VB.NET:
Dim CANalyzerBus As CANalyzer.Bus = mCANalyzerApp.Bus("CAN")
mCANalyzerEngineSpeed = CANalyzerBus.GetSignal(1, "EngineData", "EngineSpeed")
```
```
In C#:
CANalyzer.Bus CANalyzerBus = (CANalyzer.Bus)mCANalyzerApp.get_Bus("CAN");
mCANalyzerEngineSpeed = (CANalyzer.Signal)CANalyzerBus.GetSignal(1, "EngineData",
  "EngineSpeed");
```

(Demo Source Code → Function ConfigurationOpened)

6

Application Note AN-AND-1-117

After the signal has been assigned to `mCANalyzerEngineSpeed` its value can be accessed with the *Value property*:

In VB.NET:
```
mTextboxEngSpeedOut.Text = mCANalyzerEngineSpeed.Value.ToString
```
In C#:
```
mTextboxEngSpeedOut.Text = mCANalyzerEngineSpeed.Value.ToString();
```

(Demo Source Code → EventHandler PollSignalValues)

To modify the values of signals (CANoe only!):

In VB.NET:
```
Dim newEngineSpeed As Double
…
mCANoeEngineSpeed.Value = newEngineSpeed
```
In C#:
```
double newEngineSpeed;
…
mCANoeEngineSpeed.Value = newEngineSpeed;
```

(Demo Source Code → EventHandler mTextboxEngSpeedIn_KeyPress)

2.4 Accessing Environment Variables (CANoe only)

Environment variables describe the behavior of network nodes with regards to external inputs, outputs or they can control its behavior. They are only available for CANoe. Environment variables are managed similar to signals but of course do not appear on any bus. This paragraph explains how to access environment variables via COM.
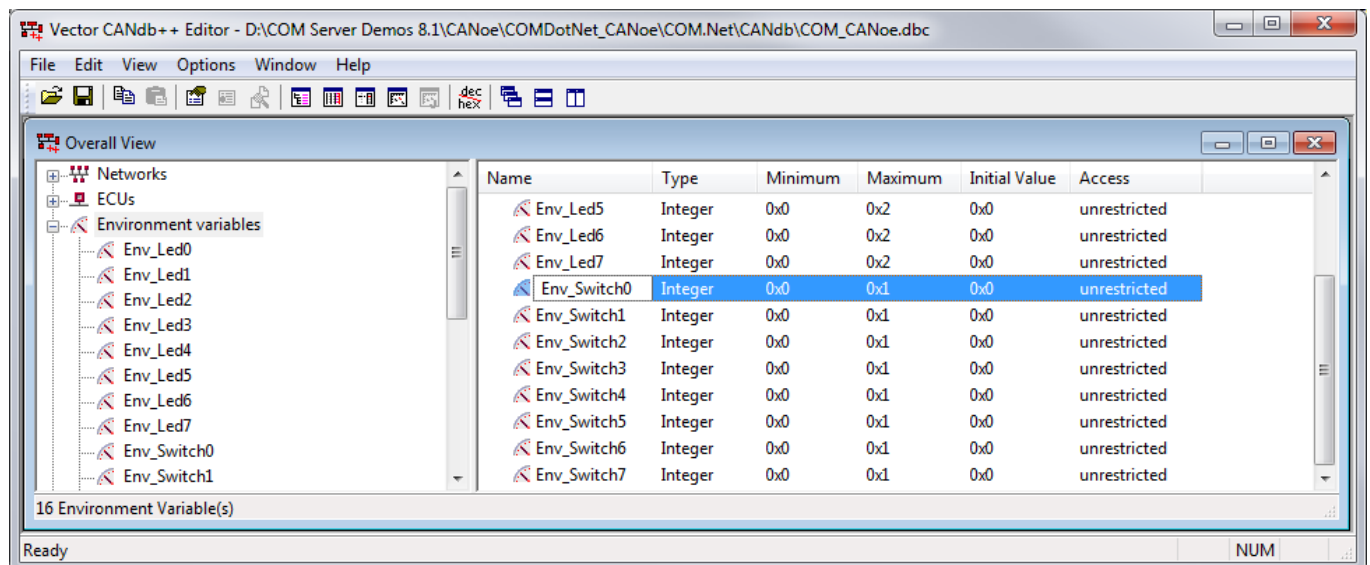

Figure 3: Environment variables in CANdb++

Environment variables are defined in CANdb databases. Figure 3 displays a screenshot of a database with environment variables.

Because environment variables represent the external inputs and outputs of a node, they can be used for interaction with a Graphical User Interface (GUI), i.e. a panel. Within CANoe it is possible to create your own panels with the panel editor / Panel Designer. Each control element on the panel (button, textbox, etc.) can be linked to an environment variable. During measurement the user is able to change the value of environment variables with the panel. Every time an environment variable is changed, an event is triggered and simulation or test nodes can react to these events.
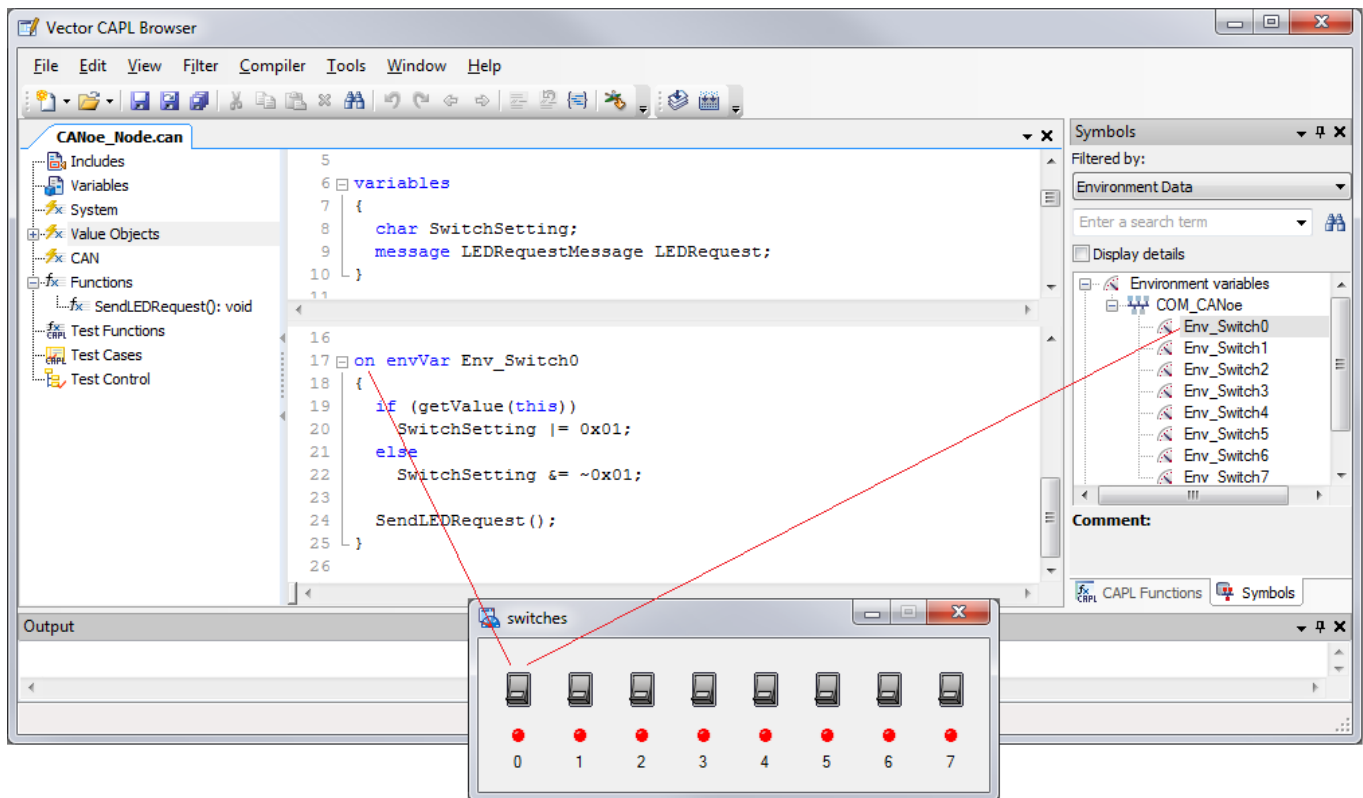
Application Note AN-AND-1-117

Figure 4: Using environment variables with a GUI

Figure 4 shows a panel. In this example switch number 0 on the control panel is linked to environment variable 'Env_Switch0'. If the user activates the switch by clicking on it, the value of environment variable 'Env_Switch0' changes from 0 to 1. This change triggers an event for which a handler is written.

Using the COM server it is possible to read and write the values of environment variables. This means that e.g. VB.NET or C# can be used to create an own GUI and to emulate external inputs and outputs of the node.

To access an *EnvironmentVariable* object in VB.NET and C#, it must first be assigned to a variable. This variable can be declared as:

```
In VB.NET:
Private WithEvents mSwitch0EnvVar As CANoe.EnvironmentVariable
In C#:
private CANoe.EnvironmentVariable mSwitch0EnvVar;
```
(Demo Source Code → Not used in demo)

The method *GetVariable* of object *Environment* can be used to assign the environment variable to the previously declared variable mSwitch0EnvVar. The environment variable must be defined in the CANdb database of the used CANoe configuration.

```
In VB.NET:
mSwitch0EnvVar = mCANoeApp.Environment.GetVariable("Env_Switch0")
In C#:
CANoe.Environment mEnvironment = (CANoe.Environment)mCANoeApp.Environment;
mSwitch0EnvVar = (CANoe.EnvironmentVariable)mEnvironment.GetVariable("Env_Switch0");
```
(Demo Source Code → Not used in demo)

After the environment variable object has been assigned to `mSwitch0EnvVar` it can be used to access or modify it with the *Value* property:

```
In VB.NET:
Dim switch0EnvVarValue As Double
switch0EnvVarValue = mSwitch0EnvVar.Value
```
```
In C#:
double switch0EnvVarValue;
switch0EnvVarValue = mSwitch0EnvVar.Value;
```

(Demo Source Code → Not used in demo)

```
In VB.NET:
mSwitch0EnvVar.Value = 1
```
```
In C#:
mSwitch0EnvVar.Value = 1;
```

(Demo Source Code → Not used in demo)

2.5 Accessing System Variables

System variables are used to exchange data between external applications and CANalyzer/CANoe nodes.

System variables are defined in namespaces, are globally unique and belong to the CANalyzer/CANoe configuration. Namespaces and system variables can either be defined in CAPL programs, via COM or with the CANalyzer/CANoe GUI.

If defined via COM, both namespaces and system variables exist until their definition is explicitly being cleared or until the configuration is being closed. When using VB.NET/C# to define variables on measurement start you should clear this definition on measurement stop, otherwise an attempt of redefine would fail on the next measurement start. Namespaces and their included variables that have been defined in CAPL are automatically being cleared on measurement stop.

The following VB.NET and C# example defines a namespace (NS) in the system, adds a new system variable (SysVar3) to it and sets a start value (val) for it:

```
In VB.NET:
Dim val As Integer
Dim CANalyzerSysVar3 As CANalyzer.Variable
Dim CANalyzerNamespaceNS As CANalyzer.Namespace
val = 1
CANalyzerNamespaceNS = mCANalyzerApp.System.Namespaces.Add("NS")
CANalyzerSysVar3 = CANalyzerNamespaceNS.Variables.Add("SysVar3", val)
```
```
In C#:
int val = 1;
CANalyzer.Variable CANalyzerSysVar3;
CANalyzer.System CANalyzerSystem = (CANalyzer.System)mCANalyzerApp.System;
CANalyzer.Namespaces CANalyzerNamespaces =
 (CANalyzer.Namespaces)CANalyzerSystem.Namespaces;
CANalyzer.Namespace mCANalyzerNamespaceNS =
 (CANalyzer.Namespace)CANalyzerNamespaces.Add("NS");
CANalyzer.Variables CANalyzerVariablesNS =
 (CANalyzer.Variables)CANalyzerNamespaceNS.Variables;
CANalyzerSysVar3 = (CANalyzer.Variable)CANalyzerVariablesNS.Add("SysVar3", val);
```

(Demo Source Code → Not used in demo)

You can change the value of a variable with the following command:

```
In VB.NET:
CANalyzerSysVar3.Value = 20
```
```
In C#:
CANalyzerSysVar3.Value = 20;
```

(Demo Source Code → Not used in demo)

The equivalent CAPL code (without error handling) is:

SysDefineNamespace("NS");
SysDefineVariableInt("NS", "SysVar3", 1);

And accordingly:

SysSetVariableInt("NS", "SysVar3", 20);

To have read and write access to a system variable that has been defined in another component, you write in VB.NET and C#:

```
In VB.NET:
Dim CANalyzerNamespaceGeneral As CANalyzer.Namespace =
 mCANalyzerApp.System.Namespaces("General")
mCANalyzerSysVar1 = CANalyzerNamespaceGeneral.Variables("SysVar1")
```
```
In C#:
CANalyzer.System CANalyzerSystem = (CANalyzer.System)mCANalyzerApp.System;
CANalyzer.Namespaces CANalyzerNamespaces =
 (CANalyzer.Namespaces)CANalyzerSystem.Namespaces;
CANalyzer.Namespace CANalyzerNamespaceGeneral =
 (CANalyzer.Namespace)CANalyzerNamespaces["General"];
CANalyzer.Variables CANalyzerVariablesGeneral =
 (CANalyzer.Variables)CANalyzerNamespaceGeneral.Variables;
mCANalyzerSysVar1 = (CANalyzer.Variable)CANalyzerVariablesGeneral["SysVar1"];
```

(Demo Source Code → Function ConfigurationOpened)

To get or set the value of a system variable use the *Value* property as described above.

In CAPL:

value = SysGetVariableInt("General", "SysVar1");

System variables are a simple and fast alternative to environment variables and it is not necessary to change DBC databases.

In case your CANalyzer/CANoe configuration contains .NET nodes please consider that these nodes rely on system variable type libraries. If you add new system variables the type libraries are recreated and thus also the .NET nodes are recompiled when measurement starts the next time. To prevent compilation, system variables should be configured once with the GUI. Then they become part of the configuration and the type libraries are only compiled when the underlying system variables definition changes.

2.6 Reacting to CANalyzer/CANoe Events

Using the COM server, it is possible to react to events that are triggered in CANalyzer/CANoe. An example of an event is the *OnInit* event of the *Measurement* object that is triggered every time a new measurement is initiated. Refer to the COM server object hierarchy in the CANalyzer/CANoe online help system to find out what events can be trapped using the COM server.

Important:	These events do not occur in the GUI thread so if you want to update indicators on the GUI, delegate functions have to be used! This is necessary because GUI controls are created in the GUI thread and for thread safety reasons it is not possible to access these controls from outside the GUI thread. You will find solutions for this problem below.

Application Note AN-AND-1-117

2.6.1 In VB.NET

In order to react to COM server object events in VB.NET, this feature must be enabled for the COM server object. In chapter 2.2, the global variable for the *Application* object was declared as:

```
Private WithEvents mCANalyzerApp As CANalyzer.Application
```
(Demo Source Code → Global declaration)

Recognize that to enable events for this specific object, the VB.NET keyword `WithEvents` must be used in the declaration of the variable.

To react to one of the COM server objects events, an event handler must be registered and implemented. For example, to react to the *OnInit* event of the *Measurement* object (using the declaration in this section), the statement would be:

```
AddHandler mCANalyzerMeasurement.OnInit, AddressOf MeasurementInitiated
```
(Demo Source Code → Function ConfigurationOpened)

This requires an event handler called *MeasurementInitiated* to be implemented. The structure of this event handler looks pretty simple:

Private Sub MeasurementInitiated()

     'Insert your code here

End Sub

This VB.NET function will be called every time a CANalyzer/CANoe measurement is initiated. The procedure for reacting to an event is the same for all COM server objects.

Do not forget to release the wired event handlers e.g. when quitting CANalyzer/CANoe:

```
RemoveHandler mCANalyzerMeasurement.OnInit, AddressOf MeasurementInitiated
```
(Demo Source Code → Function UnregisterCANalyzerEventHandlers)

When GUI elements should be manipulated due to the initialization of the measurement the following thread-safe code is used in the demo:

```
Private Delegate Sub DelSafeInvocation()
```
(Demo Source Code → Global Declaration)

```
Dim safeinvocation As DelSafeInvocation = New DelSafeInvocation(AddressOf
 MeasurementInitiatedInternal)
Invoke(safeinvocation)
```
(Demo Source Code → EventHandler MeasurementInitiated)

```
mGroupboxCAPLWriteWindowOut.Enabled = True
mGroupboxSystemVariables.Enabled = True
mGroupboxSignalValues.Enabled = True
mButtonStartStop.Text = My.Resources.StopMeasurement
mLabelMeasurementStatus.Text = My.Resources.StatusMeasurementStarted

mProgressbarSysVars.Value = mCANalyzerSysVar1.Value
mLabelSysVarsValue.Text = mCANalyzerSysVar1.Value.ToString
mLabelSysVarMin.Text = mCANalyzerSysVar1.MinValue.ToString
mLabelSysVarMax.Text = mCANalyzerSysVar1.MaxValue.ToString
```
(Demo Source Code → Function MeasurementInitiatedInternal)

Application Note AN-AND-1-117

2.6.2   In C#

Below example shows how to register events and their handlers in C#:

```
mCANalyzerMeasurement.OnInit += new
  CANalyzer._IMeasurementEvents_OnInitEventHandler(MeasurementInitiated);
```
(Demo Source Code → Function ConfigurationOpened)

It registers the CANalyzer measurement *OnInit* event and when this event is triggered (means when CANalyzer measurement is initialized) `MeasurementInitiated` handler is called.

The structure of this event handler is also simple:

private void MeasurementInitiated()

{

    // Add your code here

}

When the COM script is terminated the handler shall be unregistered:

```
mCANalyzerMeasurement.OnInit -= new
  CANalyzer._IMeasurementEvents_OnInitEventHandler(MeasurementInitiated);
```
(Demo Source Code → Function UnregisterCANalyzerEventHandlers)

Thread safety as already mentioned above can look like this in C#:

```
private delegate void DelSafeInvocation();
```
(Demo Source Code → Global Declaration)

```
DelSafeInvocation safeinvocation = new
DelSafeInvocation(MeasurementInitiatedInternal);
Invoke(safeinvocation);
```
(Demo Source Code → EventHandler MeasurementInitiated)

```
mGroupboxCAPLWriteWindowOut.Enabled = true;
mGroupboxSystemVariables.Enabled = true;
mGroupboxSignalValues.Enabled = true;
mButtonStartStop.Text = Properties.Resources.StopMeasurement;
mLabelMeasurementStatus.Text = Properties.Resources.StatusMeasurementStarted;

mProgressbarSysVars.Value = mCANalyzerSysVar1.Value;
mLabelSysVarsValue.Text = mCANalyzerSysVar1.Value.ToString();
mLabelSysVarMin.Text = mCANalyzerSysVar1.MinValue.ToString();
mLabelSysVarMax.Text = mCANalyzerSysVar1.MaxValue.ToString();
```
(Demo Source Code → Function MeasurementInitiatedInternal)

2.7   Calling CAPL Functions

CAPL programs can be used to simulate the behavior of simulation nodes, to analyze data traffic, and to create a gateway so that data can be exchanged between different buses.

CAPL can be used to implement application specific functions to perform a certain task. Using the COM server functionality it is possible to call these functions, which gives the application engineer total control over a measurement through the COM server. This section explains how the COM server can be used to call CAPL functions from VB.NET and C# applications. CAPL programs are written using the CAPL browser, which is part of CANalyzer/CANoe. Figure 5 shows a screenshot of the CAPL browser that illustrates where the CAPL functions are located.
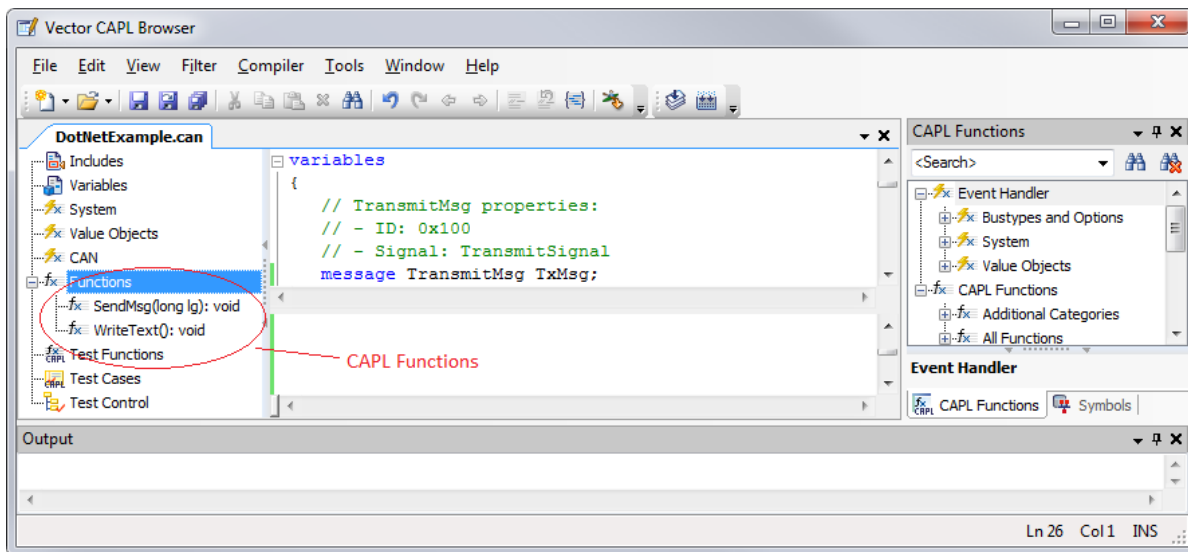
Figure 5: Location of functions in the CAPL browser

In order to call CAPL functions via COM a *CAPLFunction* object has to be declared and initialized with the *GetFunction* method of the *CAPL* object. The parameter for the *GetFunction* method is the exact name of the CAPL function.

Important: The assignment of a CAPL function to a variable can only be done in the *OnInit* event handler of the *Measurement* object. This means that the variable used to access the *Measurement* object must be declared using the `WithEvents` keyword in VB.NET as shown in 2.6.1.

To declare a *CAPLFunction* object you can use this code:

```
In VB.NET:
Private mCANalyzerMultiply As CANalyzer.CAPLFunction
In C#:
private CANalyzer.CAPLFunction mCANalyzerMultiply;
```
(Demo Source Code → Global declaration)

In this case a CAPL function called *Multiply* is assigned to the *CAPLFunction* object:

```
In VB.NET:
mCANalyzerMultiply = CANalyzerCAPL.GetFunction("Multiply")
In C#:
mCANalyzerMultiply                                                      =
(CANalyzer.CAPLFunction)CANalyzerCAPL.GetFunction("Multiply");
```
(Demo Source Code → EventHandler MeasurementInitiated)

After this, `mCANalyzerMultiply` can be used to call the CAPL function it refers to. To call the CAPL function, the *Call* method of the *CAPLFunction* object is used. In this example the *Multiply* function requires two operands as parameters and delivers the result to an integer variable which is used to display the result.

```
In VB.NET:
Dim result As Integer
result = mCANalyzerMultiply.Call(operand1, operand2)
mTextboxOperationResult.Text = result.ToString
In C#:
int result = (int)mCANalyzerMultiply.Call(operand1, operand2);
mTextboxOperationResult.Text = result.ToString();
```
(Demo Source Code → EventHandler mButtonCalculate_Click)

13

The number of input parameters is limited to 10 and it is not possible to pass on an array as a CAPL function parameter. It is practical to use input parameters of type *Long* in CAPL. In this case it is possible to pass on parameters from VB.NET/C# that are from types *Byte* (1 byte), *Integer* (2 bytes), and *Long* (4 bytes), without having to worry about the types matching up between VB.NET/C# and CAPL.

> **Important**: The return value of a CAPL function must always be of type integer. When using the COM server in combination with CANalyzer/CANoe it is only possible to use return values of CAPL functions if the P-Block representing your CAPL program is configured in CANalyzer's/CANoe's evaluation branch. Refer to Figure 7 for an explanation of the different branches in CANalyzer's/CANoe's measurement setup (*note:* the real-time branch and the evaluation branch are sometimes called simulation branch and analysis branch, respectively).
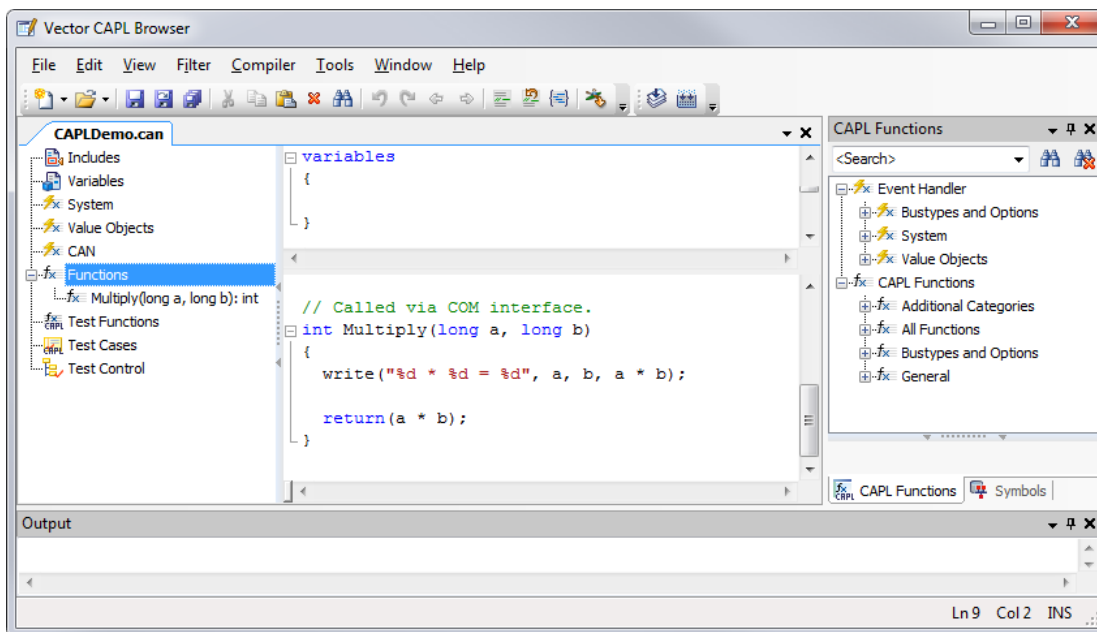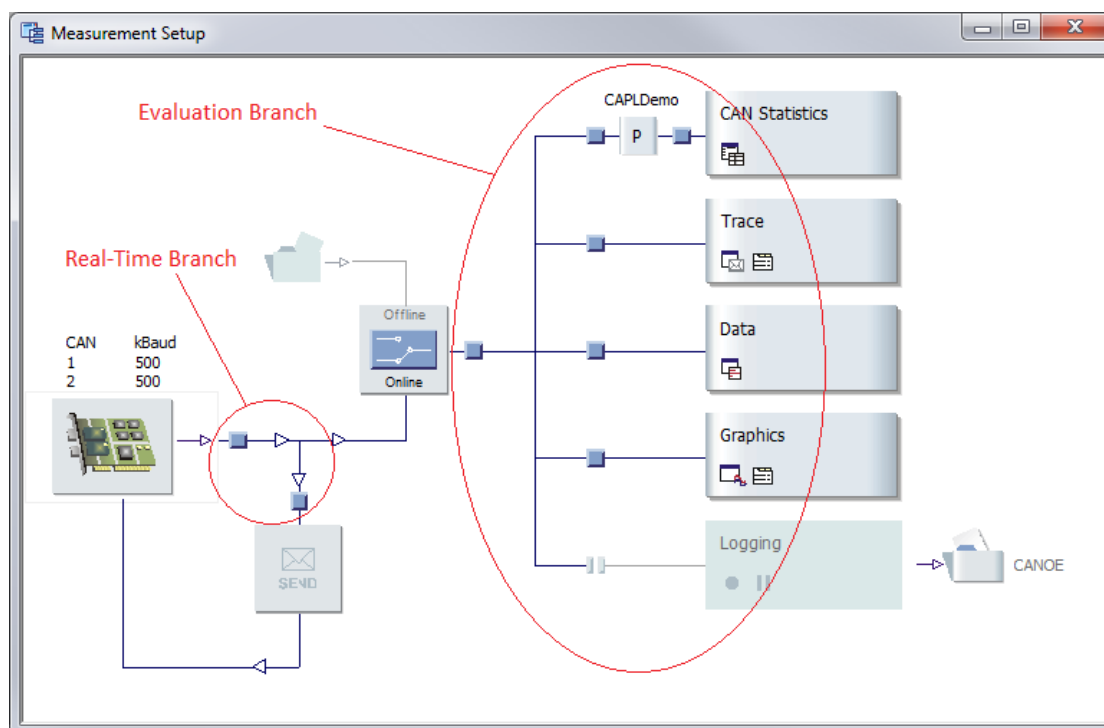

Figure 6: CAPL function 'Multiply()'

Figure 7: CANalyzer's measurement branches

2.8 Transmitting CAN Messages

The COM server doesn't provide functionality to initiate the transmission of CAN messages. However, it is still possible to initiate the transmission of CAN messages using a *CAPLFunction* object. The application engineer can therefore implement a function in his CAPL program that contains a call to the CAPL 'output()' function. The 'output()' function is a library function of CAPL that can be used in a CAPL program to transmit CAN messages or Error frames. Using the COM server's functionality to call CAPL functions, it is possible to call the application engineer's CAPL function (explained in chapter 2.7), which takes care of the transmission of CAN messages.

CANoe only:

If a signal value is modified from a VB.NET or C# program (explained in chapter 2.3) using the Interaction Layer of CANoe, the corresponding message will be sent automatically.

3.0 COM Server Example in VB.NET and C#

3.1 Screenshot

Figure 8 shows a screenshot of the example application using the COM server in VB.NET. It uses the "*COM.Net.cfg*" CANalyzer demo configuration showed in figure 9. This demo configuration is part of the CANalyzer installation and should not be moved out of its directory!
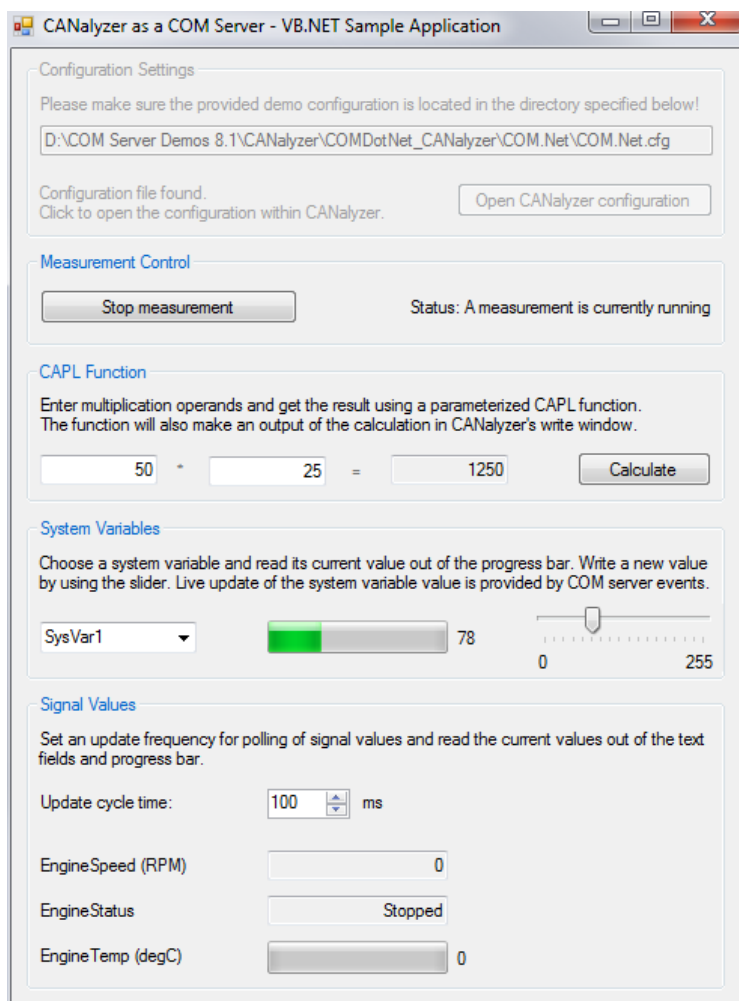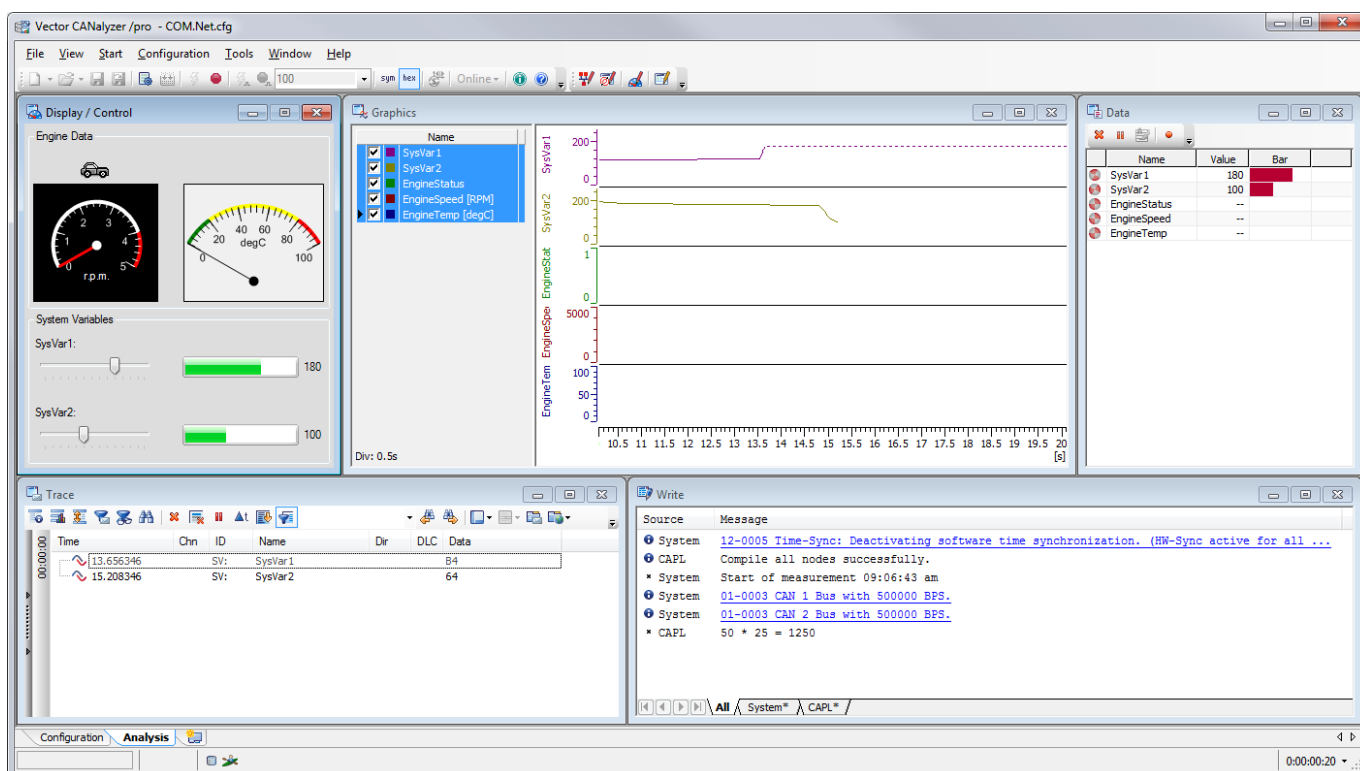

Figure 8: Screenshot of the example application

Application Note AN-AND-1-117

Figure 9: Screenshot of the example CANalyzer configuration

3.2 Quick Start Instructions

Run the demo applications .exe file out of the Release folder which can be found with the corresponding Visual Studio solution. Of course you can also open a VB.NET or C# solution and run the demo from within MS Visual Studio.

Once the application has opened, click on the button "Open CANalyzer configuration". This opens the "*COM.Net.cfg*" configuration (and CANalyzer itself if not opened before). Start a measurement by pressing the "Start measurement" button. Now you are able to multiply using a CAPL function, manipulate system variables and observe changes of signal values. If you change values within the demo application you can see the values changing in CANalyzer and vice versa.

CANoe only:

You can also change the values of signals (not only system variables) from within the demo application. This is not possible for CANalyzer due to its restrictions!

4.0 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

Application Note AN-AND-1-117