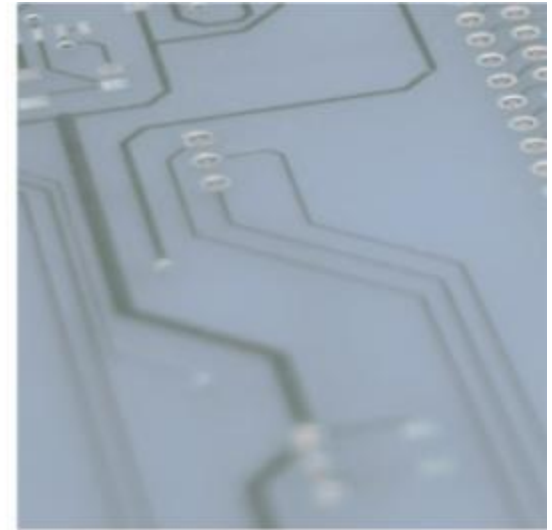
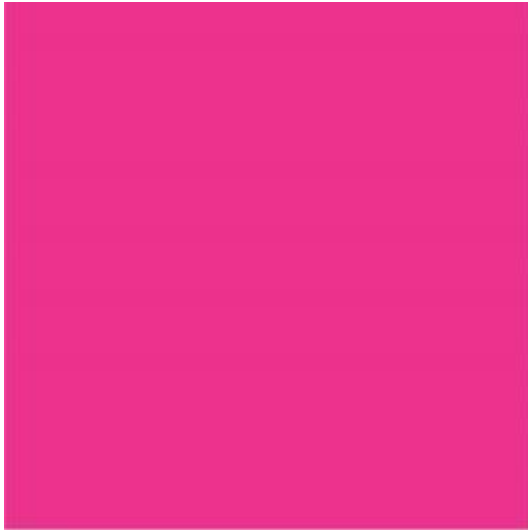


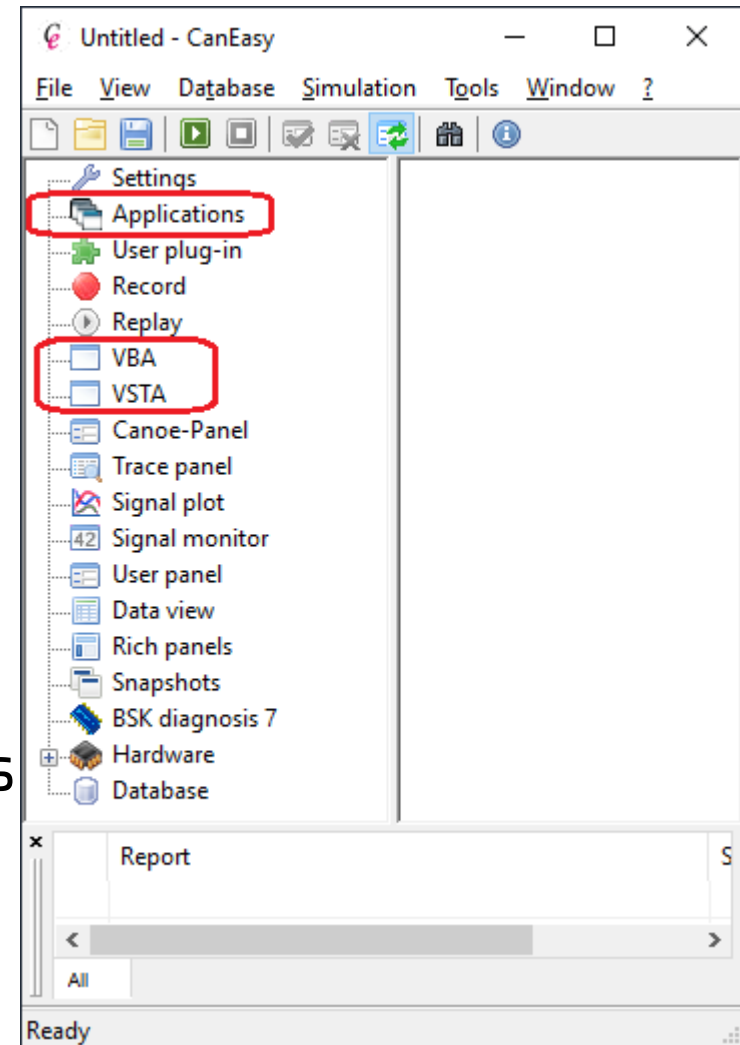
CanEasy Automation – COM



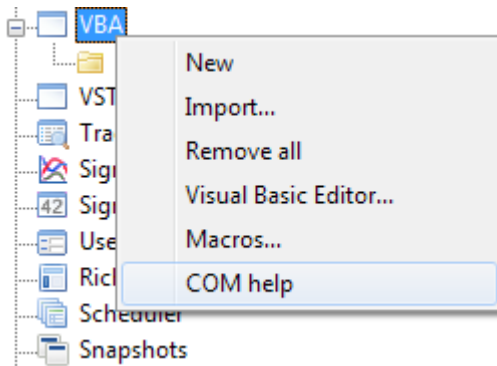
Holger Dahinten

- If the configuration options and arithmetic functions of CanEasy are not sufficient to simulate the behavior of a control unit, programming is necessary
- Programming can be done via integrated
 - **VBA** (Visual Basic for Applications)
 - **VSTA** (Visual Studio for Applications)
 - **MultiStudio** (Visual Studio Code)

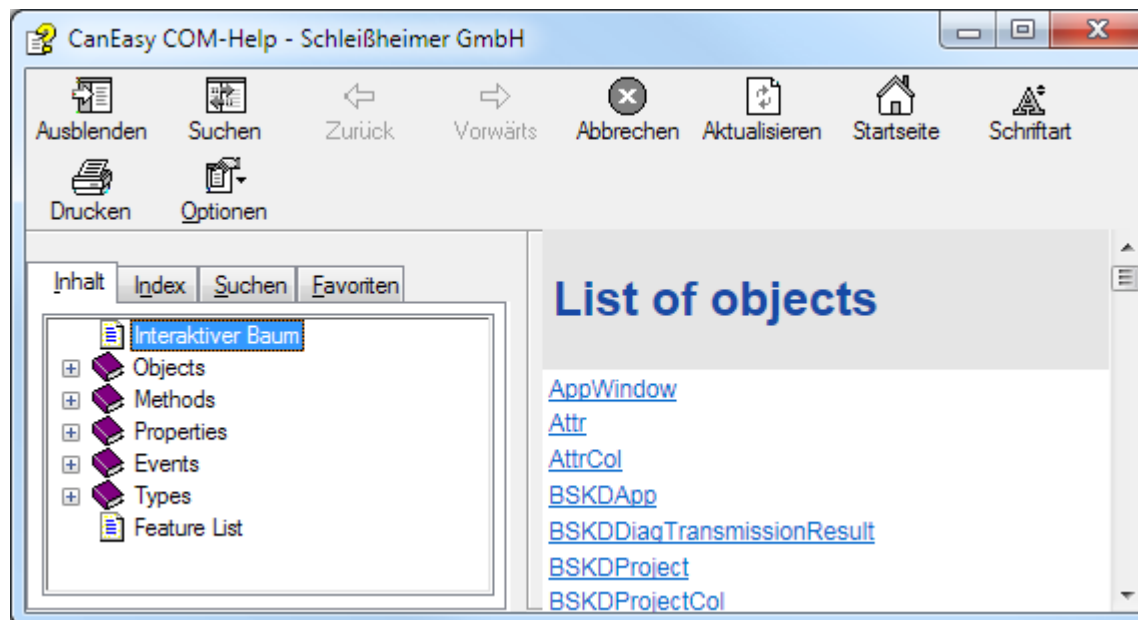
- CanEasy provides all its functions via a COM interface
- The COM interface is used by VBA, VSTA and MultiStudio-Applications
- Creating or modifying VSTA code requires installed Visual-Studio 2012 Professional [or newer]
- VBA has integrated IDE but has less development features (e.g. no multi threading, and only simple forms)



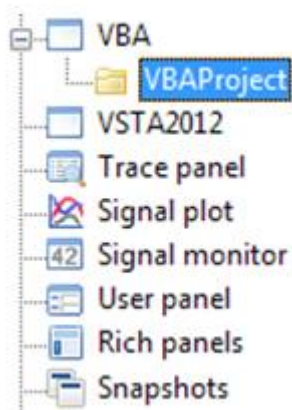
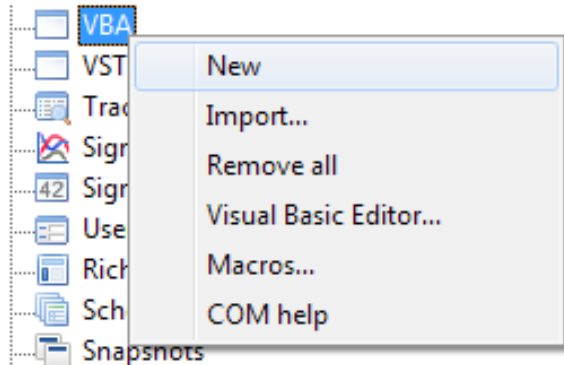
COM – Finding help



- You can open the COM help using the context menu of VBA or pressing F1 inside of the IDE
- Online examples can be found on GitHub: <https://github.com/HolgerDahinten/CanEasySamples>
- Online API: caneasy.de/api

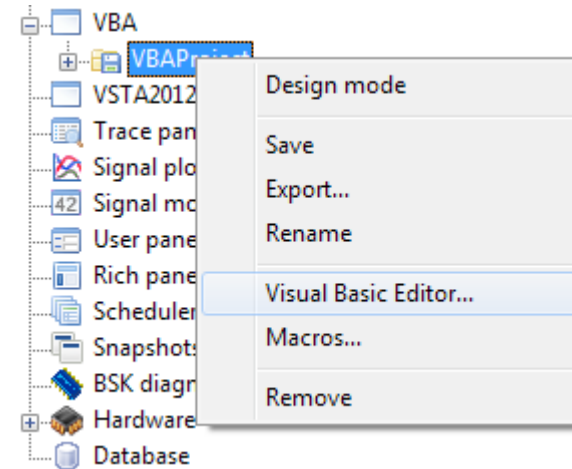


VBA – Create a project

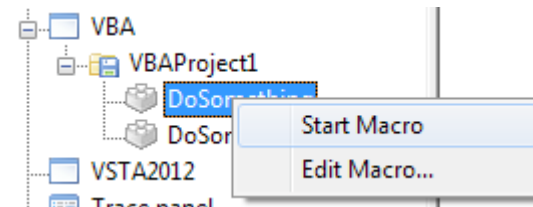


- Create new projects using the context menu of the VBA or VSTA2012 tree entries
- Using VSTA you can choose between C# and VB
- All project sources are stored in the workspace
- Import and export is possible
- Use project references to other projects to implement basic libraries

- Open the VBA IDE (equal to Excel) via the projects context-menu



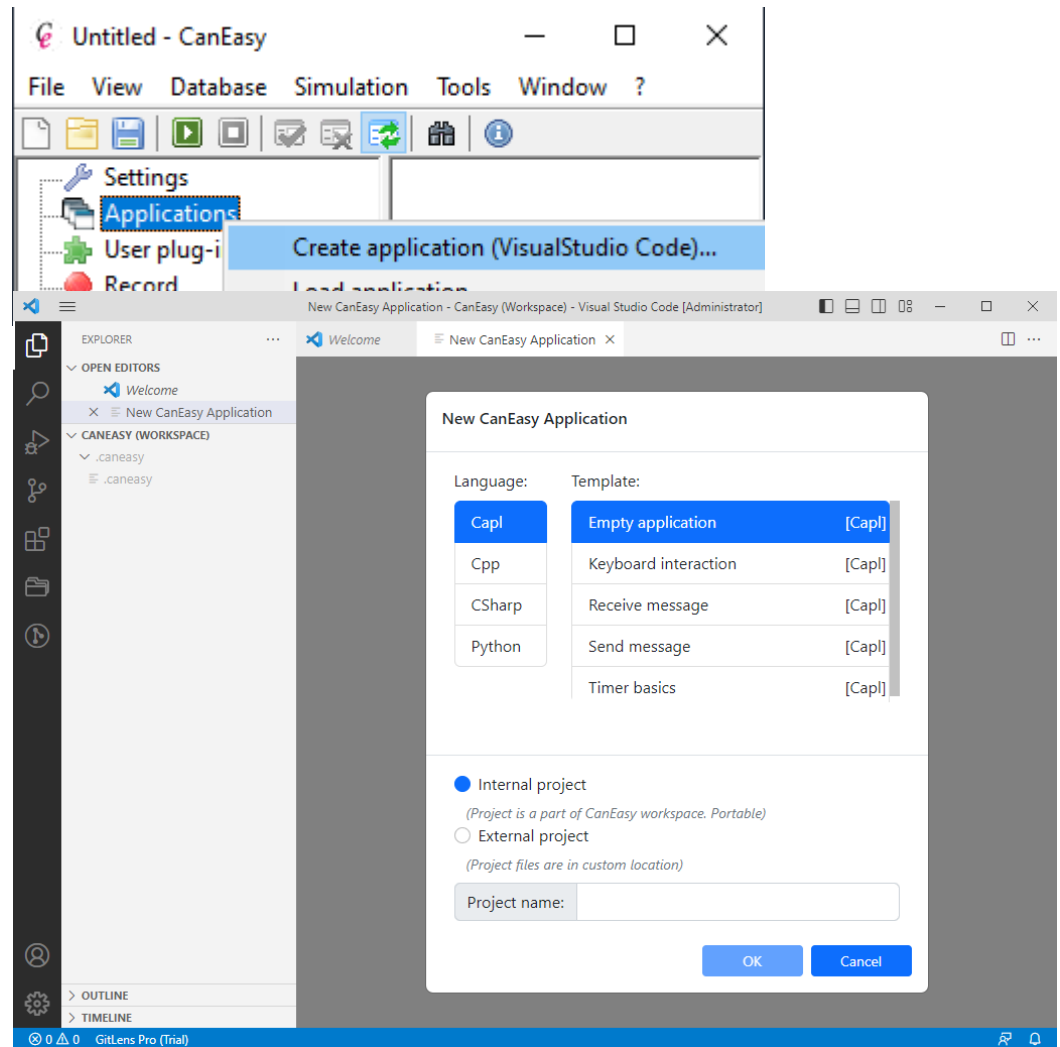
- Start public macros directly from the CanEasy tree



- MultiStudio integrates Visual-Studio-Code into CanEasy
- It allows writing code in:
 - C#
 - C++
 - Python
 - JavaScript
 - CAPL

MultiStudio – Create a project

- Via the context menu you can create a new application
- Choose your programming language and some code template
- Enter a name for your project
- Decide whether you want an internal or external application



MultiStudio – Internal and External Apps

- Internal projects are stored in the CanEasy workspace (csm file)
 - You can just copy the workspace on another machine and it will run
 - External projects are stored just as reference. They must be copied together with the workspace
 - Code should be placed relative to the workspace
 - External projects have the benefit of using Git
-

MultiStudio – Compile

- All applications except Python must be compiled before starting
- Press Ctrl+Shift + B to compile
- Or choose menu Terminal -> Run Build Task
- Look into the output window to check for any compiler errors

- Running more than one Python application requires to activate “Multiple Python” mode.
- See CanEasy Settings->Applications.
- Without this setting you can just run one Python script which can be required from some external libraries.

MultiStudio – Single run

- If you write an application that needs to run only once, you can call the Exit function.
- This stops the execution of your application.
- Because Exit will not terminate the thread don't forget to add return to stop further execution.

```
virtual void OnStarting() override
{
    CanEasy().MakeReport("Hello, World!", ReportType::ReportTypeInformation);
    Exit();
    return;
}
```

- **OnStarting**
function is called when the application is started.
 - **OnStopping**
is called when application is stopped.
 - **OnSimulationStarted**
is called when simulation was started.
 - **OnSimulationStopped**
is called when simulation is stopped.
-

- The start mode defines when your application is started
- Following modes are supported

Mode	Description
On simulation start	Application gets started and stopped together with the CanEasy simulation
Workspace loading	Application gets started directly when the workspace was loaded
Manually only	Application gets started manually by context menu

- Use **Database** to generate your communication matrix dynamically or to handle value change events
 - Use timer events from **TimerControl** to do cyclic stuff or timeouts.
 - Process received and transmitted messages using **TransmissionEvent**
 - Analyze the recorded messages using **Record**, **RecordEntry** and **RecordIterator**
 - Use CanEasy logging via **ReportWnd**
 - Integrate CanEasy into other processes using **CanEasyApplication**
 - Remote control CanEasy from other processes using **CanEasyProcess**
 - Write tests and create reports with our **TestReport**
-

- CanEasyApplication is the root entry of all COM objects
- Because it's the application object all properties and methods can be accessed directly from VBA
- Using the application object, CanEasy can be integrated into other processes
- Provides access to the database, timers, the record,

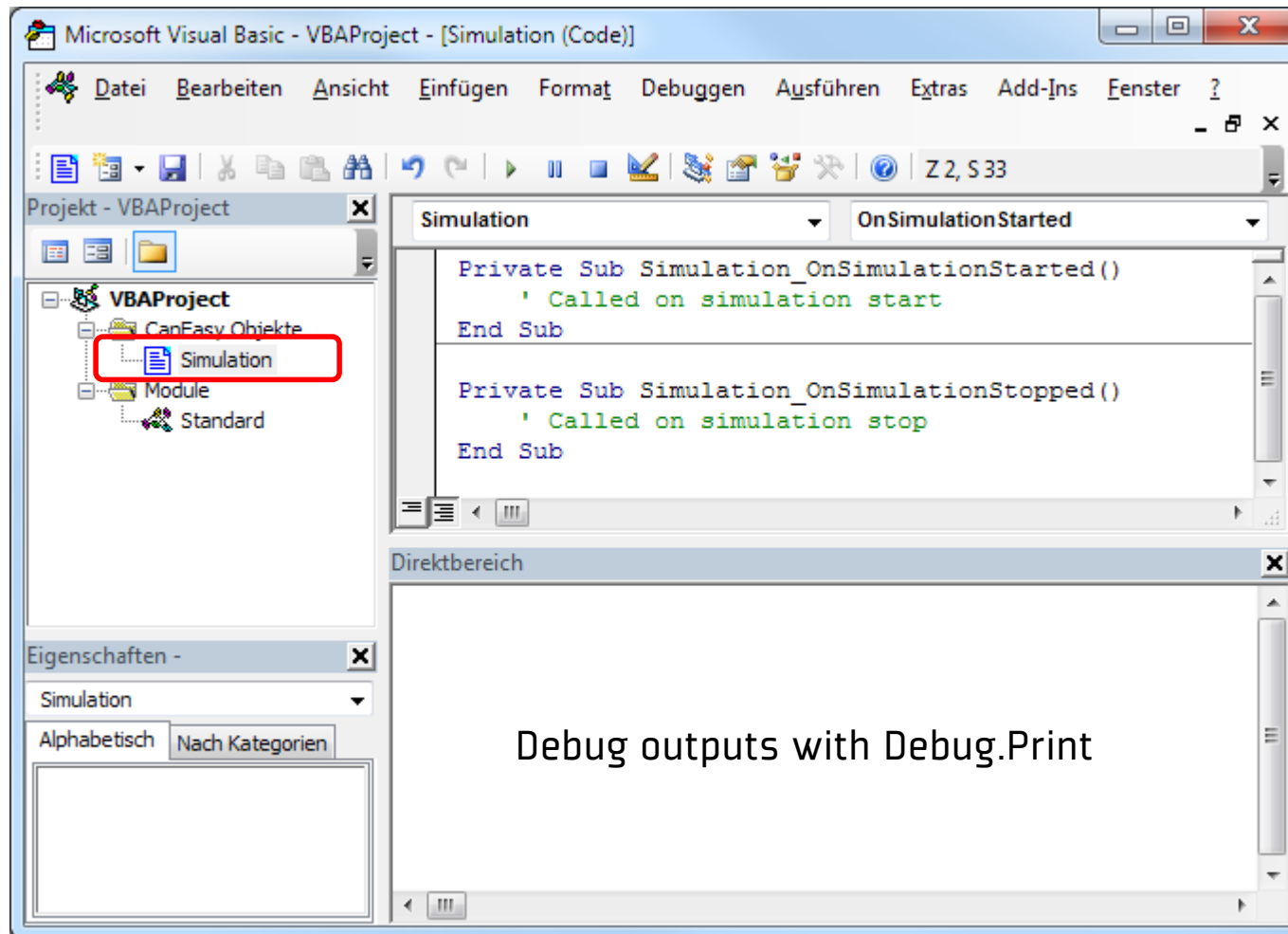
...

```
Sub App_Example ()  
  
    CanEasyApplication.StartSimulation  
    StopSimulation  
  
End Sub
```

- Provides events for simulation start and stop
- In most cases this event is the entry point for your application
- Creating a new VBA project, one global simulation object already exists and can be used to process events

```
Private Sub Simulation_OnSimulationStarted()  
    ' Called on simulation start  
End Sub  
  
Private Sub Simulation_OnSimulationStopped()  
    ' Called on simulation stop  
End Sub
```

COM – Simulation (VBA IDE)



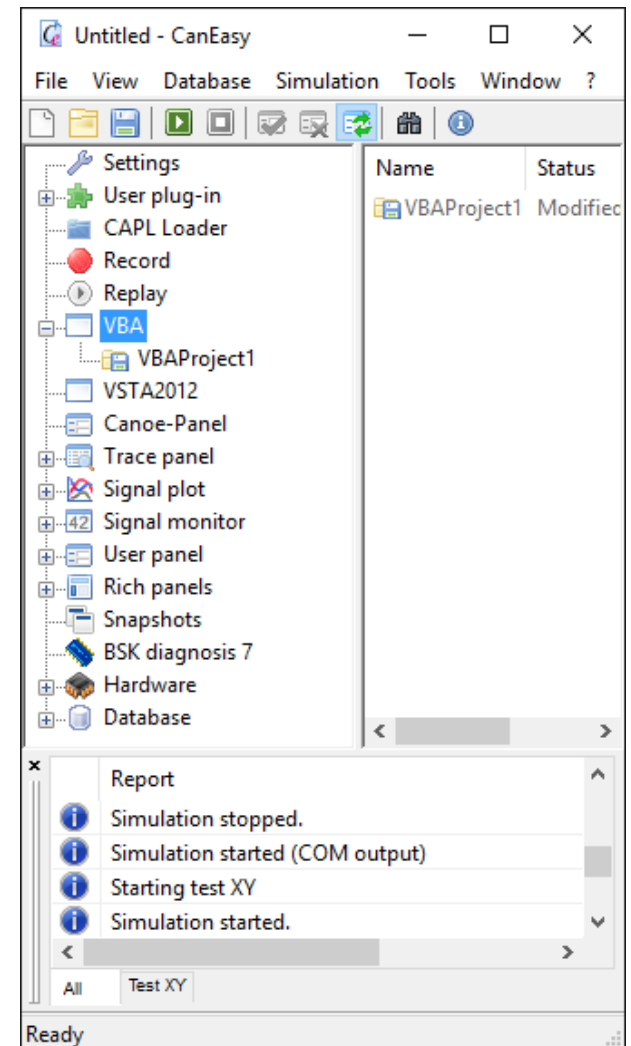
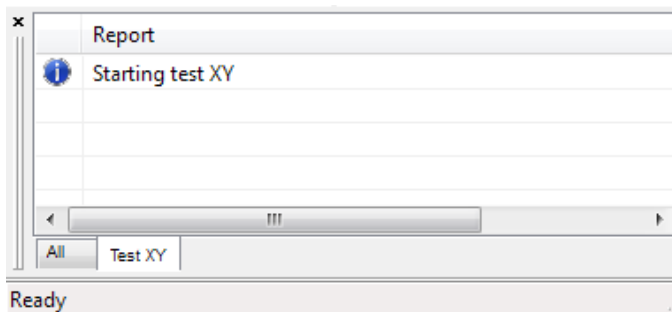
COM – Report window

- For user feedback CanEasy provides a report window
- Using COM you can create as much report tabs as needed
- Every report has one of the following types
 - Information
 - Warning
 - Error
- Call “MakeReport” to write into the global tab

COM – Report window

- The example creates a new report window and writes some test output on simulation start

```
Private Sub Simulation_OnSimulationStarted()  
  
    MakeReport "Simulation started (COM output)", _  
        ReportTypeInfoInformation  
    Dim oReport As ReportWnd  
    Set oReport = ReportWndCol.Add("Test XY")  
    oReport.Write "Starting test XY", ReportTypeInfoInformation  
  
End Sub
```



- The database is the entry point to access channels, ECUs, messages, signals, variables, ...
- Database items can be created, modified or deleted
- Every database item has
 - a name
 - attributes
 - children
 - a parent
- This allows to process the database in recursive way
- To get some specific item **FindObject** or **GetObjectByStringRef** can be used
- To get the path of a database item use Ctrl+ C

```
Private Sub Simulation_OnSimulationStarted()  
  
    Dim oSig As Signal  
    Set oSig = Database.GetObjectByStringRef("Sig:CurrentSpeed")  
    oSig.ValuePhys = oSig.ValuePhys + 1  
  
End Sub
```

COM – Create a database

- Every database item is inside a collection
- The collection can be used to insert new items
- Every database item must have a unique name within the collection
- Creating new items is also possible if simulation is started

```
Public Sub CreateDatabase()  
  
    Set oBus = Database.Busses.AddBus(BUSTYPE_CAN, "MyBus")  
    Set oEcu = oBus.ControlUnits.AddControlUnit("MyEcu")  
    Set oMsg = oEcu.Messages.AddMessage("MyMsg", &H123, 8)  
    Set oSig = oMsg.Signals.AddSignal("MySig", 0, 8, BYTEORDER_INTEL)  
  
End Sub
```

COM – Iterate over database

- Use Children to iterate recursive
- Every IDatabaseItem has
 - Name
 - Description
 - Attributes
 - Children
 - Parent

```
Private Sub PrintItem(oItem As IDatabaseItem, space As Long)
    Dim output As String
    For i = 0 To space
        output = output + " "
    Next
    output = output + oItem.Name
    Debug.Print output

    Dim oChild As IDatabaseItem
    For Each oChild In oItem.Children
        PrintItem oChild, space + 1
    Next
End Sub

Sub PrintDatabase()
    PrintItem Database, 0
End Sub
```

- Use IsValue helper functions to wait for a value with a timeout
- See:
 - IsValueEqual
 - IsValueGreater
 - IsValueLess
 - IsValueLessOrEqual
 - IsValueGreaterOrEqual
 - IsValueInRange

```
Const Sig = "Sig:Signal"

Sub Test_Example()

    ' Check Sig = 5
    If Not Database.IsValueEqual(Sig, 5, 100) Then _
        Debug.Print "Value of signal not equal to 5 within 100 ms -> " & Database.GetValue(Sig)

End Sub
```

COM – SetValue and GetValue

- Use functions to write/read database values
- See:
 - SetValue
 - GetValueString
 - GetValue

```
Const Sig = "Sig:Signal"

Sub Test_Example()

    ' Set Sig to physical value
    Database.SetValue Sig, 5

    ' Use string of value table of physical value
    Database.SetValue Sig, "6"

    ' Use GetValueString to get value table or phys value
    MakeReport "Signal = " & Database.GetValueString(Sig), ReportTypeInfoation

    ' Use GetValue to get physical value
    MakeReport "Signal = " & Database.GetValue(Sig), ReportTypeInfoation

End Sub
```

COM – Create sequences using Sleep

- Simple sequences can be created by using `CanEasyApplication.Sleep`
- Displayed speed must always be lower than current speed

```
Sub TestValue(oSigCurSpd As Signal, oSigDispSpd As Signal, dVal As Double)

    oSigCurSpd.ValuePhys = dVal
    Sleep 500
    If oSigDispSpd.ValuePhys > dVal Then _
        MakeReport „Invalid DisplaySpeed detected“, ReportTypeError

End Sub

Sub TestSequence_DisplaySpeed()

    Dim oSigCurSpd As Signal
    Dim oSigDispSpd As Signal
    Set oSigCurSpd = Database.GetObjectByStringRef("Sig:CurrentSpeed")
    Set oSigDispSpd = Database.GetObjectByStringRef("Sig:DisplaySpeed")

    TestValue oSigCurSpd, oSigDispSpd, 0
    TestValue oSigCurSpd, oSigDispSpd, 100
    TestValue oSigCurSpd, oSigDispSpd, 200

End Sub
```

COM – Transmit and Execute

- Both is sending a message
- Execute also executes a scheduler table
- See:
 - Execute
 - TransmitRef

```
Sub Test_Example()  
  
    ' Execute schedule table to send message  
    Database.Execute "Tab:Tabelle"  
  
    ' Transmit message (or service)  
    Database.TransmitRef "Msg:Botschaft"  
  
End Sub
```

- CanEasy provides different timers
 - The timers have a resolution of one millisecond
 - Types of timers:
 - TimerControl
 - Allows to set a value in milliseconds
 - Supports different modes like one shot
 - If the timer is started the registered event handler will be called cyclically
 - CaptureTimer
 - Allows to capture user defined data together with some timestamp
 - Can be used in time critical context (fast memory management)
 - CompareTimer
 - Can be initialized with content of the CaptureTimer for play back
 - Calls the registered event handler corresponding to the configured timer list
-

- Useful for doing something cyclically or to implement timeouts

```
Private WithEvents oTimer As TimerControl
Private oSig As Signal

Private Sub Simulation_OnSimulationStarted()

    Set oSig = Database.GetObjectByStringRef("Sig:CurrentSpeed")
    Set oTimer = CreateTimerControl
    oTimer.Value = 1000
    oTimer.Start

End Sub

Private Sub oTimer_OnTimer()

    oSig.ValuePhys = oSig.ValuePhys + 1

End Sub
```

- On simulation start, get signal reference from database
- Create a new timer with 1000 ms and start it
- After simulation start oTimer event will be called every second and increments the signal value

COM – Value change events

- Every value item supports events for value change
Examples: Messages, signals, variables, attributes
- By default event handler is asynchronous
- Change events must be activated using **DataChangeEvents** property

```
Private WithEvents oSig As Signal

Private Sub Simulation_OnSimulationStarted()

    Set oSig = Database.GetObjectByStringRef("Sig:CurrentSpeed")
    oSig.DataChangeEvents = True

End Sub

Private Sub oSig_OnChanged()

    oSig.Send

End Sub
```

- On signal value change send corresponding message

COM – Transmission events

- CanEasy supports events for transmitted and received messages
- Transmission events are supported by database, busses ECUs, messages and signals
- It provides various optional filters like message ID ranges, channel and payload
- Events are queued and notified asynchrony

```
Private WithEvents oMsg As Message

Private Sub Simulation_OnSimulationStarted()
    Set oMsg = Database.GetObjectByStringRef („Msg:Motor1“)
    oMsg.TransmissionEvent.Active = True
End Sub

Private Sub oMsg_OnTransmission(ByVal transmission As CanEasy.TransmissionData)
    If transmission.Received Then
        MakeReport transmission.Message.Name & " received", ReportTypeInformation
    Else
        MakeReport transmission.Message.Name & " transmitted", ReportTypeInformation
    End If
End Sub
```

- As event parameter you can access the TransmissionData object which provides the following information
 - Timestamp
 - Direction [rx,tx]
 - Bus object
 - Message object containing signals to access physical values
 - Message ID and payload

```
Private WithEvents oMsg As Message

Private Sub Simulation_OnSimulationStarted()
    Set oMsg = Database.GetObjectByStringRef("Msg:Motor1")
    oMsg.TransmissionEvent.Active = True
End Sub

Private Sub oMsg_OnTransmission(ByVal transmission As CanEasy.TransmissionData)
    If transmission.Received Then
        MakeReport transmission.Message.Name & " received", ReportTypeInformation
    Else
        MakeReport transmission.Message.Name & " transmitted", ReportTypeInformation
    End If
End Sub
```

COM – Pre transmit events

- Similar to the transmission events you can register events that are notified before a message is transmitted by CanEasy
- From this event you have the possibility to change the payload to be transmitted
- Execution must be fast because time critical threads must wait till event was processed!
- It is also possible to deny sending a message

```
Private WithEvents oMsg As Message
Private speedVal As Long

Private Sub Simulation_OnSimulationStarted()
    Set oMsg = Database.GetObjectByStringRef("Msg:Motor1")
    oMsg.PreTransmitEvents = True
End Sub

Private Sub oMsg_OnPreTransmit(ByVal transmission As CanEasy.TransmissionData, _
                                ByVal allowSend As CanEasy.BoolValue)
    transmission.Data.Byte(1) = speedVal
    speedVal = speedVal + 1
End Sub
```

COM – Pre copy events

- Equal to the Pre transmit event there is some pre copy event which is notified before received messages are processed by CanEasy
- It is possible to deny messages or to change received payload

COM – Access recorded messages

- The following example shows how to create a **RecordFilter** and a **RecordIterator** to process all messages.
- Processing the record is also possible while simulation is running

```
Sub ProcessRecord()  
  
    Dim oFilter As RecordFilter  
    Set oFilter = Record.CreateFilter(RecordEntryType.MsgRecordEntry)  
  
    Dim oIter As RecordIterator  
    Set oIter = Record.CreateIterator(oFilter)  
  
    While oIter.Next  
        Dim oMsgEntry As MsgRecordEntry  
        Set oMsgEntry = oIter.RecordEntry  
        Call MakeReport(oMsgEntry.Timestamp & vbTab & _  
                        oMsgEntry.BusName & vbTab & _  
                        "0x" & Hex(oMsgEntry.Id) & vbTab & _  
                        oMsgEntry.DLC & vbTab, ReportTypeInfo)  
    Wend  
  
End Sub
```

COM – Workspace module

1. Using the COM interface you can save user data into the CanEasy workspace (csm file) by creating a global object of **WorkspaceModule**
2. If the user saves the workspace **WorkspaceModule.OnSaveWorkspace** will be called
3. Initialize **WorkspaceModule.ModuleData** with your user data
4. When a new workspace was loaded **Simulation.OnWorkspaceLoaded** is called
5. There you need to call **WorkspaceModule.ReloadModuleData** to get event for
6. **WorkspaceModule.OnLoadWorkspace**. As paramter you get your stored data

```
Private WithEvents oWorkspaceModule As WorkspaceModule 1.

Private Sub oWorkspaceModule_OnSaveWorkspace(ByVal success As CanEasy.BoolValue) 2.
    oWorkspaceModule.ModuleData = Array("Some data", 11) 3.
    success = True
End Sub

Private Sub Simulation_OnWorkspaceLoaded() 4.
    Set oWorkspaceModule = CreateWorkspaceModule("MyModuleName")
    oWorkspaceModule.ReloadModuleData 5.
End Sub

Private Sub oWorkspaceModule_OnLoadWorkspace(ByVal data As Variant, \ 6.
                                           ByVal success As CanEasy.BoolValue)

    For Each oEntry In data
        Debug.Print oEntry
    Next
    success = True
End Sub
```

COM – Integration into a process

- CanEasy can be integrated into other processes
- By default CanEasy will show no main window and no blocking message boxes
- Useful if you have time-critical requirements
- COM reference to **“Schleissheimer CanEasy Typbibliothek”** must be set
- CanEasyApplication.Init must be called to load all plug- ins into the process space

```
Sub Example_Integrate_CanEasy()  
  
    Dim oApp As CanEasy.CanEasyApplication  
    Set oApp = New CanEasy.CanEasyApplication  
    oApp.Init  
  
    ' Open main window of CanEasy  
    oApp.AppWindow.Open  
  
End Sub
```

- The CanEasy process can be controlled by other processes
- Creating **CanEasyProcess** starts CanEasy
- If CanEasy is already started, CanEasyProcess is attached to it
- COM reference to “**Schleissheimer CanEasy Process Typbibliothek**” must be set
- Call **GetApplication** to get CanEasyApplication root entry
- Examples for Robot-Framework, LabView, Matlab Simulink

```
Sub Example_Remote_CanEasy()  
  
    Dim oProcess As CanEasyProcess.CanEasyProcess  
    Set oProcess = New CanEasyProcess.CanEasyProcess  
    oProcess.KeepAlive  
  
    Dim oApp As CanEasy.CanEasyApplication  
    Set oApp = oProcess.GetApplication  
  
End Sub
```

- Use the TestReport class to generate XML and HTML reports

```
Sub TestReport()  
  
    Dim oTest As TestReport  
  
    Set oTest = New TestReport  
  
    oTest.StartTestReport "c:", "TestReport"  
  
    oTest.OpenGroup "group", "desc", "id"  
  
        oTest.OpenTestcase "testcase1 name", "testcase1 desc", "testcase1 id"  
        oTest.StepWarning "step1 id", "step1 desc"  
        oTest.Step "step2 id", "step2 desc"  
        oTest.StepPass "step3 id", "step3 desc", 1  
  
    oTest.CloseGroup  
  
    oTest.OpenTestcase "testcase2 name", "testcase2 desc", "testcase2 id"  
        oTest.OpenPattern "pattern name", "pattern desc", "pattern id"  
        oTest.StepFail "fail id", "fail desc"  
  
    oTest.CloseTestReport  
  
End Sub
```

Thank you for your attention!
