Integration for Visual Basic Interface

Integration for Visual Basic Interface

Version 06-Nov-2017

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Debugger Basics - Training"** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"T32Start"** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **"General Commands"** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:

  - Choose **Help** menu > **Processor Architecture Manual**.

- **"RTOS Debuggers"** (rtos_<*x*>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate RTOS manual informs you how to enable the OS-aware debugging.

Release Information

TRACE32 Visual Basic Interface (VBI), release 1.0, January 2000, requires TRACE32 software dated after 26/10/99. The VBI is compatible to Visual Basic Version 6.0. The example for Microsoft Excel is produced with Excel 97.

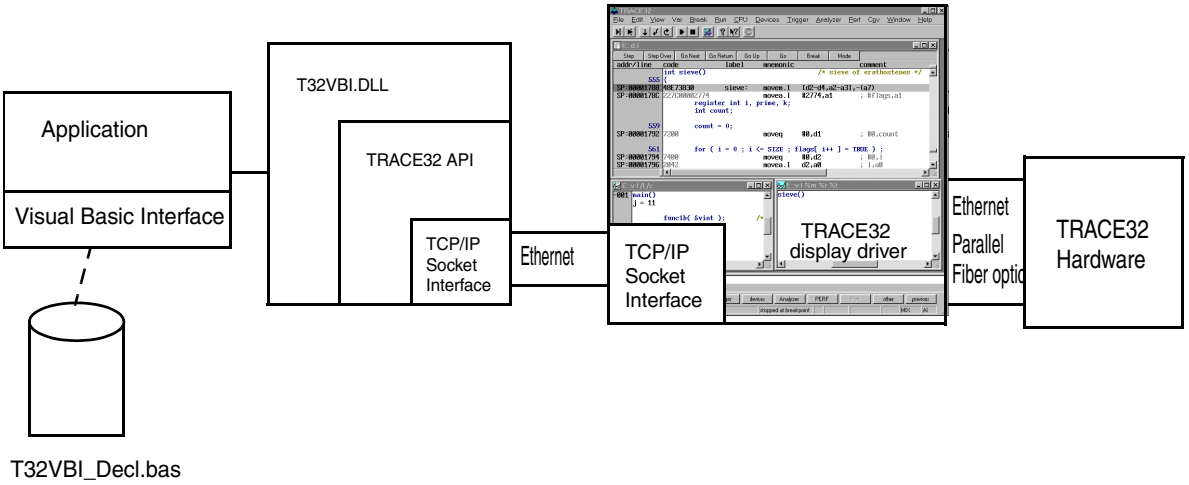
Release 1.2, December 2008, adds multi-channel support to allow communication with more than one TRACE32 PowerView GUIs at the same time. FDX and low-level JTAG API calls will be added in later releases.

Introduction

The TRACE32 Software distribution contains a Visual Basic interface for external control of TRACE32. This TRACE32 Visual Basic Interface (further referred as VBI) gives any application which Visual Basic (further referred as VB) interface, e.g. Microsoft Word or Microsoft Excel, the possibility to control the emulator.

The TRACE32 VBI is built as a Dynamic Link Library (DLL) which internally uses the TRACE32 Remote Application Programming Interface (TRACE32 Remote API) to control the TRACE32 application. The DLL communicates with the TRACE32 PowerView GUI (not with the TRACE32 PowerDebug device itself!) using TCP/IP socket interface. The command chain using TRACE32 VBI looks like this:

Application--->	TRACE32 VBI--->	TRACE32 display driver-->	TRACE32
(VB Functions)	(sockets)	(HW interface)	

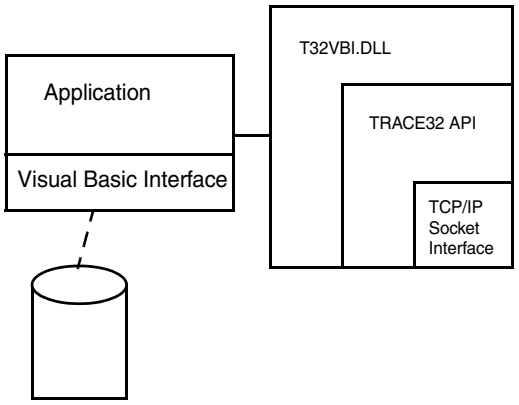


Restrictions in Demo Mode

The TRACE32 VBI is blocked in “demo mode”, i.e. if you do not have a valid TRACE32 license. You will not be able to create successful connections between the VBI and TRACE32.

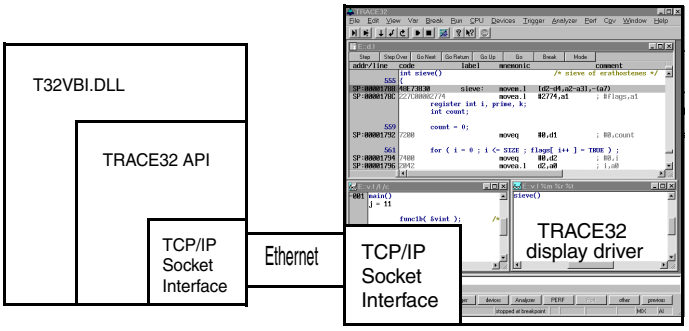
If you need to evaluate the VB without having a full license, contact Lauterbach for an evaluation license of your TRACE32 system.

Application --> TRACE32 VBI



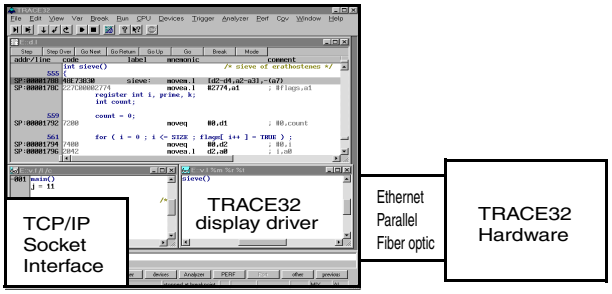
The application uses the TRACE32 VBI through ordinary Visual Basic functions. These are defined in the file T32VBI_Decl.bas.

TRACE32 VBI --> TRACE32 display driver



The communication to the TRACE32 PowerView GUI is implemented as a socket interface. Therefore the VB application using the TRACE32 VBI and the TRACE32 PowerView GUI can reside on two different hosts, using UDP/IP network connections for communication. But be aware that this connection is not fault tolerant, because no network error detection is implemented in the VBI. It is therefore recommended to run both programs on the same host.

TRACE32 display driver --> TRACE32



The TRACE32 PowerView GUI translates and routes the VBI requests to the TRACE32 PODBUS Device. This interface is the one chosen for your debugger or emulator, e.g. it could be Ethernet, USB, parallel or optical interface.

The answers for the request go exactly the opposite way, returning information to the application in passed buffers.

Operation of the VBI Requests

The VBI requests operate parallel to normal TRACE32 operation. You can use both, the TRACE32 window and the VB application using VBI at the same time, although it is not recommended. The application won't be informed about changes that are done in the PowerView GUI. Also, unpredictable errors may occur, if e.g. an VBI request and a running practice file interfere.

VBI Files

The VBI consists of:

- **T32VBI_Decl.bas**

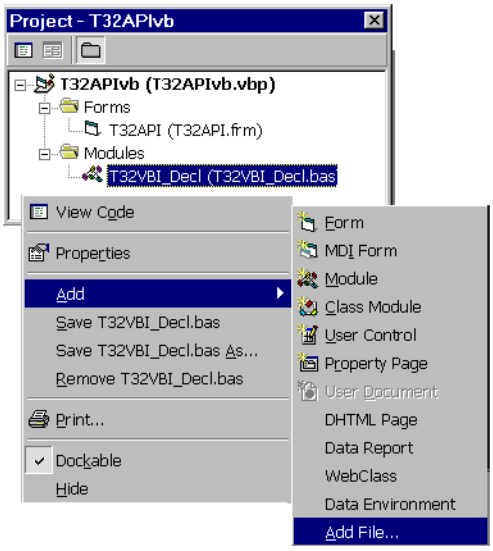
  This file contains the declarations of the VB functions of the VBI needed to be included in your VB application.

- **T32VBI.DLL**

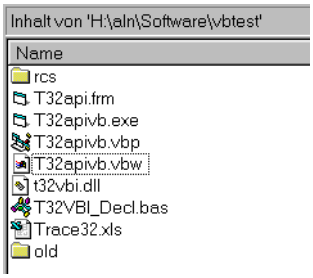
  DLL containing the Visual Basic Interface (VBI) functions.

  Please note that this file is not identical to t32api.dll. (Visual Basic uses a different calling convention than C, and some functions require additional wrapper code to make them easier to use from Visual Basic.)

Whenever a VB application uses the VBI, the file `T32VBI_Decl.bas` must be added to the 'Modules' list of the VB project.



The file `T32VBI.dll` must be copied to directory of your VB project and the PATH environment variable must point to this directory, e.g:



The directory `<t32 installation directory>\demo\api\vbapi\sample\vbapp` contains a VB test project `trace32.vbp`. The file `t32vbi.dll` must be copied into the same directory.

The PATH variable must be set in the system environment or a batch file like this:

```
SET PATH=C:\T32\demo\api\vbapi\sample\vbapp;%PATH%
```

Preparing TRACE32 Device Driver

To use the VBI, TRACE32 has to be set up for use via remote control. To enable and configure remote control, add these lines to your TRACE32 PowerView GUI configuration file (e.g. `config.t32`):

```
RCL=NETASSIST
PACKLEN=1024
PORT=20000
```

`PACKLEN` specifies the maximum package length in bytes for the socket communication. It must not be bigger than 1024 and must fit to the value defined by `T32Config()`.

The port number specifies the UDP port which is used to communicate with the VBI. The default is 20000. If this port is already in use, try one higher than 20000.

Configuring VBI

The VBI must be configured with the commands `T32Config()` and `T32Init()`. `T32Config`() takes two string arguments, both of them concatenated give an option line like they are defined in `config.t32`. Normally this is used to set the `NODE` name and the `PORT` number. The function `T32Init` then does a setup of the communication channel. The `T32Exit` function closes the connection and should always be called before terminating the application.

```
Private Sub Init_TRACE32()
     T32Config "NODE=", "localhost"
     T32Config "PORT=", "20016"
     If T32Init < 0 Then
          MsgBox "Error Initializing TRACE32 (init)", vbCritical,_
           "TRACE32"
          T32Exit
     Else
          If T32Attach(1) Then '1 = T32_DEV_ICD
               MsgBox "Error Initializing TRACE32 (attach)", vbCritical,_
                "TRACE32"
               T32Exit
          End If
     End If
End Sub
```

See chapter "**Generic VBI functions**" for a detailed description of these functions.

What is the reason for `Error-53 t32vbi.dll file not found`**?**

Please check:

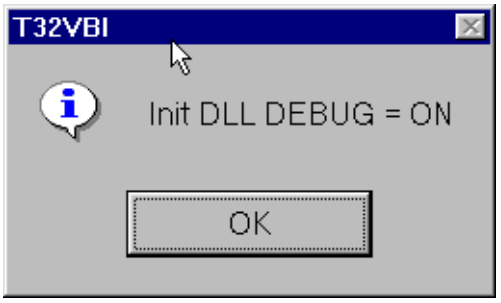
- PATH variable set to correct directories, and contains directory with t32vbi.dll?
- t32vbi.dll exists in this directory?
- wsock32.dll, user32.dll and msvcrtd.dll exist in system directory?

How can I check if the T32VBI.DLL is loaded correctly?

To make sure the VB application has loaded T32VBI.DLL correctly, you can set the environment variable T32VBDEBUG to ON, either in the system environment or a batch file (e.g. in `autoexec.bat`):

```
SET T32VBDEBUG=ON
```

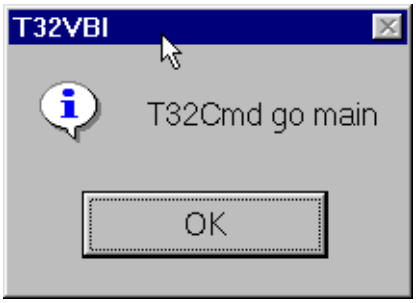
If `T32VBDEBUG=ON` the T32VBI.DLL will show the following message when loading the file into memory, i.e. when calling the first VBI function:

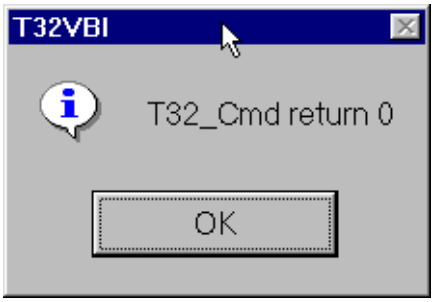


This also activates debug information output for every VBI function (see also T32SetDLLDebug).

How can I check the communication between my VB application and T32VBI.DLL?

Use the VBI function `T32SetDLLDebug(1)` to get information about the given parameters and the return values. E. g. VBI call `ret = T32Cmd("go main")`:





T32SetDLLDebug — Configure debug information output

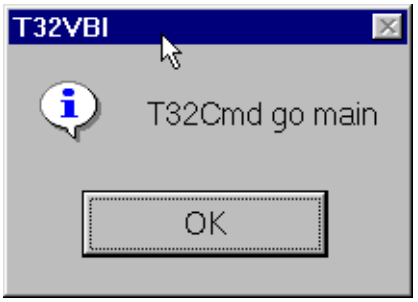
Prototype:

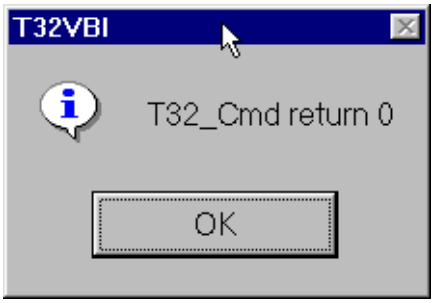
```
Declare Function T32SetDLLDebug Lib "t32vbi.dll" (ByVal mode as Long) _
As Integer
```

Parameters: `mode` mode = 1 give debug information
mode = 0 no debug information is given

Returns: 0 for ok, otherwise Error value

After a call to `T32SetDLLDebug(1)`, every time a VBI function is called, messages similar to these occur:





Prototype:

```
Declare Function T32Config Lib "t32vbi.dll" (ByVal envstring As String, _
ByVal value As String) As Integer
```

Parameters: `envstring`, `value` Commands to configure the UDP communication

Returns: 0 for ok, otherwise Error value

The two strings are concatenated and the resulting command is sent to the communication driver. On UNIX/VMS systems this driver is the standard Ethernet interface driver of TRACE32. All commands described for this interface can be used here. Usually three commands will be used:

```
NODE=localhost
PACKLEN=1024
PORT=20000
```

`NODE` defines, on which host the TRACE32 display driver runs - normally localhost.
`PACKLEN` specifies the maximum data package length and must not be bigger than 1024 and must fit to the value defined in the `config.t32` file (see chapter 3.1).
The `PORT` command defines the UDP port to use. If omitted, it defaults to 20000. Be sure that these settings fit to the `RCL` settings in the `config.t32` file.

Example:

```
Dim ret As Integer

ret = T32Config("NODE=", "localhost")
ret = T32Config("PACKLEN=", "1024")
ret = T32Config("PORT=", "20010")
```

Prototype:

```
Declare Function T32Init Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

This function initializes the driver and establishes the connection to the TRACE32 display driver. If zero is returned, the connection was set up successfully.

Example:

```
Dim ret as Integer
ret = T32Init()
```

T32Exit Close Connection

Prototype:

```
Declare Function T32Exit Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

This function ends the connection to the TRACE32 display driver. This command should always be called before ending the application.

Example:

```
Dim ret as Integer
ret = T32Exit()
```

Prototype:

```
Declare Function T32Attach Lib "t32vbi.dll" (ByVal target As Long) _
As Integer
```

Parameters: `target` Device specifier

Returns: 0 for ok, otherwise Error value

This command attaches the control to the specified TRACE32 device. It is recommended to attach to `T32_DEV_ICD` immediately after `T32_Init()`, to have access to all API functions.

Constant	Value	Meaning
`T32_DEV_OS`	0	Basic operating system of the TRACE32 ("::"), disables all device specific commands (default)
`T32_DEV_ICD`	1	In Circuit Emulator ("`E::`") or In Circuit Debugger ("`B::`"), including Basic OS commands

Note: The public constant names were added to `t32vbi_decl.bas` in December 2008.

Example:

```
Dim ret as Integer
ret = T32Attach (1)
```

Prototype:

```
Declare Function T32Nop Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

Send an empty message to the TRACE32 display driver and wait for the answer.

Example:

```
Dim ret as Integer
ret = T32Nop ()
```

T32Ping Send ping message

Prototype:

```
Declare Function T32Ping Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

Sends a "ping" message to the TRACE32.

Example:

```
Dim ret as Integer
ret = T32Ping ()
```

Prototype:

```
Declare Function T32Stop Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

If a PRACTICE script file (*.cmm) is currently running, it is stopped. If an application is running in the target or ICE, it will not be affected by this command, i.e. it continues running.

Example:

```
Dim ret as Integer
ret = T32Stop ()
```

T32Cmd Execute PRACTICE command

Prototype:

```
Declare Function T32Cmd Lib "t32vbi.dll" (ByVal cmd As String) As Integer
```

Parameters: `cmd` PRACTICE command to execute

Returns: 0 for ok, otherwise Error value

With this function a PRACTICE command is started at the TRACE32. You can use any valid command.

Example:

```
Dim ret as Integer
ret = T32Cmd ( "Data.Set %Long 12200 033FFC00" )
```

Prototype:

```
Declare Function T32EvalGet Lib "t32vbi.dll" (ByRef result As Long) _
As Integer
```

Parameters: `result` pointer to variable to catch evaluation result

Returns: 0 for ok, otherwise Error value

Some of the PRACTICE commands and/or functions set a global variable to store return values, evaluation results or error conditions. The value is always specific to the command used. The function `T32EvalGet` reads this value.

Example:

```
Dim result As Long
Dim ret as Integer
Dim cmd As String
cmd = "EVAL 0x1234"
ret = T32Cmd (cmd)
ret = T32EvalGet (result)
MsgBox "Result of last PRACTICE command: "+result, vbInformation,_
"TRACE32"
```

Prototype:

```
Declare Function T32GetMessage Lib "t32vbi.dll" (ByRef message _
As String, ByRef mode As Long) As Integer
```

Parameters: `message` pointer to an array of 128 characters for the message
`mode` pointer to variable to catch the message mode

Returns: 0 for ok, otherwise Error value

Most PRACTICE commands write messages to the message line of TRACE32. This function reads the contents of the message line and the mode of the message.

"`message`" must be an user allocated character array of at least 128 elements.

The message modes are currently defined as following and can be combined:

Mode	Meaning
1	General Information
2	Error
8	Status Information
16	Error Information
32	Temporary Display
64	Temporary Information

Example:

```
Dim mode As Long
Dim msg As String
Dim ret As Integer

T32Cmd "print ""Hello World!"" "
msg = String(128, " ")
ret = T32GetMessage (msg, mode)
If mode = 1 Then
     MsgBox msg, vbInformation, "TRACE32"
End If
```

ICD/ICE VBI Functions

This chapter describes all function available with a debugging device of the TRACE32. See `T32Attach()` for how to specify a device.

T32GetState — Get state of debugger

Prototype:

```
Declare Function T32GetState Lib "t32vbi.dll" (ByRef state As Long) _
As Integer
```

Parameters: state pointer to variable to catch ICE state

Returns: 0 for ok, otherwise Error value

Use this function to get the main state of the TRACE32 device. `state` can have four different values:

0	System is down, not prepared for download or emulation.
1	System is halted, CPU makes no cycles (r.g. STOP instruction)
2	Emulation is stopped
3	Emulation is running

Example:

```
Dim state As Long
Dim ret As Integer

ret = T32GetState (state)
Select Case mode
     Case 0
          MsgBox "System is down.", vbInformation, "TRACE32"
     Case 1
          MsgBox "System is halted.", vbInformation, "TRACE32"
     Case 2
          MsgBox "System is stopped.", vbInformation, "TRACE32"
     Case 3
          MsgBox "System is running.", vbInformation, "TRACE32"
     Case Else
          MsgBox "Cannot get system information.", vbCritical, "TRACE32"
End Select
```

Prototype:

```
Declare Function T32GetCpuInfo Lib "t32vbi.dll" _
(ByRef cpu As String, ByRef fpu As Long, ByRef endian As Long, _
ByRef ctype As Long) As Integer
```

Parameters:
`cpu` pointer to a variable receiving the CPU name
`fpu` pointer to a variable receiving the FPU type
`endian` pointer to a variable receiving the byte order
`ctype` pointer to a long variable receiving additional internal information

Returns: 0 for ok, otherwise Error value

This function gives information about the CPU type. `cpu` will contain an ASCII string with the CPU type and family. It must be allocated by the user for 256 characters. `fpu` describes, whether a FPU is present or not. This is currently not used and always zero. `endian` describes the byte order of the CPU: zero means big endian (`12 34` becomes `1234`), otherwise little endian (`12 34` becomes `3412`). `ctype` is for internal information and useless to the user.

Example:

```
Dim state As Long
Dim cpu As String
Dim fpu, endian, ctype As Long
Dim ret as Integer
cpu = String(256, " ")
ret = T32GetCpuInfo (cpu, fpu, endian, ctype)
MsgBox "Cpu: "+cpu+" Fpu "+fpu+" Endian "+endian, vbInformation,
"TRACE32"
```

Prototype:

```
Declare Function T32GetRam Lib "t32vbi.dll" _
(ByRef start As Long, ByRef rend As Long, ByRef access As Integer) _
As Integer
```

Parameters:

`start`	pointer to variable containing and receiving start address
`rend`	pointer to variable receiving end address
`access`	pointer to variable containing the access type

Returns: 0 for ok, otherwise Error value

This function retrieves the memory mapping of the emulator. `start` specifies the first address to search for a memory block. A zero will force to search from beginning of the address space. After return, `start` contains the first address, at which the specified memory is mapped and `end` contains the last address of the continuously mapped block To get all mapped blocks, call `T32GetRam` again until `access == 0`. `access` must contain the access mode.
Currently there are two modes: 1 for Data RAM ("`D:`") and 2 for Program RAM ("`P:`").
If access contains zero after return, and no error occurred, then no (more) mapped memory was found. Otherwise `access` is not equal to zero (but changed!).

Example:

```
Dim start As Long
Dim rend As Long
Dim access As Integer
Dim ret as Integer

start = 0
access = 2
ret = T32GetRam (start, rend, access)
MsgBox "Ram area: "+start+" -- "+rend , vbInformation, "TRACE32"
```

Prototype:

```
Declare Function T32ResetCpu Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

Prepares the ICD/ICE for debugging/emulation. This is done by executing the PRACTICE commands `SYStem.UP` and `Register.RESet`. This function can also be used to get control after the target software has crashed.

Example:

```
Dim ret as Integer
ret = T32ResetCPU ()
```

Prototype:

```
Declare Function T32ReadMemory Lib "t32vbi.dll" (ByVal address As Long, _
ByVal access As Long, ByVal size As Long, ByRef buffer As Any) As Integer
```

Parameters:

`address`	target memory address to start read
`access`	memory access flags
`size`	number of bytes to read
`buffer`	pointer to buffer area for memory content

Returns: 0 for ok, otherwise Error value

Reads data from target memory. The size of the data block is not limited. The access flags define the memory access class and access method:

Bit 0..3:	Memory Class, 0=Data, 1=Code, other=undefined
Bit 6:	Set for emulation memory access (dual port access)

Example:

```
Dim address As Long
Dim access As Integer
Dim buffer(16) As Byte
Dim size As Integer
Dim ret As Integer

address = &H0
access = &H40
size = 4
ret = T32ReadMemory (address, access, size, buffer(0))
MsgBox "Memory content: "+buffer(0)+buffer(1)+buffer(2)+buffer(3), _
vbInformation, "TRACE32"
```

Prototype:

```
Declare Function T32WriteMemory Lib "t32vbi.dll" (ByVal address _
As Long, ByVal access As Long, ByVal size As Long, ByRef buffer As Any)_
As Integer
```

Parameters:

`address` target memory address to start write
`access` memory access flags
`size` number of bytes to write
`buffer` pointer to host buffer data area to write

Returns: 0 for ok, otherwise Error value

Writes data to target memory. The size of the data block is not limited. This function should be used to access variables and make other not time critical memory writes. The access flags define the memory access class and access method:

Bit 0..3:	Memory Class, 0=Data, 1=Code, other=undefined
Bit 6:	Set for emulation memory access (dual port access)
Bit 7:	Set to enable verify after write

Example:

```
Dim address As Long
Dim access As Integer
Dim buffer(16) As Byte
Dim size As Integer
Dim ret As Integer

address = &H0
access = &H40
size = 4
buffer(0) = &HAA
buffer(1) = &HBB
buffer(2) = &HCC
buffer(3) = &HDD
ret = T32WriteMemory (address, access, size, buffer(0))
```

Prototype:

```
Declare Function T32WriteMemoryPipe Lib "t32vbi.dll" _
(ByVal address As Long, ByVal access As Long, ByVal size As Long, _
ByRef buffer As Any) As Integer
```

Parameters:

`address` target memory address to start write
`access` memory access flags
`size` number of bytes to write
`buffer` pointer to host buffer data area to write

Returns: 0 for ok, otherwise Error value

Writes data to target memory with pipelining. Pipelinig means, that the memory write operation of the emulator is done in parallel to the downloading process. This speeds up the download. The return value of the function always refers to the previous Write command. The result of the last write command must be fetched by calling the function with size=0. The size of the data block is not limited. This function should be used to download an application program. The access flags define the memory access class and access method (see `T32WriteMemory`).

Example:

```
Dim address As Long
Dim access As Integer
Dim buffer(16) As Byte
Dim size As Integer
Dim ret As Integer

address = &H0
access = &H40
size = 4
buffer(0) = &HAA
buffer(1) = &HBB
buffer(2) = &HCC
buffer(3) = &HDD
ret = T32WriteMemoryPipe (address, access, size, buffer(0))
```

Prototype:

```
Declare Function T32ReadRegister Lib "t32vbi.dll" (ByVal mask1 As Long, _
ByVal mask2 As Long, ByRef buffer As Any) As Integer
```

Parameters: `mask1`, `mask2` Register addressing mask
`buffer` pointer to buffer receiving the register data

Returns: 0 for ok, otherwise Error value

The two 32-bit values `mask1` and `mask2` form a 64-bit bitmask. Each bit corresponds to one CPU register. Bit 0 of `mask1` is register #0, bit 31 of `mask2` is register #63. Registers are only read from the debugger/emulator, if their corresponding bit is set. The values of the registers are written in an array. Array element 0 is register 0, element 63 is register 63. The actual mapping from CPU register to bit is target-specific.

Example:

```
Dim mask1 As Long
Dim mask2 As Long
Dim buffer(64) As Long
Dim ret As Integer
mask1 = &H3FF
mask2 = 0
ret = T32ReadRegister ( mask1, mask2, buffer(0) )
; read the first 10 registers
```

Prototype:

```
Declare Function T32WriteRegister Lib "t32vbi.dll" (ByVal mask1 _
As Long, ByVal mask2 As Long, ByRef buffer As Any) As Integer
```

Parameters: `mask1`, `mask2` Register addressing mask
`buffer` pointer to buffer containing register data

Returns: 0 for ok, otherwise Error value

The two 32-bit values `mask1` and `mask2` form a 64-bit bitmask. Each bit corresponds with one CPU register. Bit 0 of `mask1` is register #0, bit 31 of `mask2` is register #63. Registers are only written, if their corresponding bit is set. The values of the registers are passed as an array. Array element 0 is register 0, element 63 is register 63.

Example:

```
Dim mask1 As Long
Dim mask2 As Long
Dim buffer(64) As Long
Dim ret As Integer

mask1 = &HA
mask2 = 0
buffer(1) = &HAAAAAAAA
buffer(3) = &HBBBBBBBB
ret = T32WriteRegister ( mask1, mask2, buffer(0))
```

Prototype:

```
Declare Function T32ReadPP Lib "t32vbi.dll" (ByRef pp As Long) _
As Integer
```

Parameters: `pp` pointer to variable receiving the program pointer value

Returns: 0 for ok, otherwise Error value

This function reads the current value of the program pointer. It is only valid, if the application is stopped, I.e. the state of the ICE is "Emulation stopped" (see `T32GetState`). The program pointer is a logical pointer to the address of the next executed assembler line. Contrary to `T32ReadRegister`, this function is processor independent.

Example:

```
Dim pp As Long
Dim ret As Integer
ret = T32ReadPP ( pp )
MsgBox "Current Program Pointer: "+pp, vbInformation, "TRACE32"
```

Prototype:

```
Declare Function T32ReadBreakpoint Lib "t32vbi.dll" (ByVal address _
As Long, ByVal access As Long, ByVal size As Long, ByRef buffer As Any) _
As Integer
```

Parameters:

`address` address to begin reading breakpoints
`access` memory access flags
`size` number of addresses to read
`buffer` pointer to buffer to catch breakpoint data

Returns: 0 for ok, otherwise Error value

Read breakpoint and flag information from emulator. The `access` variable defines the memory class and access method (see `T32ReadMemory`). The size of the range is not limited. The buffer contains 16-bit words in the following format:

Bit 0:	execution breakpoint (Program)
Bit 1:	HLL stepping breakpoint (Hll)
Bit 2:	spot breakpoint (Spot)
Bit 3:	read access breakpoint (Read)
Bit 4:	write access breakpoint (Write)
Bit 5:	universal marker a (Alpha)
Bit 6:	universal marker b (Beta)
Bit 7:	universal marker c (Charly)
Bit 8:	read flag (read), if mapped
Bit 9:	write flag (write), if mapped

Example:

```
Dim buffer(64) As Byte
Dim ret As Integer
ret = T32ReadBreakpoint &H81064, &HC0, 16, buffer(0)
```

Prototype:

```
Declare Function T32WriteBreakpoint Lib "t32vbi.dll" _
(ByVal address As Long, ByVal access As Long, ByVal breakpoint As Long, _
ByVal size As Long) As Integer
```

Parameters:
`address` address to begin writing breakpoints
`access` memory access flags
`breakpoint` breakpoints to set or clear in area
`size` number of addresses to write

Returns: 0 for ok, otherwise Error value

Set or clear breakpoints. The `access` variable defines the memory class and access method (see `T32WriteMemory`). The `size` of the range is not limited. The `breakpoint` argument defines, which breakpoints to set or clear over the memory area:

Bit 0:	execution breakpoint (Program)
Bit 1:	HLL stepping breakpoint (Hll)
Bit 2:	spot breakpoint (Spot)
Bit 3:	read access breakpoint (Read)
Bit 4:	write access breakpoint (Write)
Bit 5:	universal marker a (Alpha)
Bit 6:	universal marker b (Beta)
Bit 7:	universal marker c (Charly)
Bit 8:	Set to clear breakpoints

Example:

```
Dim ret As Integer
ret = T32WriteBreakpoint ( &H81064, &HC0, &H1, 16 )
```

Prototype:

```
Declare Function T32Step Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

Executes one single step on the emulator.

Example:s

```
Dim ret As Integer
ret = T32Step ()
```

Prototype:

```
Declare Function T32StepMode Lib "t32vbi.dll" (ByVal mode as Long) _
As Integer
```

Parameters: `mode` Stepping mode

Returns: 0 for ok, otherwise Error value

Executes one step on the emulator. The mode parameter controls the stepping mode:

0	assembler step
1	HLL step
2	mixed = assembler step with HLL display

Bit 7 of mode defines step into or step over a function call

Example:

```
Dim ret As Integer
ret = T32StepMode (&H81);
'Steps over a function call, halting on the next HLL line.
```

Prototype:

```
Declare Function T32Go Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

Start the target (ICD) or realtime emulation (ICE). The function will return immediately after the emulation has been started. The `T32GetState` function can be used to wait for the next breakpoint. All other commands are allowed while the emulation is running.

Example:

```
Dim ret As Integer
ret = T32Go ()
```

T32Break Stop realtime

Prototype:

```
Declare Function T32Break Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise Error value

Stops the realtime emulation asynchronously.

Example:

```
Dim ret As Integer
ret = T32Break ()
```

Prototype:

```
Declare Function T32GetSymbol Lib "t32vbi.dll" (ByVal symbol As String, _
ByRef address As Long, ByRef size As Long, ByRef access As Long) _
As Integer
```

Parameters:

`symbol`	pointer to symbol name
`address`	pointer to variable for the symbol address
`size`	pointer to variable for symbol size (if any)
`access`	pointer to variable for symbol access class

Returns: 0 for ok, otherwise communication error value.

This function returns the symbol information for a specified symbol name. If the specified symbol was not found, `address`, `size` and `access` contains `-1`.

NOTE:	It is not possible to get symbol information of non-static local variables -- they have no fixed address but are dynamically stored on the stack or in registers.

This function can also be used to get the address of a source line.

Example:

```
Dim ret As Integer
Dim addr As Long
Dim size As Long
Dim access As Long
Dim state As Byte
Dim buffer(64) As Byte
' get information about a variable
ret = T32GetSymbol ("flags", addr, size, access )
MsgBox "flags is located at: "+Hex(addr)+" size "+size, vbInformation, _
 "TRACE32"

' get information about line 12 in file.c
ret = T32GetSymbol ("\\file\\12", addr, size, access )
MsgBox "file.c line 12 is located at: "+Hex(addr)+" size "+size, _
vbInformation, "TRACE32"
```

Prototype:

```
Declare Function T32GetSource Lib "t32vbi.dll" (ByVal address As Long, _
ByRef filename As String, ByRef line As Long) As Integer
```

Parameters:

`address`	address from which to get the source
`filename`	pointer to an array of characters to catch the filename
`line`	pointer to variable receiving the source line

Returns: 0 for ok, otherwise Error value

With a given target address, this function calculates and gets the according source filename and source line. `filename` **must** be an array of characters with at least 256 elements.

Example:

```
Dim ret As Integer
Dim msg As String
Dim addr As Long
Dim line As Long

msg = String(256, " ")
ret = T32ReadPP (addr)  ' get program pointer
ret = T32GetSource (addr, msg, line)
MsgBox "Current source line: "+msg+" line "+line, vbInformation,
"TRACE32"
```

Prototype:

```
Declare Function T32GetSelectedSource Lib "t32vbi.dll" _
( ByRef filename As String, ByRef line As Long) As Integer
```

Parameters: `filename` pointer to an array of characters to catch the filename
line pointer to variable to catch the source line

Returns: 0 for ok, otherwise Error value

This function requests the source filename and line number of a selected source line in the TRACE32. The source line can be selected in any TRACE32 window containing source (e.g. "`a.l`" or "`d.l`"). If no previous selection was done, or if no source line is selected, the function returns with `filename` pointing to a NULL string.
`filename` **must** be an array of characters with at least 256 elements.

Example:

```
Dim ret As Integer
Dim msg As String
Dim line As Long

msg = String(256, " ")
ret = T32GetSelectedSource (msg, line)
MsgBox "Current source line : "+msg+" line "+line, vbInformation,_
 "TRACE32"
```

Prototype:

```
Declare Function T32AnaStatusGet Lib "t32vbi.dll" (ByRef state As Byte, _
ByRef size As Long, ByRef min As Long, ByRef max As Long) As Integer
```

Parameters:

`state`	pointer to variable for the current analyzer state
`size`	pointer to variable to catch the trace buffer size
`min`	pointer to variable to catch the minimum record number
`max`	pointer to variable to catch the maximum record number

Returns: 0 for ok, otherwise communication error value

This function requests the state of the TRACE32 State Analyzer.

"`state`" contains the current analyzer state:

0	analyzer is switched off
1	analyzer is armed
2	analyzer recording breaked

`size` contains the trace buffer size. It specifies the amount of records, which can be recorded, **not** the amount of records, which are actually stored in the buffer.

`min`, `max` contain the minimum and the maximum record number stored in the trace buffer. Note that the record numbers can be negative or positive.

Example:

```
Dim ret As Integer
Dim min As Long
Dim max As Long
Dim size As Long
Dim state As Byte
Dim buffer(64) As Byte
ret = T32AnaStatusGet (state, size, min, max)
MsgBox "Analyzer state : "+state+" size "+size+" Area: "+min+" -- "+max,
_
vbInformation, "TRACE32"
ret = T32AnaRecordGet -3, 16, buffer(0)
```

Prototype:

```
Declare Function T32AnaRecordGet Lib "t32vbi.dll" (ByVal recordnr _
As Long, ByVal length As Long, ByRef buffer As Any) As Integer
```

Parameters:
`recordnr` record number of record to read
`buffer` byte array to catch the record information
`length` number of bytes to read from record

Returns: 0 for ok, otherwise communication error value

This function reads the record information of one record of the Analyzer trace buffer.

`recordnr` specifies the record number to read. `buffer` contains the read record information (see below). `length` specifies the number of bytes to read from the information into the buffer. This can be used to limit the amount of bytes transmitted and written into the buffer. If you specify “0”, all information will be transmitted; in this case allocate an array with 256 bytes at least.

The buffer will then contain the following data:

Index	Content
0	return value: 0=Ok -1=no analyzer present -2=invalid record number
1	reserved
2	physical access class: bit0 = Data bit1 = Program bit2 = First Cycle bit3 = (reserved) bit4 = Breakpoint Cycle bit5 = (reserved) bit6 = Write Cycle bit7 = Opfetch1 Cycle
3	reserved
4-7	physical address (little endian)
8-15	bus data (max. 8 bytes, depending on bus data width)
16	bus data width

17	bus access cycle (read/write/fetch, processor dependant)
18-19	status lines, processor dependant
20-27	time stamp (one bit equals 20/256 ns)
28/29	external trigger A/B inputs
30	logical access class: 1=Data 2=Program
31	reserved
32-35	logical address
rest	reserved

Example:

```
Dim ret As Integer
Dim buffer(64) As Byte
ret = T32AnaRecordGet -3, 4, buffer(0)
MsgBox "Analyzer Record -13: "+buffer(0)+buffer(1)+buffer(2)+buffer(3) _
, vbInformation, "TRACE32"
```

Functions for using the API with Multiple Debuggers

A single API instance can be used with several TRACE32 debuggers (e.g. for Multi-Core debugging) by creating a communication channel to each of the debuggers. Instead of passing the channel as parameter to API calls, the whole API is switched to a specific channel via `T32SetChannel()`.

A channel is created by allocating the required amount of memory, initializing this memory with channel defaults, and activating it via `T32SetChannel()`. Then use `T32Config()`, `T32Init()` and `T32Exit()` as would be done on the default channel. The default channel is the only one with a given channel number, its channel number is zero (0). As soon as additional to default channel zero (0) a new channel is created, at the end of the session the allocated memory must be freed with T32ExitAllChannels.

Note that despite the channel concept, each debugger must be assigned a unique PORT address in its configuration file (e.g. config.t32), and in order to communicate with it `T32Config()` must be used to set the same port number in the API.

The functions for using the API with multiple debuggers were added to `t32vbi.dll` and `t32vbi_decl.bas` in December 2008.

T32SetChannel — Set Channel to TRACE32 Instance

Prototype:

```
Declare Function T32SetChannel Lib "t32vbi.dll" (ByVal channel As Long) _
As Integer
```

Parameters: `channel` number of communication channel to set

Returns: 0 for ok, otherwise communication error value

This function switches to a set of communication parameters (a different HOST/PORT setting) to allow debugging with more than one TRACE32 instance in the same debug session, e.g. for multi-core debugging, for debugging and parallel logging, for bus analysis, etc.

Subsequent calls to `T32Config()`, `T32Init()` and `T32Exit()` will deal with the currently active channel.

Channel 0 (zero) is the default channel and automatically selected at application start. It does not need additional memory allocations. If you use any other channel number, the first invocation of `T32SetChannel()` for the new channel number allocates and initializes a new memory block. At the end of the debug session, all allocated memory blocks must be freed with `T32ExitAllChannels()`.

Tip: Note that you can make connections to different TRACE32 instances sequentially on one channel. `T32SetChannel()` is only necessary if two different sessions need to be open at the same time.

Prototype:

```
Declare Function T32ExitAllChannels Lib "t32vbi.dll" () As Integer
```

Parameters: none

Returns: 0 for ok, otherwise communication error value

This function cleans up and frees any additional memory blocks that were allocated by `T32SetChannel()` (with an argument different from 0). Note that `T32Exit()` is internally called before the memory is freed.

Channel 0 is the default channel and does not need additional memory allocations. If you use any other channel, the allocated memory must be freed with `T32ExitAllChannels()` when you are done with all channels. In this case, `T32ExitAllChannels()` is equivalent to `T32Exit()`.

Example:

```
Dim ret as Integer
Dim chan as Long

chan = 5  'anything except zero

ret = T32SetChannel(chan) 'now we create a new channel
ret = T32Config("HOST=", "localhost")
ret = T32Config("PORT=", "20000") 'first TRACE32 instance
ret = T32Init()
ret = T32Cmd("SYStem.Up")

ret = T32SetChannel(0) 'select default channel
ret = T32Config("HOST=", "localhost")
ret = T32Config("PORT=", "20006") ' second TRACE32 instance
ret = T32Init()
ret = T32Cmd("SYStem.Up")
ret = T32Cmd("Data.List 0x1000")

ret = T32SetChannel(chan) 'talk to first instance again
ret = T32Cmd("do test.cmm")

ret = T32ExitAllChannels()  'exit and clean up everything
```

Version Control

Document version control:

Version	Date	Change
1.0	18.01.2000	Initial revision
1.1	24.02.2000	Changes for compatibility with Windows 98
1.2	10.12.2008	Channel description added, named attach constants, cleanup