1 Testo “Toolbox”

The Testo Toolbox is a program library which makes it possible to easily activate Testo AG measuring instruments from your own application program.

The interface is independent of instrument. In this way, you can always activate the functions which are available in all of the instruments in the same way. They are as follows:

- Set up connection to instrument
- Read out current readings (online measurement)
- Save identifier (sites)
- Read out saved readings (protocols)

Instrument-specific extensions are available for some instruments (testo 174, testo 175, testo 177, testo 350 M/XL) using which you can benefit from additional features in the instrument. For example, you can set up a logger program such that a specific measurement is carried out at a specified time.

The Toolbox is set up as an OLE Automation Server. In this way, integration with each programming language supporting ActiveX Controls can take place. They are for example:

- Microsoft® Visual Basic from Version 5
- Microsoft® Visual Basic .NET
- Microsoft® Visual Basic for Applications (VBA) as included in Microsoft Word, Excel and Access.
- Microsoft® Visual C++ from Version 5
- Microsoft® Visual C++ .NET
- Borland Delphi® Version from Version 3
- National Instruments LabView® from Version 5

The Toolbox is installed together with Comfort Software. Online documentation for the interface is also installed together with sample programs for Microsoft® Visual Basic, Microsoft® Visual C++ and National LabView®.

Contents:

2 Working with the Testo “Toolbox”

First, establish the type information of your programming environment:

Microsoft VB from version 5	Project, References	Tcddka
Microsoft VC from version 5	#import	<installation folder>\Tcddka.dll

In most other languages, you will find what you are searching for in the „ActiveX controls“ category using the keyword “tcddka”.

2.1 Connecting the instrument

The object tcddk requires special care by you, and in particular the “Life time” should be checked. One frequent error is the use of global variables. They are usually “closed” in random order at the end of the program which inevitably leads to problems. Tcddk requires a manual “close”. In Visual Basic® by allocating “nothing”.

No two objects can operate simultaneously with a connected instrument. This is pointed out again in Can connections.

VB

```
Dim inst As tcddk
Dim bRet As Boolean

Set inst = New tcddk
bRet = inst.Init(0, "testo175-177", 5000)

If True = bRet Then
    '...
End If

Set inst = Nothing
```

2.1.1 Instruments with serial interface

Use the method Tcddk::Init to start working with a serially connected instrument.
Init has three parameters

WidCom	Number of COM Port	0 based, use 0 for the first serial interface (COM1) etc.
LpszDevicename	Instrument code	Following values are valid (watch out for upper and lower cases): testostor175, testostor171 testo400-650-950

		testo445-645-945-946-545 testo454-2000 testo175-177 testo174 testo300-M-XL
DwTimeOut	Max. initialisation time in [msec].	Varies according to instrument. Since initialisation also reads the complete directory of the instrument memory, a successful initialisation in t400, t454-2000 can take up to 30 seconds.

Init supplies “True” or “False” as result. Since initialisation is a critical procedure which can fail for many reasons, the asynchronous error remedy function in your programming environment should be used: C++Exceptions or Visual Basic® OnError directives. An error object which may occur contains a description of the possible cause.

VB

```
On Error GoTo ErrorLabel
Dim inst As tcddk
Dim bRet As Boolean

Set inst = New tcddk
bRet = inst.Init(1, "testo175-177", 5000)

If True = bRet Then
    '...
End If

GoTo EndLabel

ErrorLabel:
MsgBox Err.Description

EndLabel:
Set inst = Nothing
```

2.1.2 Instruments with Businterface

Bus instruments are activated using their bus address. The object Tcbus determines the addresses of all the Testo instruments connected to the bus (provided they are ready to operate). Tcbus::item supplies the corresponding object tcddk.

Item supplies the known tcddk object. Please note that it is no longer necessary to call up with tcddk::init. This object is already initialised.

Item creates a new tcddk object each time it is called up. Repeated calls on the same item (with the same address) can lead to unwanted results. Ensure that an item is “closed” (in Visual Basic® by allocating “Nothing”) before using the address again.

```
VB
Dim bus As tcbus
Set bus = New tcbus

bus.Init

Dim dev(255) As tcddk

Dim addr As Variant

i% = 0
For Each addr In bus
    Set dev(i) = bus.Item(addr)
    i% = i% + 1
Next
```

2.1.3 Instruments with USB interface

Please use the function InitSerial for connecting an instrument with USB interface, as e. g. testo 435,635,735.

```
VB
Set dev = New tcddk
dev.InitSerial serial, "testo435-635-735", 5000
Set dev = Nothing
```

The parameter “serial” is the serial number of the instrument here. It is printed on the label.

The data loggers 175-177 are offered as well with USB interface as an option. Here however a serial interface is emulated. For the Testo Toolbox these are no USB instruments.

Serial numbers can be determined as well program-controlled. The object tcusbserials determines the serial numbers of all Testo instruments of a specific type connected via USB.

```
VB
Dim usbserials As tcusbserials
Set usbserials = New tcusbserials

usbserials.Init ("testo435-635-735")

Dim serial As Variant

For Each serial In usbserials
' …
Next
```

2.2 Online measurement

Ready-to-measure instruments are indicated in tcddk::NumCols by a value greater than 0.

VB

```
If inst.NumCols > 0 Then
        inst.Get
        For i% = 0 To inst.NumCols - 1 Step 1
            If inst.IsNonNumeric(i%) Then
                msg$ = Format(i%, "0")
                msg$ = msg$ + " " + inst.StringVal(i%)
                MsgBox (msg$)
            Else
                msg$ = Format(i%, "0")
                msg$ = msg$ + " " + Format(inst.RecentVal(i%),
"##0.0") + inst.Unit(i%)
                MsgBox (msg$)
            End If
        Next i
    End If
```

Tcddk::MinRate issues the shortest possible repeat interval for Online “Queries”. The value is in [msec]. If you call up tcddk::Get() time-controlled, avoid falling short of this limit. If the limit is fallen short of, this may lead to invalid results.

Tcddk::MinRate depends on the instrument and probe assignment. It is not correct to determine Tcddk::MinRate once and then to use it as a constant. Other instruments may require a different repeat interval.

2.2.1 Multithreading

Multithreading is a subject for the more advanced. Tcddk::Get gets updated data from your instrument. Since it may take a while, the calling thread is put on hold until the result arrives from the instrument. Putting on hold implies that only very little computing time is used up. Make sure when using several threads that COM objects are working in the thread which creates the object. Method queries over thread limits could produce unexpected results.

2.3 Working with instrument memory

Saved measurement data is filed as a protocol (“Protocol”). The memory organisation of the individual instruments is in harmony with the respective application and extends from long-term memories which can only manage one protocol to memory hierarchies with folder structure.

In order to make it easier for simple programming languages to handle memory content, there are batch objects which combine objects of the type. Tfolders, Descs and Protocols are found here.

The “tcddk”contains three batch objects (collections) from which, depending on the instrument, only one or two result in protocols. The other properties provide the value ZERO in the respective case.

Tcddk::<Property>	Valid for instruments:	
Tfolders	T400	
Descs	Tx45, T400, T454-2000	
Protocols	T171, T174, T175-177	

2.3.1 Example: T177

VB

```
Dim inst As tcddk
Dim bRet As Boolean
Dim prot As Protocol
Dim key As Long

On Error GoTo ErrorLabel

Set inst = New tcddk
bRet = inst.Init(1, "testo175-177", 5000)

If True = bRet Then
    For Each prot In inst.Protocols
        key = prot.NumKey
        msg$ = Format(key, "000")
        MsgBox (msg$)
    Next
End If

GoTo EndLabel

ErrorLabel:
MsgBox Err.Description

EndLabel:
Set inst = Nothing
```

2.3.2 Example: Tx45

VB

```
Dim inst As tcddk
Dim bRet As Boolean
Dim desc As desc
Dim prot As Protocol
Dim key As Long
Dim pc As ProtocolCollection
```

```
On Error GoTo ErrorLabel

Set inst = New tcddk
bRet = inst.Init(1, "testo445-645-945-946-545", 5000)

If True = bRet Then
    For Each desc In inst.Descs
        Set pc = desc.Protocols
        If pc.Count > 0 Then
            For Each prot In pc
                key = prot.NumKey
                msg$ = desc.key + " " + Format(key, "000")
                MsgBox (msg$)
            Next
        End If
        Set pc = Nothing
    Next
End If

GoTo EndLabel

ErrorLabel:
MsgBox Err.Description

EndLabel:
Set inst = Nothing
```

Watch out for the `pc.Count > 0` query. ForEach-like constructs should not be used to iterate over an empty collection.

Note the explicit initialisation of the variables allocated with set (`pc = Nothing`).

2.3.3 Example: T400

Memory hierarchies with folder structures can be worked through. The example uses the function `WalkDescs` to list all the sites of a hierarchy level. `WalkFolder` lists all the sites and locations of a hierarchy level. `WalkFolder` is called up for every sub-folder batch. Compare this with a procedure which processes the files and directories of a hard disk. Tfolder corresponds to the directory, Desc of a file. A “path” with the following setup: [Tfolder]* Desc NumKey runs simultaneously to the protocol.

VB

```
Private Sub WalkDescs(ADescCollection As DescCollection)
Dim ADesc As Desc
Dim AProtocolCollection As ProtocolCollection
Dim AProt As Protocol

If ADescCollection.Count > 0 Then
    For Each ADesc In ADescCollection
```

```
        MsgBox (ADesc.key)
        Set AProtocolCollection = ADesc.Protocols
        If AProtocolCollection.Count > 0 Then
            For Each AProt In AProtocolCollection
                key = AProt.NumKey
                msg$ = ADesc.key + " " + Format(key, "000")
                MsgBox (msg$)
            Next
        End If
        Set AProtocolCollection = Nothing
    Next
End If
End Sub

Private Sub WalkFolder(ATFolderCollection As TFolderCollection)

Dim ATFolder As TFolder
Dim MoreTFolders As TFolderCollection
Dim ADescCollection As DescCollection

If ATFolderCollection.Count > 0 Then
    For Each ATFolder In ATFolderCollection
        MsgBox (ATFolder.key)

        Set ADescCollection = ATFolder.Descs
        WalkDescs ADescCollection
        Set ADescCollection = Nothing

        Set MoreTFolders = ATFolder.TFolders
        WalkFolder MoreTFolders
        Set MoreTFolders = Nothing
    Next
End If

End Sub

Private Sub Ex4_Click()

Dim inst As tcddk
Dim bRet As Boolean
Dim ADescCollection As DescCollection
Dim ATFolderCollection As TFolderCollection


On Error GoTo ErrorLabel

Set inst = New tcddk
bRet = inst.Init(1, "testo400-650-950", 20000)
```

```
If True = bRet Then
    Set ADescCollection = inst.Descs
    WalkDescs ADescCollection
    Set ADescCollection = Nothing

    Set ATFolderCollection = inst.TFolders
    WalkFolder ATFolderCollection
    Set ATFolderCollection = Nothing
End If

GoTo EndLabel

ErrorLabel:
MsgBox Err.Description

EndLabel:
Set inst = Nothing

End Sub
```

Please ensure that the above examples always use the ForEach command in order to process all batch objects in one loop. The simple reason is that the Protocols::Item method needs a value as a parameter which can only be taken from a protocol as Protocol::Numkey Attribut – a classical chicken egg problem. If your user is to be able to select from an overview at a later stage, it is best to save the corresponding key values when going through the instrument memory.

If you should determine that Numkey supplies a trivial sequence of the type 0,1,2 it should not be seen as a generalisation. Other assignments are technically possible. If your program is based on this type of assumption, at some stage it is possible that it will not function properly.

z.B.:

```
For Each p in inst.Protocols
        Keys(i) = p.NumKey
        i = i+1;
Next
```

Subsequent “read out” will then be as follows:

```
Set p = inst.Protocols.Item(Keys(selected))
p.Get();
```

2.4 Additional functions not specific to instrument

Ident	The serial number of your instrument

2.5 Additional functions specific to instrument

Additional functions are available for different instruments at the programming interface. In addition to tcddka, this type information has to be made known to your programming environment.
Access is via the property tcddk::Instrument. It supplies an object which is then allocated to the variable of a corresponding type. Corresponding type means: you now have to name your instrument. testo175-177 is programmed in the following example. The type library is t177a, the instrument in this case is T177aInstrument. Other instruments will show objects from T<xxx>aInstrument type. Allocation can fail if the type is incompatible or a detailed interface is missing.

VB

```
Dim tcd As tcddk
Dim Inst As T177aInstrument
Set tcd = New tcddk
bInit = tcd.Init(0, "testo175-177", 20000)  ' use com1
If (True = bInit) Then
    Set Inst = tcd.Instrument
    ...
    Set Inst = Nothing
    tcd.Exit
End If
Set tcd = Nothing
```

2.5.1 testo 175-177

Type library: T177a

T177aInstrument contains subordinated objects:

Program	T177aProg	Measurement program
Configuration	T177aConf	General settings such as Display on/off
Sockets	T177aSocketCollection	Definition of “probe”configuration

Each of these sub-objects includes a “Save” method using which modified settings are transmitted to the instrument.

Most modifications require a break in the measurement for ::Save. Note T177aInstrument::Mode. Valid values in this case are T177_WAIT and T177_END. Use TxxxAProg::Stop to stop the measurement, if necessary.

Only one single protocol can be managed by the 175-177 logger. ::Save renders saved data invalid, i.e. they are irreversibly lost. Save the contents of your instrument memory prior to ::Save.

See Online Help “t177a.hlp” for details

2.5.2 testo 174

Type library: T174a

2.5.3 testo 300XXL, testo 350 M/XL

The t35na extension specific to instrument is available for the testo 300XXL, testo 350 M/XL measurement system.

The additions specific to instrument have the following functions.

- The error mode of the instrument can be read out at any time. Several instrument errors can exist at any time. The error number and an error text is issued for each error in the respective language version. Critical errors (e.g. error in instrument) as well as warnings (e.g. sensor spent) are issued. The application program´s job is to react correctly.
- A logger program can be loaded and started..
  - Define the type of start (on command or at a pre-determined time)
  - Define stop conditions (memory full, number of measurements, specified time)
  - Define measurement interval
  - Define gas and fresh air cycle
  - Define fuel
  - Switch pellistor on or off (if available)
  - Define dilution
  - Determine site name
  - Load last logger program, save modified settings
  - Start logger program
  - Stop logger program
  - Call up status (waiting on start, measurement in progress, measurement completed)
- The following features can be called up by the instrument
  - Voltage of rechargeable battery or battery. It is the job of the application program to interpret this voltage e.g. to react if there is a threat of the rechargeable battery discharging.
  - Memory available for measurement data.
  - Name of control element or measurement box. This name can also be written anew.
  - Instrument type query (300 XXL, 350 M, 350 XL, respective control unit or memory box)
  - Name of smoke tester
- Instrument memory (measurement data and measurement sites) can be deleted completely.

Detailed documentation is included in the t35na.hlp file.

Connection is either via the serial interface from PC to operating unit or directly via a bus connection from the bus insert cards in the PC to the measurement box.

3 Information on specific development environments

3.1 Borland Delphi®

The import of Typelibrary [Project|Import Type Library] creates a “unit” tcddka_TLB.pas®. The generated code may require reworking which is a problem associated with the Delphi development environment.

In the tcddka_TLB.pas file change all the defective function queries of the type DefaultInterface.Get_<xxx>() in DefaultInterface.<xxx>[]. Ensure that the round brackets are replaced by square brackets.

3.2 National Instruments LabView®

When changing version, it may be necessary to update the “Library” setting of all “Property” and “Invoke” nodes. Proceed as follows:

Select the “Automation Refnum” symbol with the name “tcddka.Itcddk” on the front panel of a VI (virtual instrument), cut it out and paste it in again. Set “tcddka.Itcddk” in the “Select ActiveX Class” context menu.

Check the “Property” and “Invoke” nodes in the diagram overview. In the context menu (right mouse button), you will either see “Features” (if it is a property node) or “Methods” (if is an Invoke node). Ensure that the entry in the respective submenu is marked with a suitable method or property name.

3.2.1 Examples

Examples in the Labview5 folder were set up with Labview Version 5.1, examples in Labview6 folder with Version 6.1.

CAN Measure Example : Example of exporting readings online from the Testo instruments (t454) via CAN-BUS.

Serial Measure Example : Example of exporting readings online from Testo instruments via serial interface.

t171 Example : Example of exporting readings online from a t171.

t171_t177 Example : Example of exporting readings online from a t171 or t177.

t350 gas analyser serial Example : Example of exporting readings serially from a t350M/XL measurement box.

Control Unit serial Example : Example of exporting readings serially from a t350M/XL Control Unit.

T454 serial Example : Example of exporting readings serially from a t350M/XL measurement box.

3.2.1.1 Serial instrument connection:

Select instrument example for the Testo instrument at hand.
Select COM interface and set up Waittime, if necessary.
Start VI.

3.2.1.2 CAN instrument connection:

Select **CAN Measure Example.vi** example.
Set CAN-BUS address and set Waittime, if necessary.
Start VI.

3.2.1.3 Integrate in your own VI:

Connect the Icons (Init, Measure, Close), located in the **testo toolbox.llb**, in accordance with the selected interface such as in **Serial Measure Example.vi** or **CAN Measure Example.vi**. Or copy the respective VI diagrams to your own diagram and design the display elements in the front panel as required.

You will find a description of the respective methods and features in tcddk.hlp.